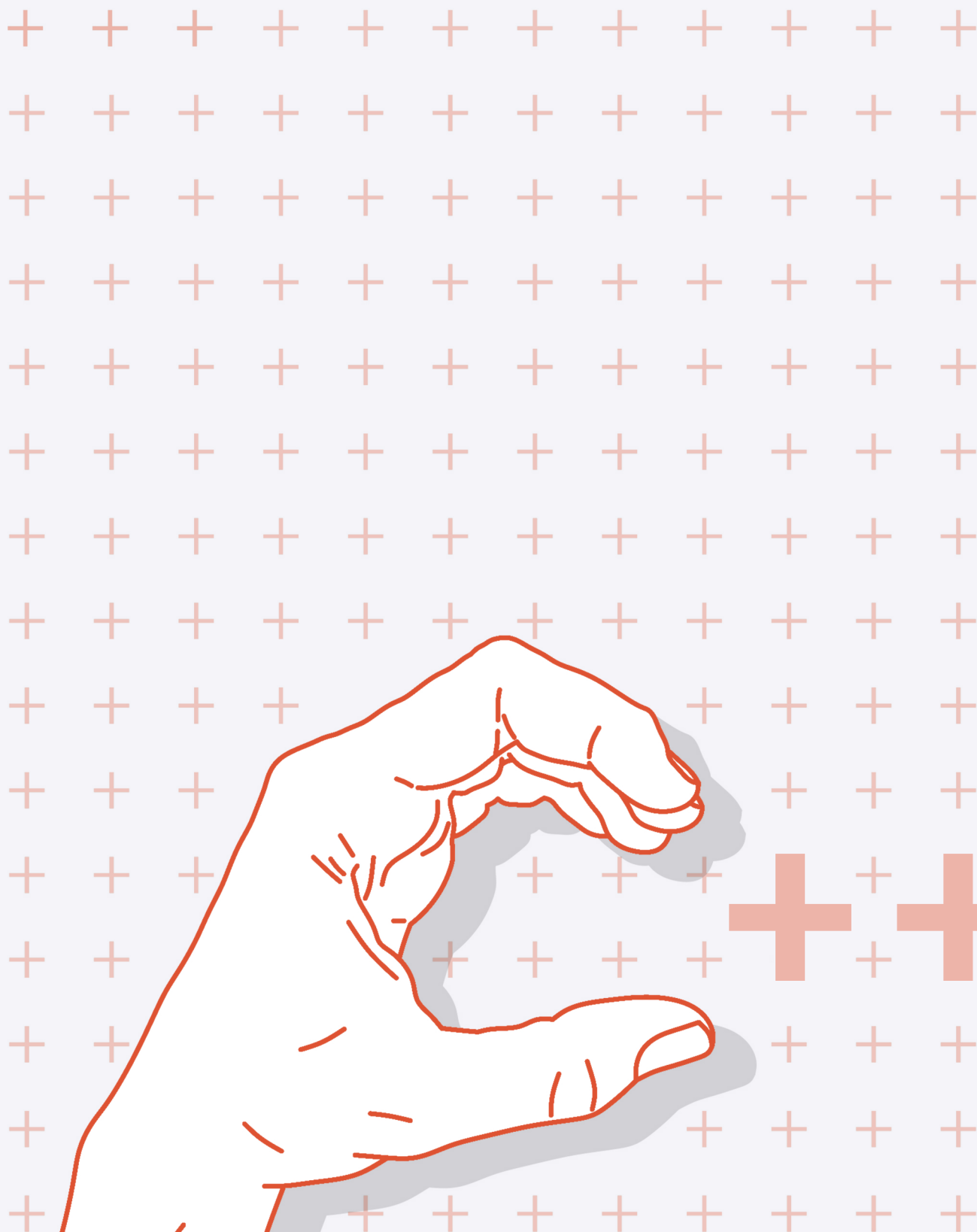


Методичка збирка задатака из основа програмирања - C++



 Petlja

*Друштво математичара Србије
Фондација Пејља*

Методичка збирка задатака из основа програмирања — С++

Београд, 2019.

Аутори:

Филић Марић, професор на Математичком факултету у Београду

Нина Алимџић, професор у Математичкој гимназији у Београду

Небојша Васиљевић, фондација „Петља”

Милан Вујделија, фондација „Петља”

Ђуша Вуковић, професор у Математичкој гимназији у Београду

Мијодраг Ђуришић, професор у Математичкој гимназији у Београду

Весна Маринковић, доцент на Математичком факултету у Београду

Сјанка Мајковић, професор у Математичкој гимназији у Београду

Јелена Хаџи-Пурић, асистент на Математичком факултету у Београду

Милан Чабаркаја, професор у Математичкој гимназији у Београду

Методичка збирка алгоритамских задатака са решењима — С++
основни ниво

Издавачи: Друштво математичара Србије и Фондација „Петља”

За издавача: *др Војислав Андрић*

Обрада текста и илустрације: *аутори*, Дизајн корица: *Иван Авдић*

Издање: 1.1 (електронско)

Садржај

Предговор	1
О збирци	1
Препоруке читаоцу који почиње да учи програмирање	1
1 Аритметика	3
1.1 Елементи програмског језика	3
1.1.1 Основна структура програма	3
1.1.2 Коментари	4
1.1.3 Испис, читавање, променљиве, типови	4
1.1.4 Основне аритметичке операције и изрази	7
1.1.5 Уграђене математичке функције	8
1.1.6 Именоване константе	9
1.1.7 Дефинисање функција	9
1.1.7.1 Више повратних вредности	10
1.2 Елементарна целобројна и реална аритметика	10
1.2.1 Геометријске формуле	10
Задатак: Тренинг	11
Задатак: Лист папира	11
Задатак: Столњак	12
Задатак: Фудбалски терен	13
Задатак: Растојање тачака	13
Задатак: Површина троугла датих темена	14
Задатак: Ниво базена	16
Задатак: Правоугаоник дат наспрамним теменима	16
Задатак: Троугао одређен правом и координатним почетком	16
1.2.2 Линеарне једначине и системи линеарних једначина	16
Задатак: Ограда терасе	17
Задатак: Збирови по три странице правоугаоника	18
Задатак: Просек на такмичењу	18
1.2.3 Линеарна зависност и пропорција	19
Задатак: Подела интервала	19
Задатак: Генератор случајних бројева	19
Задатак: Група радника	20
Задатак: Такси	21
Задатак: Курс	21
1.2.4 Степеновање и кореновање	21
Задатак: Степен и корен	21
1.2.5 Кретање	22
Задатак: Путовање	22
Задатак: Бициклиста	23
Задатак: Сустизање аутомобила	23
Задатак: Растојање кућа	24
Задатак: Колона	24
Задатак: Браћа и пас	24
1.3 Целобројно дељење	25
1.3.1 Појам целобројног количника и остатка	25

	Задатак: Разломак у мешовит број	25
	Задатак: Врста и колона	25
	Задатак: Шаховска табла број црних поља	26
	Задатак: Икс-окс	26
	Задатак: По два и по три	27
1.3.2	Позициони запис	28
	Задатак: Збир цифара четвороцифреног броја	28
	Задатак: Јарди, стопе и инчи	29
	Задатак: Октални бројеви	30
	Задатак: Избаци цифру стотина	31
	Задатак: Поноћ	32
	Задатак: Трајање вожње	34
	Задатак: UNIX време	34
	Задатак: Угао сатне казаљке	35
	Задатак: Угао између казаљки	37
	Задатак: Размени цифре	38
	Задатак: Бројање оваца	38
	Задатак: Обрни цифре	38
	Задатак: Цифре сдесна	39
	Задатак: Чекање	39
	Задатак: Тркачи	39
	Задатак: Радијани	39
	Задатак: Гпс	40
	Задатак: Поклапање казаљки	40
2	Гранање	41
2.1	Елементи програмског језика	41
2.1.1	Релацијски оператори	41
2.1.2	Логички оператори	41
2.1.3	Наредба if	42
2.1.4	Условни израз	42
2.2	Једноставно гранање	43
2.2.1	Релацијски оператори	43
	Задатак: Јабуре	43
	Задатак: Збир година браће и сестре	43
	Задатак: Теме правоугаоника	44
	Задатак: Једнакостранични троугао датог обима	45
2.2.2	Логички оператори	45
	Задатак: Радно време	45
	Задатак: Кућни ред	46
	Задатак: Постоји ли троугао датих дужина страница	46
	Задатак: Преступна година	47
	Задатак: Два броја истог знака	48
	Задатак: Исти квадрант	48
	Задатак: Тачка у правоугаонику и кругу	49
	Задатак: Статус објављен током школе	49
	Задатак: Да ли се две даме нападају	50
2.3	Угнежђено гранање	50
2.3.1	Елементи програмског језика	50
2.3.2	Гранање на основу припадности интервалима	52
	Задатак: Агрегатно стање	52
	Задатак: Успех ученика	53
	Задатак: Одмор на пола пута	54
	Задатак: Растојање од тачке до дужи	55
	Задатак: Школарина	57
	Задатак: Солидарни порез	57
	Задатак: Правоугаони прстен	57
	Задатак: Оцена на испиту	58

2.3.3	Гранање на основу коначног скупа вредности	58
	Задатак: Редни број месеца	58
	Задатак: Број дана у месецу	59
	Задатак: Назив месеца	60
2.3.4	Лексикографско поређење торки	60
	Задатак: Пунолетство	60
	Задатак: Бољи у две дисциплине	62
	Задатак: Ко је пре стигао	64
2.3.5	Хијерархија услова	64
	Задатак: Два броја истог знака	64
	Задатак: Линеарна једначина	64
	Задатак: Икс-окс	65
	Задатак: Квадранти и осе	66
	Задатак: Размакнуте дијагонале	67
	Задатак: Положај две праве	67
2.3.6	Минимум и максимум два броја	67
	Задатак: Куглице	68
	Задатак: Проја	69
	Задатак: Скупови спортиста	69
	Задатак: Домине	70
	Задатак: Интервали	71
	Задатак: Позитиван део интервала	72
	Задатак: Део правоугаоника у првом квадранту	73
	Задатак: Темена правоугаоника	74
	Задатак: H_2SO_4	74
	Задатак: Сијалице	75
	Задатак: Краљево растојање	75
	Задатак: Део квадрата у првом октанту	75
2.3.7	Разни задаци са гранањем	76
	Задатак: Сутрашњи датум	76
	Задатак: Јучерашњи датум	77
	Задатак: Врста троугла на основу углова	77
	Задатак: Врста троугла на основу страница	77
	Задатак: Тенис	78

3 Итерација 79

3.1	Основни алгоритми над малим серијама елемената	79
3.1.1	Ажурирање вредности променљивих	79
	Задатак: Цена хлеба	81
	Задатак: Сутрашњи датум	82
	Задатак: Подела јабука	82
	Задатак: Јучерашњи датум	84
3.1.2	Основне статистике (збир, производ, просек, минимум, максимум)	84
	Задатак: Најлошији контролни	87
	Задатак: Најтоплији дан	89
	Задатак: Најбољи у две дисциплине	90
	Задатак: Најбољи такмичари	91
	Задатак: Најјефтинији за динар	93
	Задатак: Просек скијаша	93
	Задатак: Врста троугла на основу углова	93
	Задатак: Растојање тачка правоугаоник	93
	Задатак: Позиција највећег угла троугла	94
	Задатак: Најближи просеку	94
	Задатак: Први и други максимум	94
	Задатак: Аутомобил у вођству	95
3.1.3	Филтрирање	95
	Задатак: Јутарње температуре	95
	Задатак: Квалификациони праг	96

	Задатак: Претицање	97
	Задатак: Берза	98
3.1.4	Претрага	99
	Задатак: Чудно радно време	100
	Задатак: Постоји ли троугао датих дужина страница	101
	Задатак: Статус објављен током школе	101
	Задатак: Непливачи	101
3.1.5	Сортирање и примене	102
	Задатак: Уреди три броја	103
	Задатак: Најлошији контролни	104
	Задатак: Подела бројевне праве на интервале	104
	Задатак: Најмањи троцифрени број	105
	Задатак: Једнакокраки од 4 дужи	106
	Задатак: Прозор и кутија	108
	Задатак: Сумо рвачи	109
	Задатак: Одсутна	111
	Задатак: Најјефтинији за динар	111
	Задатак: Први и други максимум	111
	Задатак: Најбољи такмичари	112
	Задатак: Оријентација троугла	112
3.1.6	Позициони запис - вредност и цифре	112
	Задатак: Збир цифара четвороцифреног броја	113
	Задатак: Октални бројеви	113
	Задатак: UNIX време	114
	Задатак: Збир два троцифрена	115
	Задатак: Цифре сдесна	116
	Задатак: Избаци цифру	117
	Задатак: Бројање оваца	119
	Задатак: Обрни цифре	119
	Задатак: Поноћ	119
	Задатак: Тајмер	119
3.1.7	Позициони запис - сабирање и одузимање	119
	Задатак: Збир два троцифрена	119
	Задатак: Време завршетка филма	121
	Задатак: Тајмер	122
	Задатак: Поноћ	122
	Задатак: Трајање вожње	122
3.2	Основни алгоритми над серијама елемената	122
3.2.1	Елементи програмског језика	122
	3.2.1.1 Петља while	122
	3.2.1.2 Петља for	122
	3.2.1.3 Петља do-while	123
	3.2.1.4 Прекиди петље (break и continue)	123
3.2.2	Итерација кроз правилне серије бројева	123
	Задатак: Бројеви од а до b	123
	Задатак: Бројање у игри жмурке	124
	Задатак: Троцифрени парни бројеви	125
	Задатак: Одбројавање уназад	126
	Задатак: Најаве емисије у правилним временским интервалима	127
	Задатак: Подела интервала на једнаке делове	127
	Задатак: Геометријска серија	129
3.2.3	Учитавање серија бројева	129
	Задатак: Збир n бројева	130
	Задатак: Читање до нуле	130
	Задатак: Читање до краја улаза	131
	Задатак: Читање до -1 или до n-тог броја	132
3.2.4	Пресликавање серије бројева	133
	Задатак: Прерачунавање миља у километре	133

	Задатак: Табелирање функције	134
3.2.5	Елементарне статистике (број елемената, збир, производ, просек, средине, ...)	135
	Задатак: Факторијел	135
	Задатак: Степен	136
	Задатак: Просек свих бројева до краја улаза	136
	Задатак: Средине	137
	Задатак: Просечан раст цена	138
	Задатак: Производња малина	139
	Задатак: Сума низа бројева	139
	Задатак: Једнакост растојања	140
	Задатак: Тежиште	140
3.2.6	Филтрирање серије бројева	141
	Задатак: Бројеви дељиви са 3	141
	Задатак: Бројање гласова за омиљеног глумца	142
	Задатак: Просек одличних	143
	Задатак: Категорије цудиста	144
	Задатак: Статистике33	145
	Задатак: Секција	145
	Задатак: Купци	146
3.2.7	Минимум и максимум	146
	Задатак: Најнижа температура	146
	Задатак: Најтоплији дан	147
	Задатак: Просек скокова	148
	Задатак: Најближи датом целом броју	149
	Задатак: Број максималних	150
	Задатак: Други на ранг листи	151
	Задатак: Друга вредност по величини	154
	Задатак: Победник у три дисциплине	156
	Задатак: Најмањи круг	158
	Задатак: Редни број максимума	158
3.2.8	Линеарна претрага	158
	Задатак: Негативан број	159
	Задатак: Дељив бројевима од 1 до n	161
	Задатак: Прва негативна температура	162
	Задатак: Последња негативна температура	164
	Задатак: Парно непарни	164
	Задатак: Први и последњи приступ	165
	Задатак: Провера тробојке	166
3.2.9	Позициони запис броја	166
	Задатак: Број и збир цифара броја	166
	Задатак: Да ли запис броја садржи цифру	168
	Задатак: Различите цифре	169
	Задатак: Армстронгов број	170
	Задатак: Трансформација броја у производ цифара	172
	Задатак: Најмањи број са највећим збиром парних цифара	173
	Задатак: Број формиран од датих цифара с лева на десно	177
	Задатак: Број формиран од датих цифара здесна на лево	178
	Задатак: Замени цифре 0 са 5	178
	Задатак: Децимале броја $1/n$	179
	Задатак: K децимала у бинарном запису	181
	Задатак: Бројеви који не садрже цифру 5	182
	Задатак: Комбинација два броја минимумом и максимумом одговарајућих цифара	182
3.2.10	Однос суседних елемената серије	182
	Задатак: Максимална разлика суседних	182
	Задатак: Провера монотоности	184
	Задатак: Парно непарни	185
	Задатак: Растуће цифре	185
	Задатак: Поређани датуми	186

	Задатак: Тестераст низ	187
	Задатак: Продужавање титлова	190
3.3	Угнежђене петље	191
3.3.1	Генерисање свих могућности (варијације, комбинације, пермутације, партиције)	191
	Задатак: Бројеви у датој основи	192
	Задатак: Варијације тројки	193
	Задатак: Мали лото	193
	Задатак: Коцкице за јамб	194
	Задатак: Комбинације поена	195
	Задатак: Цикличне пермутације	196
	Задатак: Троуглови целобројних страница, задатог обима	197
3.3.2	Сви подсегменти серије	197
	Задатак: Сви суфикси низа бројева од 1 до n	197
	Задатак: Сви префикси низа бројева од 1 до n	198
	Задатак: Све подречи	198
	Задатак: Све подречи дужине n	199
	Задатак: Све подречи по опадајућој дужини	199
	Задатак: Цикличне подречи	200
3.3.3	Цртежи помоћу карактера	200
	Задатак: Квадрат од звездица	200
	Задатак: Троугао од звездица	201
	Задатак: Троугао од речи	202
	Задатак: Ромб од звездица	203
	Задатак: Ћилим од звездица	203
	Задатак: V од звездица	203
3.3.4	Генерисање неких правилних серија	204
	Задатак: Карирана застава	204
	Задатак: Серије 123	205
	Задатак: Таблица множења	205
	Задатак: Серије непарни парни	205
4	Детаљнији преглед основних типова података	206
4.1	Рад са целим бројевима	206
4.1.1	Распон целобројних типова	206
	Задатак: Множење великих бројева	207
	Задатак: Јабуре у троуглу	207
	Задатак: Средина интервала	208
	Задатак: Провера прекорачења	209
	Задатак: Факторијел 64бит	209
4.1.2	Модуларна аритметика	210
	Задатак: Операције по модулу	210
	Задатак: Монопол	211
	Задатак: Сат	212
	Задатак: Збир бројева по модулу	214
	Задатак: Разбрајалица	214
	Задатак: Жмурке	214
	Задатак: Трајање вожње кроз два дана	214
	Задатак: Навијање сата	215
	Задатак: Време завршетка филма	215
	Задатак: Аутобус	215
	Задатак: Точак среће	215
4.2	Везе између целих и реалних бројева	216
4.2.1	Заокруживање реалних бројева	216
	Задатак: Пертла	217
	Задатак: Точак	217
	Задатак: Скалирање	218
	Задатак: Кружна мета	219
	Задатак: Успех ученика	219

4.2.2	Проблеми тачности записа реалних бројева	219
	Задатак: Динари и паре	220
	Задатак: Лимунада	221
4.2.3	Заокруживање количника целобројним дељењем	222
	Задатак: Поклони	222
	Задатак: Лифт	223
	Задатак: Оцена на испиту	224
	Задатак: Крушке	225
	Задатак: Планинари	225
	Задатак: Сечење плочица	227
	Задатак: Папир	228
	Задатак: Кофа	229
	Задатак: Воћни пакети	229
	Задатак: Дељиви око броја	230
	Задатак: Број дељивих у интервалу	230
	Задатак: Судоку	230
	Задатак: Магазин сокова	231
	Задатак: H ₂ SO ₄	231
4.3	Карактерски тип података	232
	Задатак: Класификација карактера	232
	Задатак: Трансформација карактера	233
	Задатак: Абецедно огледало	234
4.4	Набројиви тип	235
	Задатак: Врста троугла на основу углова	235
	Задатак: Линеарна једначина	237
	Задатак: Школарина	238
	Задатак: Положај две праве	238
	Задатак: Берза	238
4.5	Структурни тип	238
	Задатак: Разломци	238
	Задатак: Бољи у две дисциплине	239
4.6	Торке	240
	Задатак: Пунолетство	241
	Задатак: Бољи у две дисциплине	241
	Задатак: Најбољи у две дисциплине	242
	Задатак: Поређани датуми	242
5	Низови, ниске, матрице	243
5.1	Једнодимензионалне колекције података	243
5.1.1	Елементи програмског језика	243
	5.1.1.1 Статички алоцирани низови	243
	5.1.1.2 VLA	244
	5.1.1.3 Динамички алоцирани низови	244
	5.1.1.4 Библиотеке колекције (vector, array)	244
5.1.2	Елементарно коришћење низова	246
	Задатак: Испис у обратном редоследу	246
	Задатак: Линије у обратном редоследу	246
	Задатак: Најнижа температура	247
	Задатак: Средине	248
	Задатак: Минимално одступање од просека	249
	Задатак: Ниска палиндром	250
	Задатак: Погођена комбинација	251
	Задатак: Парни и непарни елементи	252
	Задатак: Редни број максимума	255
	Задатак: Разлика сума до мах и од мах	255
	Задатак: Просечно одступање од минималног	255
	Задатак: Максимална разлика суседних	256
	Задатак: Палиндромска реченица	256

	Задатак: Провера монотоности	256
	Задатак: Поређани датуми	256
	Задатак: Прерачунавање миља у километре	256
5.1.3	Трансформације низова	256
	Задатак: Транслација тачака	256
	Задатак: Избацивање елемената	258
	Задатак: Различити елементи низа	260
	Задатак: Обртање низа	262
	Задатак: Циклично померање за једно место	262
	Задатак: Необрисани бројеви	263
5.1.4	Низови као репрезентација вектора, полинома и великих бројева	263
	Задатак: Скаларни производ	263
	Задатак: Вредност полинома	264
	Задатак: Аритметика над полиномима	265
	Задатак: Збир два троцифрена	267
	Задатак: Линеарна функција над великим бројевима	268
5.1.5	Библиотечке функције за рад са низовима	270
	Задатак: Бројеви од a до b	271
	Задатак: Збир n бројева	272
	Задатак: Просек свих бројева до краја улаза	272
	Задатак: Факторијел	272
	Задатак: Најнижа температура	273
	Задатак: Претицање	273
	Задатак: Најтоплији дан	274
	Задатак: Минимално одступање од просека	274
	Задатак: Максимална разлика суседних	275
	Задатак: Прерачунавање миља у километре	276
	Задатак: Транслација тачака	276
	Задатак: Магацин сокова	277
	Задатак: Средине	277
	Задатак: Бројеви дељиви са 3	278
	Задатак: Просек одличних	278
	Задатак: Парни и непарни елементи	279
	Задатак: Избацивање елемената	279
	Задатак: Бројање гласова за омиљеног глумца	280
	Задатак: Број максималних	280
	Задатак: Различити елементи низа	281
	Задатак: Први и последњи приступ	281
	Задатак: Негативан број	282
	Задатак: Дељив бројевима од 1 до n	282
	Задатак: Парно непарни	283
	Задатак: Погођена комбинација	283
	Задатак: Ниска палиндром	284
	Задатак: Обртање низа	284
	Задатак: Бројање у игри жмурке	285
	Задатак: Једнакост растојања	285
	Задатак: Степен	285
	Задатак: Просечан раст цена	285
	Задатак: Редни број максимума	285
	Задатак: Најмањи круг	286
	Задатак: Победник у три дисциплине	286
	Задатак: Разлика сума до мах и од мах	286
	Задатак: Просечно одступање од минималног	286
	Задатак: Најближи просеку	286
	Задатак: Берза	286
	Задатак: Палиндромска реченица	286
	Задатак: Прва негативна температура	286
	Задатак: Последња негативна температура	286

5.1.6	Сортирање	286
	Задатак: Сортирање бројева	286
	Задатак: Сортирање такмичара	289
	Задатак: Просек скокова	294
	Задатак: Највреднији предмети	294
	Задатак: Сортирање на основи растојања од О	295
5.2	Карактери и ниске	295
5.2.1	Елементи програмског језика	295
5.2.2	Подниске	297
	Задатак: Презиме па име	297
	Задатак: Све подречи	297
	Задатак: Подниска	298
	Задатак: Подниске	300
	Задатак: Реч Франкенштајн	301
	Задатак: Избацивање подниски	302
	Задатак: Подела линије на речи	303
5.2.3	Поређење ниски	304
	Задатак: Лексикографски поредак	304
	Задатак: Лексикографски поредак без разлике између великих и малих слова	305
	Задатак: Лексикографски минимум	306
5.2.4	Трансформисање ниски	307
	Задатак: Цезаров кôд	307
5.2.5	Запис бројева и бројевних израза	309
	Задатак: Бројевне основе	309
	Задатак: Spreadsheet колоне	312
	Задатак: Вредност једноставног израза	313
	Задатак: Арапски у римски	315
	Задатак: Римски у арапски	316
5.3	Пресликавања	317
5.3.1	Класични низови у функцији пресликавања	317
	Задатак: Назив месеца	317
	Задатак: Успех ученика	318
	Задатак: Различите цифре	318
	Задатак: Да ли се даме нападају	318
	Задатак: Фреквенција знака	319
	Задатак: Изоморфне ниске	320
	Задатак: Број дана у месецу	321
	Задатак: Хистограм	321
	Задатак: Двојке и тројке дељиве са 3	322
	Задатак: Најчешће мало и велико слово	322
5.3.2	Асоцијативни низови (мапе, речници)	323
	Задатак: Редни број месеца	324
	Задатак: Римски у арапски	324
	Задатак: Изоморфне ниске	324
	Задатак: Фреквенција знака	325
	Задатак: Фреквенције речи	326
	Задатак: Миноморија	327
	Задатак: Телефонски именик	327
5.4	Вишедимензиони низови и матрице	328
5.4.1	Елементи програмског језика	328
	5.4.1.1 Матрице	328
	5.4.1.2 Разуђени низови	328
5.4.2	Матрице	329
	Задатак: Број бомби у околини	329
	Задатак: Број правоугаоника на слици	330
	Задатак: Пресликавања матрице	332
	Задатак: Да ли се даме нападају	335
	Задатак: Највећа дијагонала	337

	Задатак: Спирални испис матрице	339
	Задатак: Множење матрица	341
	Задатак: Релација зависности	344
	Задатак: Преседање	346
	Задатак: Особине релације	346
	Задатак: Асоцијативност	347
	Задатак: Гарантовани износ	347
	Задатак: Осигурање	348
	Задатак: Магичан квадрат	348
	Задатак: Матрица 1248	349
	Задатак: Премештаљка	349
	Задатак: Троуглови и широка дијагонала	349
	Задатак: Врхови планине	350
	Задатак: Турнир	350
	Задатак: Потез у игри 2048	351
	Задатак: Сортирање по просеку	351
5.4.3	Разуђени низови	352
	Задатак: Електронски дневник	352

Предговор

О збирци

Ова збирка се бави елементарним алгоритмима и погодна је за почетни ниво учења програмирања у гимназијама, стручним школама и на факултетима, као и почетни ниво припреме за такмичења из програмирања.

Потребно математичко предзнање не прелази ниво основне школе, а математички појмови и формуле које се користе у решењима су резимирани пре сваког поглавља. Са друге стране, постоје математички појмови који се недовољно разрађују у класичној настави математике, а који су важни у програмирању (и рачунарству уопште), па су они кроз ову збирку нешто детаљније разрађени. Такве су, на пример, област целобројног дељења, модларне аритметике, позиционог записа бројева и слично.

Изолагање је систематизовано по коришћеним идејама (алгоритмима), пратећи аналогичне између тих идеја и постепено повећавајући сложеност. Технички детаљи који одвлаче пажњу од идеја су одложени колико је могуће, да би се олакшало савладавање основних алгоритама и што пре почело решавање задатака. У складу са тиме, програмски језик C++, на коме су написана сва решења, уводи се само у мери која је довољна за решавање постављених задатака (преглед језика је у другом плану). На пример, у већем делу збирке за представљање целих бројева се користи само тип `int`, (а детаљнији опис целобројних типова и питања прекорачења су одложени); бројчани подаци се читавају и исписују по један у реду (а парсирање улаза и форматирање излаза се појављују касније); променљивама се вредност додељује само једанпут у програму (све до итеративних алгоритама), итд. Када су основне технике већ усвојене, у једној групи задатака се приказује употреба библиотечких колекција и функција која значајно поједностављује запис решења, а читаоци се подстичу да користе ове могућности језика C++ (на пример, уместо имплементирања алгоритама сортирања, за сортирање низа се користи библиотечка функција `sort`). Преглед могућности стандардне библиотеке није исцрпан, већ се приказују само изабрани елементи који су довољно једноставни за почетника, а могу значајно да олакшају решавање задатака.

Програмски интерфејс (енгл. API) у свим задацима се своди на читавање података са стандардног улаза и исписивање на стандардни излаз. Претпоставља се да су сви улазни подаци исправно задати у описаном формату и не врши се провера коректности улаза. Тиме је фокус јасно стављен на моделовање проблема и затим разраду и реализацију алгоритама, што сматрамо кључним вештинама при бављењу програмирањем, а што би могло бити запостављено прераним учењем богатијег корисничког интерфејса (нпр. графичког). Програми какви се користе у овој збирци (тзв. конзолни програми) врло лако могу да буду аутоматски покренути, па их је лако аутоматски тестирати (што је омогућено за све задатке на порталу <https://petlja.org>), а имају и значајну примену при аутоматској обради великих количина података.

По завршетку збирке, читалац може наставити да се бави програмирањем на различите начине. Један од могућих наставака је изучавање сложености алгоритама и налажење ефикасних решења проблема. Задаци у овој збирци нису довољно захтевни да би та, иначе важна тема дошла до изражаја при њиховом решавању. Зато смо задацима у којима се траже посебно оптимизовани алгоритми посветили други том збирке.

Препоруке читаоцу који почиње да учи програмирање

Програмирање данас обично подразумева коришћење неког интегрисаног развојног окружења. За језик C++ данас је веома популарно окружење Code::Blocks, које је бесплатно и може се слободно преузети са интернета. Могуће је користити и окружење Microsoft Visual Studio, које је у својој основној варијанти (Community Edition) бесплатно и такође се може слободно преузети са интернета. С обзиром на специфичну природу конзолних програма и задатака у збирци, коришћење тог окружења није неопходно (могуће је, на пример,

користити едитор текста попут нпр. Emacs, Vim, VS Code, Notepad++ и компилатора за C++ попут слободно доступног gcc). Чак је могуће на интернету пронаћи портале који нуде могућност писања једноставних програма на разним језицима, укључујући и C++ (нпр. repl.it).

Збирка представља штампану верзију збирке, доступне на порталу <https://petlja.org>. Саветујемо ти да пре читања решења сваки задатак покушаш самостално да решиш и своја решења аутоматски тестираш у онлајн издању збирке. Знамо да може да буде тешко, поготово на почетку. Немој да одустанеш. Када се осећаш несигурно и не знаш како да напишеш програм, покушај да себи помогнеш постављањем ових питања:

- Шта је дато, а шта се тражи?
- Можеш ли да решиш задатак без рачунара на датом примеру?
- Можеш ли да наведеш кораке којима се долази до резултата?
- Постоји ли пример који се другачије решава?
- Да ли ти је јасан поступак добијања тражених величина у сваком могућем случају?
- Можеш ли да изразиш тај поступак на начин који се лако претвара у наредбе програмског језика?

Дешаваће се да напишеш програм, а да при извршавању не добијаш резултате које очекујеш. Када у таквој ситуацији немаш идеју зашто програм не даје добар резултат, покушај следеће кораке:

- Пронађи што једноставније улазне податке за које твој програм не ради исправно.
- Можеш ли за те улазне податке да решиш задатак без рачунара (ручно)?
- Сада замисли да си рачунар који је добио твој програм. Извршавај наредбу по наредбу. Ако ти је лакше, пиши на папиру вредности променљивих током извршавања програма, тако да не мораш да их памтиш (овај корак можеш да изведеш и лакше ако умеш да на рачунару извршаваш програм корак по корак или део по део, и пратиш вредности променљивих).
- Ако је програм велики, или ти је тешко да пратиш како би рачунар извршавао поједине наредбе, уместо претходног корака додај привремено на нека места у програму наредбе за исписивање вредности променљивих (касније их обриши).
- Упореди оно што добијаш праћењем рада програма (који год начин праћења рада програма да одабереш) са тачним међурезултатима и резултатима добијеним ручним решавањем.
- Да ли сада разумеш зашто програм не решава задатак исправно?
- Можеш ли сада то да поправиш?

Ако после озбиљног покушаја закључиш да ипак не умеш да решиш задатак, погледај решење и потруди се да га разумеш. Врло је вероватно да ће ти то помоћи при решавању следећег задатка.

Било би добро да прочиташ решење и када успешно решиш задатак, јер и тада можеш да научиш нешто ново ако је дато решење суштински другачије од оног до кога ти дођеш.

У Београду, јун 2019. Аутори

Глава 1

Аритметика

У овој глави приказаћемо како се могу решавати разни задаци у којима се врше основна аритметичка израчунавања. Такви задаци се обично срећу у математици, физици и хемији, а коришћење рачунара олакшава њихово решавање, јер је потребно само записати формуле на основу којих се долази до решења, док је рачунар тај који преузима посао извршавања рачунских операција.

1.1 Елементи програмског језика

Пре задатака укратко ћемо описати елементе програмског језика C++ које ћемо користити у решењима у овој глави.

1.1.1 Основна структура програма

Опишимо за почетак једноставан програм који на екран исписује поруку `Zdravo svete!`.

```
#include <iostream>
using namespace std;

int main() {
    // на екран исписујемо поздравну поруку
    cout << "Zdravo svete!" << endl;
    return 0;
}
```

Покушај да овај програм прекуцаш, преведеш и покренеш. Обрати пажњу на то да језик C++ прави разлику између малих и великих слова и врло је важно да ли је нешто написано малим или великим словом.

Централни део програма чини следећи програмски код:

```
// на екран исписујемо поздравну поруку
cout << "Zdravo svete!" << endl;
```

Линија `cout << "Zdravo svete!" << endl;` представља једну **наредбу** којом се на екран исписује поздравна порука `Zdravo svete` (без двоструких наводника), након чега се прелази у нови ред. Објекат `cout` (од енглеског “console output”) представља **стандардни излаз** нашег програма, што је најчешће екран, и у њега се “улива” прво текст `Zdravo svete`, а затим и **прелазак у нови ред** који се означава са `endl` (од енглеског “end line”). Ово “уливање” је представљено симболима `<<` (текст “тече” и улива се на `cout`, па се `cout` назива и **стандардни излазни ток**).

Линија `// на екран исписујемо pozdravnu poruku` је **коментар** и он служи да ономе ко чита овај програм објасни шта се постиже наредном линијом (та линија није битна рачунару за превођење и извршавање овог програма, па је он занемарује). Да би се добио програм који може да се преведе и изврши, овај централни део програма потребно је окружити још неким линијама чије ћемо значење покушати сада да објаснимо.

Сви програми морају да садрже **функцију** `main` тј. у коду је потребно да постоји део облика


```
int main() {
    // ovde se navode naredbe našeg programa
    return 0; // ovim se signalizira da je program uspešno izvršen
}
```

Линијом `int main()` започињемо дефиницију функције `main`. О функцијама ће бити више речи касније, оно што је сада битно да знаш јесте то да је функција `main` главна функција и извршавање сваког С++ програма почиње извршавањем наредби наведених у оквиру ове функције. Део функције између витичастих заграда назива се тело функције. Тело функције садржи наредбе које се извршавају када се позове та функција. Када корисник покрене програм, тада оперативни систем позове функцију `main` тог програма и крене се са извршавањем наредби наведених у њеном телу. Последња наредба у функцији `main` је најчешће `return 0;` којом наш програм оперативном систему враћа вредност 0 и тиме јавља да је његово извршавање успешно завршено. Ако функција врати неку вредност различиту од 0 то може означавати да је дошло до неке грешке приликом извршавања програма, но у овој збирци се нећемо фокусирати на могуће грешке (претпостављаћемо да је увек све у реду), па ће сви наши програми увек само враћати вредност 0.

У првој линији програма директивом `#include <iostream>` у наш програм укључујемо библиотечку за рад са улазно-излазним токовима. То између осталог значи да у нашем програму можемо да неки текст испишемо на екран, да неке вредности учитамо са тастатуре и слично. У овом програму користили смо `cout` и `endl` који користимо када желимо да пређемо у нови ред. Да нисмо навели `#include <iostream>`, добили бисмо поруку о томе да преводилац нашег програма не разуме шта су `cout` и `endl`. Пошто ће сваки програм који будемо писали писати нешто по екрану, сваки ће почињати директивом `#include <iostream>`.

Након директиве `include` наведена је директива `using namespace std;`. Она омогућава да се сви елементи стандардне библиотеке користе без префикса `std::`. На пример, излазни ток се означава са `cout`, те уместо да свуда пишемо да је он део стандардне библиотеке `std`, тј. да пишемо `std::cout` можемо једноставније само писати `cout`. Да нисмо навели `using namespace std;`, тада би централни део нашег програма морао да буде написан у наредном облику.

```
// na ekran ispisujemo pozdravnu poruku
std::cout << "Zdravo svete!" << std::endl;
```

1.1.2 Коментари

Већ смо рекли да у коду можемо писати **коментаре** – текст којим се објашњава шта се у неком делу програма ради и који је намењен ономе ко буде читао програм (или ономе ко је тај програм писао, ако некада касније буде потребе да га доради или преправи). Коментаре рачунар игнорише приликом превођења програма. У језику С++ коментар почиње навођењем ознаке `//` и простира се до краја тог реда (овакве коментаре често називамо линијским коментарима). Можемо писати и коментаре који се протежу кроз неколико суседних редова (тзв. вишелинијске коментаре) и они почињу ознаком `/*`, а завршавају се ознаком `*/`. У овој збирци ћемо се трудити да наводимо коментаре што више (често много више него што је то уобичајена пракса), да бисмо вам помогли у разумевању приложених решења.

1.1.3 Испис, читавање, променљиве, типови

Већ смо видели да се испис на стандардни излаз (то је најчешће екран рачунара тј. такозвана конзола) врши наредбом облика `cout << "...";`, при чему се текст који се исписује наводи између двоструких наводника.

У једном програму је могуће навести и више оваквих наредби (обично наведених један испод другог). На пример,

```
#include <iostream>
using namespace std;

int main() {
    cout << "Ucim da programiram.";
    cout << "Koristim C++.";
    cout << "Zdravo svima!";
    return 0;
}
```

1.1. ЕЛЕМЕНТИ ПРОГРАМСКОГ ЈЕЗИКА

Иако текст програма не мора бити сложен овако уредно (наредбе су увучене, поравнате једна испод друге), то је веома пожељно. Када се програм покрене, иако су наредбе сложене једна испод друге, наведене реченице се исписују једна до друге.

```
Ucim da programiram.Koristim C++.Zdravo svima!
```

Исти ефекат би се постигао навођењем једне наредбе исписа облика:

```
cout << "Ucim da programiram." << "Koristim C++." << "Zdravo svima!";
```

или мало другачије сложено

```
cout << "Ucim da programiram."
      << "Koristim C++."
      << "Zdravo svima!";
```

Ако се жели да се након исписа текста пређе у нови ред, онда је потребно након ниске под двоструким наводницима исписати и знак за прелаз у нови ред `endl`. На пример, функција

```
int main() {
    cout << "Ucim da programiram." << endl;
    cout << "Koristim C++." << endl;
    cout << "Zdravo svima!" << endl;
    return 0;
}
```

исписује реченице једну испод друге:

```
Ucim da programiram.
Koristim programski jezik C++.
Zdravo svima!
```

Текст може да се прочита од корисника. Размотримо наредни програм.

```
#include <iostream>
using namespace std;
```

```
int main() {
    cout << "Kako se zoves?" << endl;
    string ime;
    cin >> ime;
    cout << "Zdravo, ti se zoves " << ime << endl;
    return 0;
}
```

Наредбом `cout << "Kako se zoves?" << endl;` на екран исписујемо текст `Kako se zoves?`, што је веома слично првом програму који смо анализирали.

Након тога желимо да корисник унесе своје име. Текст који корисник унесе морамо негде да упамтимо, да бисмо га касније исписали. Да бисмо упамтили разне вредности (у овом примеру то је текст који је корисник унео, а у наредним примерима ће то бити разни бројеви са којима ћемо вршити различита израчунавања) користимо **променљиве**. У наведеном примеру користимо променљиву под називом `ime` и у њу смештамо текст који је корисник унео. Променљиве можемо замислити као кутијице у којима се чувају различите вредности. У сваком тренутку стара вредност може бити избачена из кутијице и у кутијицу може бити уписана нека нова вредност. C++ спада у групу такозваних **статички типизираних језика** што значи да се за сваку променљиву унапред зна **тип** вредности који се у њој може чувати. У неким променљивама можемо да чувамо само текст, у другима само целе бројеве, у трећим само реалне бројеве и слично. Приликом првог увођења неке променљиве у наш програм, дужни смо да наведемо њен тип. **Декларација** подразумева увођење имена нове променљиве и придруживање типа. У претходном примеру декларација је била линија `string ime;` Њоме смо декларисали променљиву под називом `ime` и рекли да ће она бити типа `string`, тј. да ће се у њој чувати текст.

Наредбом `cin >> ime` учитавамо текст (једну ниску карактера) од корисника. Објекат `cin` (од енглеског "console input") означава **стандардни улаз** који најчешће одговара тастатури. Наравно, очекујемо да корисник унесе своје име (мада може да унесе шта год жели - наш програм то неће приметити). Подаци опет "теку",

једино што овај пут теку са улаза тј. са тастатуре у променљиву `ime` (што је наглашено симболима `>>`). Зато се и `cin` назива **стандардни улазни ток**.

На крају, наредбом `cout << "Zdravo, ti se zoves " << ime << endl`, на стандардни излаз исписујемо прво текст `Zdravo, ti se zoves`, затим садржај променљиве `ime` (то је текст који је корисник унео) и на крају прелазимо у нови ред.

У претходном примеру видели смо како се ради са текстуалним типом података `string`. У првим програмима у овој збирци користићемо следеће основне типове података.

тип	опис	пример
<code>string</code>	текст (ниска карактера)	<code>"Zdravo"</code>
<code>int</code>	цео број	<code>1234</code>
<code>double</code>	реалан број	<code>3.141</code>
<code>bool</code>	логичка вредност	<code>true</code> или <code>false</code>

Иако треба од почетка имати на уму да променљиве не могу да чувају произвољно велике тј. произвољно мале бројеве, у почетку нећемо обрађати превише пажње на то (на пример, у променљивој типа `int` најчешће се могу чувати вредности од око минус две милијарде, па све до око две милијарде, што је сасвим довољно у свим почетним задацима). Слично важи за податке типа `double` (и овај тип има ограничени распон и прецизност тј. број децимала).

Првобитна додела вредности променљивој назива се **иницијализација**. Приликом доделе могуће је променљивама додељивати и неке конкретне вредности (кажемо константе). На пример

```
string ime = "Pera";
```

Пре него што се текстуална променљива иницијализује њена вредност је празан текст (текст `""`).

Рецимо да именовање променљивих треба да тече у складу са њиховим значењем (пожељно је избегавати кратка, неинформативна имена попут `a`, `b`, `x`, `y`, осим ако из контекста програма није потпуно јасно шта би те променљиве могле да означавају). Имена променљивих не смеју да садрже размаке и морају бити састављена само од слова, цифара и доње црте тј. подвлаке (карактера `_`), али не могу почињати цифром.

Прикажимо сада примере декларација и иницијализација вредности других типова. На пример, наредном декларацијом се у програм уводе две променљиве под називом `x` и `y` и каже се да ће оне чувати целобројне вредности.

```
int x, y;
```

Приметимо да смо у претходном примеру једном декларацијом увели две променљиве, што је краће него да смо писали посебно две декларације.

```
int x;
int y;
```

Наравно, и целобројне променљиве могу бити иницијализоване.

```
int x = 1, y = 2;
int a = 3, b, c = 4;
```

У претходном примеру је декларисано пет променљивих, а иницијализоване су четири.

Када су у питању реалне вредности, морамо нагласити да се оне наводе са децималном тачком (у складу са правописом енглеског језика), а не запетом (у складу са правописом српског језика).

```
double pi = 3.14159265;
```

Наредба исписа коју смо раније видели може бити употребљена и за испис бројевних, па и логичких вредности. На пример, наредним наредбама

```
cout << 123 << endl;
int x = 5;
cout << x << endl;
cout << 12.345 << endl;
```

1.1. ЕЛЕМЕНТИ ПРОГРАМСКОГ ЈЕЗИКА

```
double pi = 3.14159265;
cout << pi << endl;
```

исписује се

```
123
5
12.345
3.14159265
```

Приликом исписа реалних бројева могуће је заокружити их на одређени број децимала. То се може постићи коришћењем функције `setprecision(n)`, где је са n означен број децимала који желимо да прикажемо. Ова функција је декларисана у заглављу `iomanip` те је на почетку програма потребно навести директиву `#include <iomanip>`. Квалификатор `fixed` поставља да се сви бројеви у покретном зарезу приказују у фиксном децималном формату. Да би се приказивале и потенцијалне крајње нуле у приказу броја са фиксним бројем децимала користимо квалификатор `showpoint`. Сви ови квалификатори примењују се на ток `cout` у комбинацији са оператором исписа, те бисмо на пример број π могли да прикажемо на 4 децимале на следећи начин:

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    double pi = 3.14159265;
    cout << fixed << showpoint << setprecision(4) << pi << endl;
    return 0;
}
```

Овим добијамо исписан резултат 3.1416.

Напоменимо и да је приликом исписа могуће комбиновати текст и бројевне вредности. На пример,

```
double pi = 3.1415926;
cout << "Vrednost broja pi je " << pi << endl;
```

Поред текста, са стандардног улаза можемо учитавати и бројеве. Размотримо наредни програм.

```
cout << "Koliko imaš godina?" << endl;
int brGodina;
cin >> brGodina;
cout << "Zdravo, ti imaš " << brGodina << " godina." << endl;
```

Након учитавања једног целог броја са тастатуре, кориснику се исписује одговарајућа порука.

Уколико је потребно унети више вредности са тастатуре (на пример, дан и месец рођења), онда се у једној наредби може надовезати учитавање свих потребних вредности. Притом је те две вредности могуће унети у једној линији раздвојене размаком или у две засебне линије.

```
cout << "Kad si rođen?" << endl;
int dan, mesec;
cin >> dan >> mesec;
cout << "Zdravo, ti si rođen " << dan << ". " << mesec << "." << endl;
```

1.1.4 Основне аритметичке операције и изрази

Ниједан од програма које смо до сада средли није могао бити нарочито интересантан. Могли смо да учитамо податке и да их испишемо у неизмењеном облику.

Рачунар је машина која обрађује податке, тј. која примењујући рачунске операције на основу улазних података добија излазне. У рачунару је све записано помоћу бројева и на најнижем нивоу се све операције свде на основне операције над бројевима. Рачунар или компјутер (енгл. *computer*) је справа која рачуна тј. справа која је направљена тако да може веома брзо и ефикасно да изводи рачунске операције над бројевима. Рачунање се назива и **аритметика** (од грчке речи *ἀριθμός* тј. аритмос која значи број, бројање, рачунање), а рачунске операције се називају и **аритметичке операције**.

- Основна аритметичка операција је сабирање. Збир бројева 3 и 5 се у математици представља као $3 + 5$. У програмском језику C++ користи се идентичан запис $3 + 5$. Сабирање је применљиво и на целе и на реалне бројеве. На пример, код који учитава и сабира два цела броја може бити написан на следећи начин.

```
int x, y;
cin >> x >> y;
cout << x + y << endl;
```

- Поред сабирања можемо разматрати одузимање. Разлика бројева 8 и 2 се у математици представља као $8 - 2$. У програмском језику C++ користи се идентичан запис $8 - 2$. Одузимање је применљиво и на целе и на реалне бројеве.
- Још једна од основних операција је и множење. Производ бројева 4 и 6 се у математици представља као $4 \cdot 6$. У програмском језику C++ множење се означава помоћу оператора $*$ и производ бројева 4 и 6 се записује као $4 * 6$.
- У програмском језику C++, наравно, можемо и да делимо, да израчунавамо остатак при дељењу и цео део количника. Дељење реалних бројева се врши помоћу оператора $/$ и количник бројева 7,2 и 6,4 се записује помоћу $7.2 / 6.4$. Дељењем два цела броја добија се њихов целобројни количник, док се остатак при дељењу два цела броја може израчунати оператором $%$. На пример, вредност израза $14 / 4$ једнака је 3, а израза $14 \% 4$ једнака је 2. Ако желимо да одредимо реални количник два цела броја, морамо их представити у реалном облику (на пример, уместо $14/4$ пишемо $14.0/4.0$). У случају да се деле целобројне променљиве онда вршимо *експлицитну конверзију* у реалне вредности навођењем (`double`) испред назива променљиве (нпр. уместо x / y пишемо `(double) x / (double) y`). Нагласимо да је довољно да било дељеник било делилац буду реални да би се применило реално дељење. О дубљим везама између реалног и целобројног типа биће више речи у каснијим поглављима.

Слично као и у математици, од константних вредности и променљивих, применом оператора и заграда граде се **изрази**. **Приоритет оператора** је усклађен са уобичајеним приоритетом у математици, па је приоритет оператора $*$, $/$ и $%$ виши од приоритета оператора $+$ и $-$, док сви наведени оператори имају леву асоцијативност (рачунске операције се изводе с лева на десно). Приоритет и асоцијативност се могу применити навођењем заграда.

У првим програмима ћемо се трудити да приликом извођења операција не мешамо податке различитог типа. Ипак, нагласимо да ако је у изразу један број реалан, а други цео, пре извођења операција се тај цео број претвара у реалан и операција се извршава над реалним бројевима.

1.1.5 Уграђене математичке функције

У многим конкретним применама поред основних аритметичких операција примењују се и неке мало напредније **математичке функције** и неке чувене **математичке константе** (нпр. π). Да бисмо их користили у језику C++, потребно је на почетку програма укључити заглавље `<cmath>` директивом `#include <cmath>`. Функције које ће ти бити потребне у наредним задацима су:

- `pow(x, y)` - израчунава степен x^y , при чему се може применити и за израчунавање корена (не само квадратних) знајући да је $\sqrt[n]{x} = x^{\frac{1}{n}}$;
- `sqrt(x)` - израчунава квадратни корен \sqrt{x} ;
- `abs(x)` - израчунава апсолутну вредност $|x|$;

Напоменимо да је у заглављу `<cmath>` декларисана функција за израчунавање апсолутне вредности реалног броја, док је одговарајућа функција за целе бројеве декларисана у заглављу `<cstdlib>`.

Што се константе π тиче, она у језику C++ није стандардизована. Већина компилатора подржава коришћење константе `M_PI`, међутим, да би се та константа могла употребити, код неких компилатора потребно је на почетку програма (пре директива `include`) навести дефиницију

```
#define _USE_MATH_DEFINES
```

и тек након ње укључити заглавље `<cmath>`. Нагласимо још једном да је наведену дефиницију у програму потребно навести пре свих укључивања заглавља, јер се може десити и да неко друго заглавље индиректно укључи заглавље `<cmath>`.

1.1. ЕЛЕМЕНТИ ПРОГРАМСКОГ ЈЕЗИКА

Поред ових на располагању су нам и тригонометријске функције (синус, косинус, ...), њихове инверзне функције, логаритамска функција, експоненцијална функција и слично, али њих нећемо користити у почетним задацима.

1.1.6 Именоване константе

Вредности које су нам потребне у програму, а које се неће мењати током извршавања програма можемо дефинисати у виду **именованих константи**, које се дефинишу као обичне променљиве, уз навођење кључне речи `const` пре типа податка, уз обавезну иницијализацију вредности на неку константну вредност. На пример:

```
const double PI = 3.14159265;
```

Именованим константама није касније могуће променити вредност у програму.

1.1.7 Дефинисање функција

Поред библиотечких функција које су нам на располагању, програмски језици, па и језик C++ програмеру дају могућност да **дефинише функције**, што доприноси избегавању поновљених делова програма, декомпозицији проблема на мање потпроблеме и бољој организацији кода. Функције можемо замислити слично као у математици. Оне обично за неколико **улазних параметара** израчунавају једну **резултујућу вредност**. На пример, наредна функција израчунава обим правоугоника датих страница.

```
#include <iostream>
using namespace std;

// funkcija izracunava obim pravougaonika na osnovu poznatih
// duzina stranica a i b
double obim(double a, double b) {
    // formula za obim pravougaonika
    return 2*a + 2*b;
}

int main() {
    double a, b;
    // učitavamo duzine stranica pravougaonika
    cin >> a >> b;
    // izracunavamo obim pomoću definisane funkcije
    double O = obim(a,b);
    // ispisujemo izračunati obim
    cout << O << endl;
    return 0;
}
```

Прва линија `double obim(double a, double b)` се назива **декларација функције** и у њој се каже да се функција назива `obim`, да прима две улазне вредности (два параметра) који су реални бројеви и називају се `a` и `b` и да враћа резултат који је такође реалан број. У телу функције (оно је ограничено витичастим заградама) се описује како се резултат израчунава на основу улазних вредности. Када је коначни резултат израчунат, он се враћа кључном речју `return`. Након исправне дефиниције следи позив функције. У претходном примеру позив је извршен у склопу декларације `double O = obim(a, b)`. У позиву се наводе **аргументи** којима се иницијализује вредност параметара. У овом случају то су вредности променљивих које су учитане са тастатуре. Аргументи могу бити и константне вредности (параметри морају бити променљиве). На пример, допуштен је позив `obim(5.0, 7.0)`.

Унутар функције можемо декларисати и користити и променљиве у којима чувамо међурезултате. На пример, можемо дефинисати функцију за израчунавање површине једнакостраничног троугла дате дужине странице.

```
double PovrsinaJednakostranicnogTroughla(double a) {
    // izracunavamo visinu trougla
    double h = a * sqrt(3) / 2.0;
    // izracunavamo površinu na osnovu duzine stranice i visine
    double P = a * h / 2.0;
    // vraćamo konačan rezultat
}
```

```
    return P;
}
```

Овом функцијом је дефинисан начин да се израчуна површина једнакостраничног троугла и када нам год то надале затреба у програму (а у неком математичком задатку то може бити потребно више пута), можемо само позвати функцију и добити жељени резултат.

Постоје и специјалне функције које не враћају вредност. Оне се некада називају процедурама и једини задатак им је да произведу неки пропратни ефекат (на пример, да нешто испишу на екран). Као повратни тип података наводи се `void`. На пример, можемо дефинисати процедуру која око датог текста испишује украсне линије, а онда је позвати неколико пута у програму.

```
#include <iostream>
using namespace std;

void ukrasiTekst(string tekst) {
    cout << "-----" << endl;
    cout << tekst << endl;
    cout << "-----" << endl;
}

int main() {
    ukrasiTekst("Dobar dan!");
    ukrasiTekst("Zdravo, svima!");
    ukrasiTekst("Dovidjenja!");
    return 0;
}
```

1.1.7.1 Више повратних вредности

Параметри функције се по правилу сматрају њеним улазним вредностима (исто као и у математици), док се вредност враћена помоћу `return` сматра њеном излазном вредношћу. Функција, дакле, може имати више улазних и само једну излазну вредност (и ово одговара појму функције у математици). Међутим, у програмирању некада имамо потребу да дефинишемо функције које враћају више од једне вредности. То је могуће урадити у језику C++ тако што параметре **преносимо по референци** (непосредно након назива типа наведемо карактер `&`). Након тога могуће је вредности тих параметара променити унутар функције и та измена ће важити и након завршетка рада функције. На пример, наредна функција конвертује дужину задату само у центиметрима у дужину задату у метрима и центиметрима.

```
void uMetreICentimetre(int duzina, int& metri, int& centimetri) {
    metri = duzina / 100;
    centimetri = duzina % 100;
}
```

Позив ове функције се врши на следећи начин.

```
int metri, centimetri;
uMetreICentimetre(273, metri, centimetri);
```

Након овог позива, вредност променљиве `metri` биће 2, а центиметри биће 73. Параметри који се преносе по референци су **улазно-излазни параметри** функције.

1.2 Елементарна целобројна и реална аритметика

У наставку ћемо размотрити широк дијапазон задатака из домена математике, физике и хемије који се могу решити и само применом операција које смо до сада научили.

1.2.1 Геометријске формуле

У овом поглављу приказаћемо неколико задатака из области геометрије и примене геометријских формула.

1.2. ЕЛЕМЕНТАРНА ЦЕЛОБРОЈНА И РЕАЛНА АРИТМЕТИКА

Подсетимо се неких основних формула које ће нам бити потребне у наредним задацима. Обим правоугаоника чија је дужина a , а ширина b једнака је $2 \cdot a + 2 \cdot b$, док је његова површина једнака $a \cdot b$. Обим круга чији је полупречник једнак r износи $2 \cdot r \cdot \pi$, док му је површина једнака $r^2 \cdot \pi$.

Обим троугла чије су дужине страница a , b и c једнак је $O = a + b + c$, док се површина може израчунати Хероновим обрасцем $P = \sqrt{s(s-a)(s-b)(s-c)}$, где је $s = (a + b + c)/2$ полуобим троугла.

На правоугли троугао чије су дужине катета једнаке a и b , а хипотенуза је дужине c може се применити Питагорина теорема која гласи: квадрат над хипотенузом једнак је збиру квадрата над катетама, односно: $a^2 + b^2 = c^2$.

У задацима који следе претпоставићемо да су ти познати основни појмови у вези са Декартовим координатним системом. Декартов систем у равни задат је двема осама: x -осом или апсцисом и y -осом или ординатом. Тачка у равни је одређена вредностима своје две координате, које најчешће означавамо са x и y . Координатни почетак је тачка са координатама $(0, 0)$.

Задатак: Тренинг

Спортиста се на почетку тренинга загрева тако што трчи по ивицама правоугаоног терена дужине d и ширине s . Написати програм којим се одређује колико метара претрчи спортиста док једном обиђе терен.

Улаз: У првој линији стандардног улаза се налази целобројна вредност d ($0 < d \leq 100$), а у следећој линији целобројна вредност s ($0 < s \leq 100$) које редом представљају дужину и ширину терена изражену у метрима.

Изназ: Цео број метара које претрчи спортиста док једном обиђе терен.

Пример

Улаз	Изназ
50	150
25	

Решење

Дужина у метрима коју претрчи спортиста док једном обиђе терен једнака је обиму тог терена тј. збиру двоструких вредности дужине и ширине терена изражене у метрима. Према томе, потребно је исписати збир двоструких вредности два учитана броја.

```
#include <iostream>
using namespace std;

int main() {
    int duzina, sirina;
    cin >> duzina >> sirina;
    int obim = 2 * duzina + 2 * sirina;
    cout << obim << endl;
    return 0;
}
```

Задатак: Лист папира

Написати програм који на основу задате ширине и висине листа папира (правоугаоног облика) у милиметрима одређује његову површину у квадратним милиметрима.

Улаз: У једној линији стандардног улаза налазе се две целобројне вредности V и S ($0 < V \leq 300$, $0 < S \leq 300$) које представљају ширину и висину листа папира изражену у милиметрима.

Изназ: Један цео број који представља површину листа у квадратним милиметрима

Пример

Улаз	Изназ
279 216	60264

Решење

Површина листа папира изражена у квадратним милиметрима једнака је производу висине и ширине тог правоугаоника изражених у милиметрима. Према томе, потребно је исписати производ два учитана броја.


```
#include <iostream>

using namespace std;

int main() {
    int visina, sirina;
    cin >> visina >> sirina;
    int povrsina = visina * sirina;
    cout << povrsina << endl;
    return 0;
}
```

Задатак: Столњак

Написати програм којим се израчунава потребна дужина траке за поруб столњака кружног облика чија је површина P .

Улаз: У линији стандардног улаза се налази реална вредност P ($1 \leq P \leq 1000$).

Излаз: Потребна дужина траке (реалан број заокружен на две децимале).

Пример

Улаз	Излаз
7853.982	314.16

Решење

Пошто је $P = r^2\pi$, из познате површине круга P може да се одреди полупречник r (на основу везе $r = \sqrt{P/\pi}$). Када је израчунат полупречник, потребно је да се примени формула за обим круга $O = 2r\pi$.

У језику C++ константа π није стандардизована, већ да би се та константа могла употребити потребно је на почетку програма (пре `include` директива) навести дефиницију

```
#define _USE_MATH_DEFINES
```

и тек након ње укључити заглавље `cmath`. Други начин је, наравно, дефинисање константе у програму

```
const double PI = 3.14159265359;
```

међутим, поставља се питање броја децимала које је потребно навести (две, три децимале које ученици знају напамет нису увек довољне).

Алтернатива је да се π израчуна коришћењем неких тригонометријских функција (нпр. као `acos(-1)` или `4 * atan(1)`).

```
double PI = acos(-1);
```

или

```
double PI = 4*atan(1);
```

```
#define _USE_MATH_DEFINES
```

```
#include <iostream>
```

```
#include <iomanip>
```

```
#include <cmath>
```

```
using namespace std;
```

```
int main() {
    double P, r, O;
    cin >> P; // ucitavamo povrsinu
    r = sqrt(P / M_PI); // izracunavamo poluprecnik
    O = 2 * r * M_PI; // izracunavamo obim
    cout << fixed << showpoint << setprecision(2) << O << endl;
}
```

1.2. ЕЛЕМЕНТАРНА ЦЕЛОБРОЈНА И РЕАЛНА АРИТМЕТИКА

Задатак: Фудбалски терен

Фудбалски терен димензија $d \times s$ треба оградити правоугаоном оградом тако да је растојање страница ограде од линије терена r . Напиши програм који одређује дужину ограде.

Улаз: У три реда стандардног улаза налазе се три цела броја:

- d - дужина терена у метрима ($90 \leq d \leq 120$)
- s - ширина терена у метрима ($45 \leq s \leq 90$)
- r - растојање ограде од терена у метрима ($2 \leq r \leq 10$)

Излаз: Дужина ограде у метрима.

Пример

Улаз	Излаз
100	344
60	
3	

Решење

Дужине страница правоугаоне ограде су $a = d + 2 \cdot r$ и $b = s + 2 \cdot r$, па је дужина ограде обим правоугаоника са наведеним страницама, који је једнак: $2 \cdot (a + b) = 2 \cdot ((d + 2 \cdot r) + (s + 2 \cdot r)) = 2 \cdot (d + s + 4 \cdot r)$.

У програму је могуће употребити помоћне променљиве у којима се чува дужина страница ограде, а затим њиховим коришћењем израчунати обим. Тиме се добија мало дужи, али разумљивији програм.

```
int duzinaTerena, sirinaTerena, растојanje;
cin >> duzinaTerena >> sirinaTerena >> растојanje;
int duzinaOgrade = duzinaTerena + 2 * растојanje;
int sirinaOgrade = sirinaTerena + 2 * растојanje;
int обимOgrade = 2 * (duzinaOgrade + sirinaOgrade);
cout << обимOgrade << endl;
```

Друга могућност је директно применити крајњу формулу коју смо извели. Тиме се добија краћи, али мало мање разумљив програм.

```
int d, s; // duzina i sirina terena
int r;    // растојanje izmedju terena i ograde
cin >> d >> s >> r;
int обим = 2 * (d + s + 4 * r); // обим ograde
cout << обим << endl;
```

Задатак: Растојање тачака

Напиши програм који израчунава и исписује растојање између тачака задатих својим координатама.

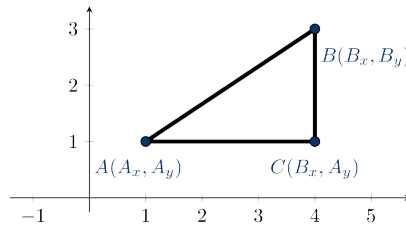
Улаз: Са стандардног улаза уносе се четири реална броја, сваки у посебном реду. Прва два броја A_x и A_y представљају координате тачке $A = (A_x, A_y)$, док друга два броја B_x и B_y представљају координате тачке $B = (B_x, B_y)$.

Излаз: На стандардни излаз исписати један реалан број који представља растојање између тачака A и B , заокружен на пет децимала.

Пример

Улаз	Излаз
0	1.41421
0	
1	
1	

Решење



Слика 1.1: Растојање тачака

Посматрајмо тачку C која има координате (B_x, A_y) . Троугао ABC је правоугли троугао са правим углом код темена C . Тражено растојање између тачака A и B једнако је дужини хипотенузе тог троугла и може се израчунати применом Питагорине теореме која тврди да је квадрат над хипотенузом једнак збиру квадрата над обе катете. Пошто су катете нашег троугла дужи AC и BC , важи да је $|AB|^2 = |AC|^2 + |BC|^2$, па је $|AB| = \sqrt{|AC|^2 + |BC|^2}$. Пошто тачке A и C имају исту y -координату, дужина дужи AC једнака је $|B_x - A_x|$. Заиста, пошто је дуж AC паралелна оси x , њена дужина једнака је дужини интервала који представља њену пројекцију на ту осу. То је интервал $[A_x, B_x]$ ако је $A_x \leq B_x$ и његова дужина је $B_x - A_x$, тј. интервал $[B_x, A_x]$ ако је $B_x \leq A_x$ и његова дужина је $A_x - B_x$. У оба случаја, дужина је једнака $|B_x - A_x|$. Слично, дужина дужи BC једнака је $|B_y - A_y|$. Зато је $|AB| = \sqrt{|B_x - A_x|^2 + |B_y - A_y|^2}$. Пошто је квадрат броја увек позитиван, није неопходно користити апсолутну вредност и важи да је $|AB| = \sqrt{(B_x - A_x)^2 + (B_y - A_y)^2}$.

Подсетимо се, у језику C++ се квадратни корен може израчунати библиотечком функцијом `sqrt` декларисаном у заглављу `<math>`.

```
double ax, ay, bx, by;
cin >> ax >> ay >> bx >> by;
double dx = bx - ax, dy = by - ay;
double d = sqrt(dx*dx + dy*dy);
cout << fixed << showpoint << setprecision(5)
      << d << endl;
```

Задатак: Површина троугла датих темена

Напиши програм који израчунава површину троугла ако су познате координате његових темена.

Улаз: Са стандардног улаза уноси се 6 реалних бројева (сваки у засебном реду). Прва два представљају x и y координату темена A , друга два x и y координату темена B и последња два представљају x и y координату темена C .

Издаз: На стандардни издаз исписати један реалан број који представља површину троугла (допуштена грешка израчунавања је 10^{-5}).

Пример

Улаз	Издаз
0	0.5
0	
0	
1	
1	
0	

Решење

Задатак је математички могуће решити на неколико начина.

Херонов образац

Један начин је да применимо Херонов образац

$$P = \sqrt{s \cdot (s - a) \cdot (s - b) \cdot (s - c)},$$

1.2. ЕЛЕМЕНТАРНА ЦЕЛОБРОЈНА И РЕАЛНА АРИТМЕТИКА

где је су a , b и c дужине страница троугла, а $s = (a + b + c)/2$ његов полуобим. Дужине страница можемо израчунати као растојање између темена троугла, применом Питагорине теореме, како је то показано у задатку **Растојање тачака**.

Мана овог решења је то што се у њему користи квадратни корен, иако се до решења може доћи и без кореновања.

```
// растојање између тачака (x1, y1) и (x2, y2)
double растојање(double x1, double y1, double x2, double y2) {
    double dx = x2 - x1;
    double dy = y2 - y1;
    return sqrt(dx*dx + dy*dy);
}

// површина троугла за задате дужине страница
double површинаТроугла(double a, double b, double c) {
    // Heronov obrazac
    double s = (a + b + c) / 2;
    return sqrt(s*(s-a)*(s-b)*(s-c));
}

int main() {
    // координате тачака троугла
    double Ax, Ay, Bx, By, Cx, Cy;
    cin >> Ax >> Ay >> Bx >> By >> Cx >> Cy;

    // дужине страница троугла
    double a = растојање(Bx, By, Cx, Cy);
    double b = растојање(Ax, Ay, Cx, Cy);
    double c = растојање(Ax, Ay, Bx, By);

    // површина троугла
    double P = површинаТроугла(a, b, c);

    cout << setprecision(5) << showpoint << fixed << P << endl;

    return 0;
}
```

Формула пертле

Задатак је могуће решити и општијом **формулом пертле** која омогућава израчунавање површине произвољних полигона којима се странице не пресецају међусобно. Заиста, ако напишемо координате тачака у следећем облику

$$\begin{array}{cc} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ a_x & a_y \end{array}$$

Формула се гради тако што се са једним знаком узимају производи одозго-наниже, слева-удесно (то су $a_x b_y$, $b_x c_y$ и $c_x a_y$), док се са супротним знаком узимају производи одоздо-навише, слева удесно (то су $a_x c_y$, $c_x b_y$, и $b_x a_y$), што, када се нацрта, заиста подсећа на цик-цак везивање пертли.

$$\frac{|b_x c_y - a_x c_y - b_x a_y - b_y c_x + a_y c_x + a_x b_y|}{2}$$

Ова се формула може извести помоћу векторских производа и детерминанти, али и елементарнијим техникама (сабирањем означених производа троуглова или трапеза).

```
double površinaTougla(double Ax, double Ay,
                    double Bx, double By,
                    double Cx, double Cy) {
    // pertle (cik-cak)
    return abs(Ax*By + Bx*Cy + Cx*Ay - Ax*Cy - Cx*By - Bx*Ay) / 2.0;
}
```

Задатак: Ниво базена

Током олује пало је n литара воде по квадратном метру. Написати програм којим се одређује за колико центиметара се подигао ниво воде у отвореном базену датих димензија, ако знамо да се вода није прелила преко базена.

Улаз: У првој линији стандардног улаза налази се реалан број n који представља количину кише која је пала по квадратном метру изражену у литрима. У следеће три линије налазе се реални бројеви a, b, c који представљају димензије базена изражене у метрима и то редом дужину, ширину и дубину.

Израз: На стандардном излазу приказати један реалан број на две децимале који представља за колико центиметара се подигао ниво воде у базену.

Пример

Улаз	Израз
75.5	7.55
4	
7	
1.7	

Задатак: Правоугаоник дат наспрамним теменима

Дате су координате два наспрамна темена правоугаоника чије су странице паралелне координатним осама. Написати програм којим се приказују дужина дијагонале одређене тим теменима, обим и површина тог правоугаоника.

Улаз: Са стандардног улаза учитавају се 4 цела броја из интервала $[-1000, 1000]$, сваки у посебној линији. Учитани бројеви представљају редом x и y координату два наспрамна темена правоугаоника.

Израз: На стандардном излазу приказати дужину дијагонале правоугаоника, заокружену на две децимале, обим и површину датог правоугаоника. Сваки податак приказати у посебној линији.

Пример

Улаз	Израз
5	10.63
-1	30
-3	56
6	

Задатак: Троугао одређен правом и координатним почетком

Одреди површину и обим троугла којег права $y = a \cdot x + b$ ($a, b \neq 0$), задата коефицијентима a и b , образује са координатним осама.

Улаз: Два реална броја различита од 0. У првој линији a , а у другој b .

Израз: Приказати у првој линији површину троугла, а у другој обим, заокружене на две децимале.

Пример

Улаз	Израз
2.5	1.80
3.0	7.43

1.2.2 Линеарне једначине и системи линеарних једначина

Линеарна једначина са једном непознатом је једначина облика: $a \cdot x = b$ (или њена еквивалентна форма $ax + b = c$), где је са x означена променљива, а са a, b и c коефицијенти који су најчешће реални бројеви. За решење ове линеарне једначине важи следеће:

1.2. ЕЛЕМЕНТАРНА ЦЕЛОБРОЈНА И РЕАЛНА АРИТМЕТИКА

- ако је $a \neq 0$, онда је решење облика $x = \frac{b}{a}$
- ако је $a = 0$ и $b = 0$, онда једначина има бесконачно много решења
- ако је $a = 0$ и $b \neq 0$, онда једначина нема решења

Најједноставнији систем линеарних једначина је са две једначине и две непознате:

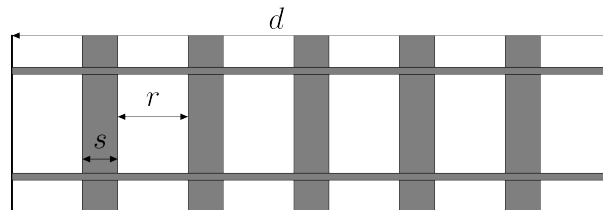
$$\begin{aligned}a_1x + b_1y &= c_1 \\ a_2x + b_2y &= c_2\end{aligned}$$

где су са x, y означене непознате, а са $a_1, a_2, b_1, b_2, c_1, c_2$ коефицијенти.

Систем има јединствено решење ако је $a_1b_2 - a_2b_1 \neq 0$ и можемо га решити тако што ћемо изразити једну непознату у једној од једначина, (на пример, ако је $a_1 \neq 0$, променљиву x из прве једначине као $x = (c_1 - b_1y)/a_1$) и уврстимо добијени израз у другу једначину. На тај начин добијамо једначину са једном непознатом y коју уметом да решимо, а на основу њене вредности знамо да израчунамо и вредност друге непознате x . Други начин за решавање оваквог система јесте да множењем ових једначина погодном одабраним коефицијентима ове једначине доведемо у стање да коефицијенти уз једну променљиву у овим једначинама буду супротни. Сабирањем овако добијених једначина добијамо линеарну једначину са једном непознатом коју знамо да решимо, а када имамо вредност једне непознате, једноставно добијамо и вредност друге непознате у систему.

Задатак: Ограда терасе

На тераси дужине d метара треба распоредити n стубића ширине s центиметара тако да растојање између стубића, као и између стубића и зида буде исто.



Слика 1.2: Ограда терасе

Улаз: Три реда стандардног улаза садрже три броја:

- d - реалан број који представља дужину терасе у м ($1 \leq d \leq 20$)
- n - цео број стубића ($1 \leq n \leq 100$)
- s - реалан број који представља ширину стубића у см ($5 \leq s \leq 15$)

Изназ: Растојање између стубића у центиметрима, заокружено на две децимале.

Пример

Улаз	Изназ
10	22.58
30	
10	

Решење

Нека је r непознато растојање између стубића. Тада n стубића покрива $n \cdot s$ см. Између n стубића постоји $n - 1$ размак, један размак је између левог зида и првог стубића и један размак је између последњег стубића и десног зида. Дужина терасе у центиметрима је $d \cdot 100$ см. Дакле, важи услов

$$n \cdot s \text{ см} + (n + 1) \cdot r \text{ см} = d \cdot 100 \text{ см}$$

Решавањем ове једначине добија се да је растојање између стубића

$$r = \frac{d \cdot 100 - n \cdot s}{n + 1}.$$

До истог закључка смо могли доћи и мало неформалније. Укупно постоји $n + 1$ размак чију дужину треба израчунати. Зато за поделу на $n + 1$ део остаје $(d \cdot 100 - n \cdot s)$ cm, чиме се добија исто решење као претходно.

```
int n;          // broj stubica
double d;      // duzina terase (u m)
double s, r;   // sirina stubica i rastojanje izmedju stubica (u cm)
cin >> d >> n >> s;
r = (d * 100 - n * s) / (n + 1);
cout << fixed << showpoint << setprecision(2) << r << endl;
```

Задатак: Збирови по три странице правоугаоника

Ако је збир неке три странице правоугаоника m , а збир неке друге три странице истог правоугаоника n , написати програм којим се одређује обим и површина тог правоугаоника.

Улаз: Прва линија стандарног улаза садржи реалан број m који представља збир неке три странице правоугаоника, друга линија садржи реалан број n који представља збир неке друге три странице правоугаоника.

Излаз: У првој линији стандарног излаза приказати на две децимале обим правоугаоника, а у другој линији површину правоугаоника такође на две децимале.

Пример

Улаз	Излаз
8	12.00
10	8.00

Решење

Обележимо странице правоугаоника са a и b . Тада је обим правоугаоника $2 \cdot a + 2 \cdot b$, а површина је $a \cdot b$. Ако је први збир три странице правоугаоника $2 \cdot a + b$ онда је збир друге три странице $a + 2 \cdot b$. Тако добијамо две једначине $2 \cdot a + b = n$ и $a + 2 \cdot b = m$ из којих можемо одредити a и b . Један начин да се дође до решења је да се реши систем једначина неком од класичних метода решавања система једначина, чиме се добија да је $a = \frac{2n-m}{3}$ и $b = \frac{2m-n}{3}$, на основу чега се веома једноставно одређују обим и површина као $O = 2(a + b)$ и $P = a \cdot b$.

```
double n, m;          // zbrovi a+2b i b+2a
cin >> n >> m;
double a = (2 * n - m) / 3.0; // duzina stranice a
double b = (2 * m - n) / 3.0; // duzina stranice b
double O = 2 * (a + b);      // obim
double P = a * b;           // površina
cout << fixed << showpoint << setprecision(2) << O << endl << P << endl;
```

Задатак: Просек на такмичењу

На такмичењу из математике учествовало је N ученика. Извештај о броју поена одштампан је на две стране. Наставник грешком није понео прву страну извештаја, али се сећа да је на дну стране писало да је просечан број поена за ту страну био p_1 . На другој страни (коју има код себе) су подаци о K ученика и просечан број поена за ту страну је p_2 . Написати програм којим се одређује колики је просечан број поена свих ученика.

Улаз: На стандардном улазу налазе се

- у првој линији природан број N укупна број ученика ($N \leq 200$)
- у другој линији природан број K број ученика на другој страни ($K \leq 100$)
- у трећој линији реалан број p_1 просечан број поена ученика на првој страни ($0 \leq p_1 \leq 100$)
- у четвртој линији реалан број p_2 просечан број поена ученика на другој страни ($0 \leq p_2 \leq 100$)

Излаз: На стандардном излазу приказати, на две децимале, просечан број свих ученика.

Пример

Улаз	Излаз
80	82.36
30	
78.20	
89.30	

1.2.3 Линеарна зависност и пропорција

Ако је зависност променљивих x и y изражена формулом облика $y = ax + b$, онда за променљиве x и y кажемо да су линеарно зависне. Везу $y = ax + b$ можемо разматрати и као функцију која слика x у y и ову функцију онда називамо линеарном функцијом, при чему је онда број a коефицијент линеарне функције, а b слободни члан.

Количник два броја називамо и њиховом размером. Ако две размере $a : b$ и $c : d$ имају исту вредност, онда кажемо да оне чине пропорцију и то записујемо у облику $a : b = c : d$. Када је један од елемената пропорције непознат, пропорцију можемо да решимо тако што је напишемо у облику једнакости производа њених унутрашњих чланова са производом њених спољашњих чланова: $b \cdot c = a \cdot d$.

Задатак: Подела интервала

Дат је интервал $[a, b]$ реалне праве. Написати програм којим се одређује тачка реалне праве која интервал дели у размери $p : q$.

Улаз: У првој линији стандарног улаза налази се реалан број a , у другој линији реалан број b ($a < b$). Трећа и четврта линија садрже природне бројеве редом p и q .

Издаз: На стандардни издаз исписати један реалан број заокружен на две децимале који представља тражену тачку.

Пример

Улаз	Издаз
0	2.00
10	
1	
4	

Решење

Интервал $[a, b]$ је дужине $b - a$ и треба га поделити у размери $p : q$. То значи да тражимо тачку x тог интервала тако да је дужина интервала $[a, x]$ једнака $p \cdot k$ а дужина интервала $[x, b]$ једнака $q \cdot k$, за неки реалан број k . Како је дужина интервала $[a, b]$ једнака збиру дужина интервала $[a, x]$ и $[x, b]$, добијамо једначину $b - a = p \cdot k + q \cdot k$ тј. $b - a = (p + q) \cdot k$. Решавањем једначине добијамо да је $k = \frac{b-a}{p+q}$, па је тражена тачка $x = a + p \cdot k$.

```
double a, b;
int p, q;
cin >> a >> b >> p >> q;
double k = (b - a) / (p + q);
double x = a + p * k;
cout << fixed << showpoint << setprecision(2) << x << endl;
```

До решења можемо доћи коришћењем основног својства пропорције да је $a : b = c : d$ еквивалентно са $a \cdot d = b \cdot c$. Тачка x дели интервал $[a, b]$ у размери $p : q$ ако су дужине интервала $[a, x]$ и $[x, b]$ у размери $p : q$, тј. ако је $(x - a) : (b - x) = p : q$. Из последње пропорције добијамо да је

$$x = \frac{q \cdot a + p \cdot b}{p + q}.$$

```
double a, b;
int p, q;
cin >> a >> b >> p >> q;
double x = (b * p + a * q) / (p + q);
cout << fixed << showpoint << setprecision(2) << x << endl;
```

Задатак: Генератор случајних бројева

У рачунарству се често јавља потреба за генерисањем случајних (насумичних) бројева. Већина програмских језика обезбеђује функцију која враћа насумично одабран реалан број из интервала $[0, 1)$ (тај број већи или једнак од 0, а строго мањи од 1), подразумевајући да програмер може накнадним аритметичким трансформацијама добити број из било ког интервала $[a, b)$ (за који важи да је $a < b$). Напиши програм који за унети

број x из интервала $[0, 1)$ одређује њему одговарајући број из интервала $[a, b)$, подразумевајући да се преликавање врши линеарном функцијом (број који се тражи дели интервал $[a, b)$ у истом односу у ком тачка x дели интервал $[0, 1)$).

Улаз: Са стандардног улаза учитава се један реалан број x из интервала $[0, 1)$ (број који је вратио генератор случајних бројева), затим, у наредном реду један реалан број a и онда у наредном реду број b (при том важи $a < b$).

Излаз: На стандардни излаз исписати један реалан број из интервала $[a, b)$ који се добија када се унети број x прелика линеарном функцијом која преликава интервал $[0, 1)$ на интервал $[a, b)$. Резултат приказати заокружен на 5 децимала.

Пример 1

<i>Улаз</i>	<i>Излаз</i>
0.5	0.50000
-2	
3	

Пример 2

<i>Улаз</i>	<i>Излаз</i>
0	-2.00000
-2	
3	

Пример 3

<i>Улаз</i>	<i>Излаз</i>
0.8	2.00000
-2	
3	

Решење

Потребно је дефинисати линеарну функцију која вредност 0 преликава у вредност a док вредност 1 преликава у вредност b . Ако је та функција облика $y = k \cdot x + n$ тада важи да је $a = k \cdot 0 + n$ и $b = k \cdot 1 + n$. Зато је $n = a$ и $k = b - a$ и тражено преликавање је $y = (b - a) \cdot x + a$. Заиста, када вредност из интервала $[0, 1)$ помножимо бројем $b - a$ добијамо број из интервала $[0, b - a)$. Додавањем броја a добијамо број из интервала $[a, b)$.

```
double x, a, b;
cin >> x >> a >> b;
double y = a + x * (b - a);
cout << fixed << showpoint << setprecision(5) << y << endl;
```

Задатак: Група радника

Група коју чини n радника уради посао за s сати. Написати програм којим се одређује за колико сати ће посао бити завршен ако се прикључи још m радника, а сваки радник ради посао подједнако ефикасно.

Улаз: У три линије стандардног улаза се редом налазе природан број n , реалан број s и природан број m .

Излаз: Један реалан број, заокружен на две децимале, који представља број сати потребних да се посао заврши када посао ради $n + m$ радника.

Пример

<i>Улаз</i>	<i>Излаз</i>
1	1.5
3	
1	

Решење

Један начин да се задатак реши је да се прво израчуна колико је радник-сати потребно да се заврши цео посао. Радник-сат посматрамо као јединицу за количину посла, тј. ону количину посла коју један радник може да обави за 1 сат. Пошто сваки од n радника ради s_n сати, укупна количина посла је $n \cdot s_n$ радник-сати. Када тај посао обавља $n + m$ радника, време потребно за обављање посла добијамо дељењем количине посла бројем радника, па је број сати за које ће посао бити завршен једнак

$$\frac{n \cdot s_n}{n + m}$$

До истог израза се може доћи и применом пропорције. Више радника брже заврши посао, па је потребно применити обрнуту пропорцију. Ако са x обележимо број сати за које посао уради већа група радника, до решења се долази решавањем пропорције $n : (n + m) = x : s_n$ (са обе стране једнакости вредности су поређане од мање ка већој).

```
int n; cin >> n; // pocetni broj radnika
double sn; cin >> sn; // vreme za koje zavrse posao
int m; cin >> m; // broj radnika koji se pridruze
```

1.2. ЕЛЕМЕНТАРНА ЦЕЛОБРОЈНА И РЕАЛНА АРИТМЕТИКА

```
double snm = (n * sn) / (n + m); // време za koje zajedno zavrse
cout << fixed << showpoint << setprecision(2) << snm << endl;
```

Задатак: Такси

У такси су истовремено ушла три путника, али су излазили један по један. Договорили су се да у плаћању сваког дела вожње подједнако учествују путници који су у том делу вожње били у таксију. Ако се зна стање на таксиметру у динарима када је изашао свако од њих, колико треба да плати путник који је изашао први, колико други, а колико трећи?

Улаз: Са стандардног улаза се читавају три реална броја, у сваком реду по један, у следећем редоследу:

- c_1 - стање на таксиметру у моменту изласка првог путника,
- c_2 - стање на таксиметру у моменту изласка другог путника,
- c_3 - стање на таксиметру у моменту изласка трећег путника.

При томе важи $c_1 < c_2 < c_3$.

Изназ: На стандардни излаз исписати три реална броја заокружена на две децимале (у сваком реду по један), који редом представљају трошкове путовања првог, другог и трећег путника.

Пример

Улаз	Изназ
1200.0	400.00
1500.0	550.00
2000.0	1050.00

Задатак: Курс

Написати програм којим се на основу датог куповног и продајног курса евра према динару и куповног и продајног курса долара према динару, израчунава куповни и продајни курс долара према еврџ и евра према долару. Куповни курс валуте 1 према валути 2 представља износ валуте 1 који се може купити јединицом валуте 2 (валута 1 се третира као роба која се купује), а продајни курс валуте 1 према валути 2 представља износ валуте 2 који се може добити продајом јединице валуте 1 (валута 1 се третира као роба која се продаје). На пример, куповни курс евра према динару је број динара потребних да би се купио један евро, док је продајни курс евра према динару број динара који се могу добити продајом једног евра. Све трансакције евра према долару и обрнуто се обављају, по прописима НБС, преко конверзије у динаре.

Улаз: У свакој од четири линије стандардног улаза налазе се редом куповни курс евра према динару, продајни курс евра према динару, куповни курс долара према динару, продајни курс долара према динару (реални бројеви).

Изназ: У свакој од четири линије стандардног излаза исписују се редом реални бројеви, куповни курс долара према еврџ, продајни курс долара према еврџ, куповни курс евра према долару и продајни курс евра према долару, заокружени на 4 децимале.

Пример

Улаз	Изназ
123.59	0.8931
122.85	0.8824
109.72	1.1332
109.06	1.1197

1.2.4 Степеновање и кореновање

Задатак: Степен и корен

Ђока је вежбао множење децималних бројева тако што је за неки број a рачунао производ $\underbrace{a \cdot a \cdot \dots \cdot a}_n$, тј. производ у којем се број a јавља n пута као чинилац. Након тога се са својом другарицом Даницом играо тако што јој је задавао да она погоди број a када јој он каже број n и резултат који је добио. Напиши програм који Ђоки помаже да провери да ли је добро рачунао, а Даница помаже да провери да ли је добро погодила непознат број.

Улаз: Са стандардног улаза уноси се реалан број x (са највише две децимале) и цео број n .

Излаз: На стандардни излаз се исписују два реална броја (заокружена на пет децимала) и то сваки у посебном реду:

- број s такав да је $s = \underbrace{x \cdot x \cdot \dots \cdot x}_n$,
- број k такав да је $x = \underbrace{k \cdot k \cdot \dots \cdot k}_n$.

Пример

Улаз	Излаз
2.5	97.65625
5	1.20112

Решење

Важи да је $\underbrace{a \cdot a \cdot \dots \cdot a}_n = a^n$. Дакле, за дато x потребно је пронаћи s тако да је $s = x^n$, што се постиже степеновањем и пронаћи k тако да је $x = k^n$, што се постиже кореновањем тј. израчунавањем $k = \sqrt[n]{x}$. У језику C++ степен је могуће израчунати библиотечком функцијом `pow` декларисаном у заглављу `<cmath>`. Иста функција се може употребити и за израчунавање корена, коришћењем формуле $\sqrt[n]{x} = x^{1/n}$. Приликом израчунавања количника $\frac{1}{n}$ бар један од бројева 1 и n мора бити задат као реалан број.

```
double x, n;
cin >> x >> n;
cout << fixed << showpoint << setprecision(5)
    << pow(x, n) << endl << pow(x, 1.0/n) << endl;
```

1.2.5 Кретање

У овом поглављу приказаћемо неколико задатака из области физике тј. кретања. Највећи број задатака биће у вези са за равномерним кретањем (што значи да се тело све време креће истом брзином). Брзина v при равномерном кретању једнака је количнику пређеног пута s и времена t , тј. при равномерном кретању важи веза $v = \frac{s}{t}$, под претпоставком да се користе одговарајуће јединице мере.

Још један правилан облик кретања је равномерно убрзано кретање, што значи да се брзина мења на основу константног убрзања. Код равномерно убрзаног кретања брзина након времена t се може израчунати формулом $v = v_0 + a \cdot t$, где је v_0 почетна брзина а a је убрзање. Пређени пут се може израчунати формулом $s = v_0 \cdot t + \frac{a \cdot t^2}{2}$.

Задатак: Путовање

Породица је кренула аутом на летовање. Аутомобилом треба да пређу s километара крећући се равномерном брзином од $v \frac{\text{km}}{\text{h}}$. Написати програм којим се одређује за колико ће сати прећи пут.

Улаз: У првој линији стандардног улаза се налази реална вредност v , а у следећој линији реална вредност s које редом представљају брзину изражену у $\frac{\text{km}}{\text{h}}$ и планирани пређени пут у километрима.

Излаз: Један реалан број заокружен на две децимале који представља потребно време у сатима.

Пример

Улаз	Излаз
60	17.50
1050	

Решење

Брзина v при равномерном кретању једнака је количнику пређеног пута s и времена t , тј. важи веза $v = \frac{s}{t}$. Зато се време израчунава као количник пређеног пута и брзине кретања (формула за време приликом равномерном кретања $t = \frac{s}{v}$). Према томе, потребно је исписати количник два учитана броја.

```
double v, s;
cin >> v >> s;
double t = s / v;
cout << fixed << showpoint << setprecision(2) << t << endl;
```

Задатак: Бициклиста

Бициклиста се кретао равномерно, неком константном брзином. На почетку дугачке низбрдице почео је да се креће равномерно убрзано са константним убрзањем. Написати програм којим се одређује колики ће део низбрдице да пређе након једног минута, као и колика ће му бити брзина у том тренутку.

НАПОМЕНА: Код равномерно убрзаног кретања пређени пут се изражава формулом $s = v_0 \cdot t + \frac{a \cdot t^2}{2}$, док се брзина након времена t изражава формулом $v = v_0 + a \cdot t$.

Улаз: У првој линији стандардног улаза се налази реална вредност v_0 која представља брзину изражену у $\frac{m}{s}$, а у следећој линији реална вредност a која представља убрзање изражено у $\frac{m}{s^2}$.

Изназ: Два реална броја, сваки у посебном реду, која представљају пређено растојање s у m и тражену брзину v у $\frac{m}{s}$.

Пример

Улаз	Изназ
10	960.00
0.2	22.00

Решење

Пређени пут и брзину рачунамо на основу датих формула. Како је брзина изражена у $\frac{m}{s}$ и убрзање у $\frac{m}{s^2}$, један минут кретања је потребно у формуле унети као 60 секунди.

```
double v0; cin >> v0;           // pocetna brzina (u m/s)
double a; cin >> a;           // ubrzanje (u m/s^2)
double t = 60;                // vreme (u s)
double s = v0 * t + (a * t * t) / 2; // predjeni put (u m)
double v = v0 + a * t;        // brzina (u m/s)
cout << fixed << showpoint << setprecision(2) << s << endl << v << endl;
```

Задатак: Сустизање аутомобила

Са истог стартног места крећу два аутомобила. Први се креће брзином $v_1 \frac{m}{s}$, други $v_2 \frac{m}{s}$ при чему је $v_2 > v_1$. Други аутомобил полази t минута после првог. Написати програм којим се одређује после колико минута од поласка други аутомобил сустиже први.

Улаз: Са стандардног улаза се уносе три реална броја која редом представљају брзину v_1 првог, брзину v_2 другог аутомобила (изражене у $\frac{m}{s}$) и време t (изражено у min) након којег креће други аутомобил. Сваки број је у посебној линији.

Изназ: На стандардном излазу приказати број минута заокружен на две децимале након којих други аутомобил сустиже први.

Пример

Улаз	Изназ
20	40.00
25	
10	

Решење

Аутомобили се крећу равномерно, тако да ће се решење заснивати на вези између брзине, пређеног пута и времена $v = \frac{s}{t}$ описаној у задатку **Путовање**.

Аутомобили полазе са истог места па до сустизања долази када аутомобили пређу исти пут. Тражено време можемо одредити изједначавањем пређених путева.

За почетак је потребно да све изразимо у истим јединицама. Ако је брзина аутомобила једнака $v \frac{m}{s}$, пошто један минут износи 60 секунди (тј. важи да је $min = 60 s$), брзина је једнака $v \frac{m}{s} \cdot 60 \frac{s}{min} = (v \cdot 60) \frac{m}{min}$.

Обележимо са x број минута после којих други аутомобил сустиже први. Први аутомобил се до сустизања креће $t + x$ минута а други x минута. Пређени пут при равномерном кретању једнак је производу брзине и времена ($s = v \cdot t$). Зато је пут који до сустизања пређе први аутомобил $(t + x) min \cdot ((v_1 \cdot 60) \frac{m}{min}) = (t + x) \cdot v_1 \cdot 60 m$,

а пут који до сустизања пређе други аутомобил $x min \cdot ((v_2 \cdot 60) \frac{m}{min}) = x \cdot v_2 \cdot 60 m$.

Пошто су у тренутку сусрета оба аутомобила прешла исто растојање, изједначавањем пређених путева добијамо једначину $(t + x) \cdot v_1 \cdot 60 \text{ m} = x \cdot v_2 \cdot 60 \text{ m}$, тј. $(t + x) \cdot v_1 = x \cdot v_2$ чије решење је $x = \frac{v_1 \cdot t}{v_2 - v_1}$.

```
double v1, v2; // brzine prvog i drugog automobila (u m/s)
double t;      // kasnjenje drugog automobila (u min)
cin >> v1 >> v2 >> t;
double x = (v1 * t) / (v2 - v1); // broj minuta do sustizanja
cout << fixed << showpoint << setprecision(2) << x << endl;
```

Задатак: Растојање кућа

Пера и Мика живе у истој улици. Микина кућа је удаљенија од школе. Они иду у школу истим путем, полазе из куће у исто време и равномерно се крећу. Пера се креће брзином $v_1 \frac{\text{m}}{\text{s}}$, а Мика брзином $v_2 \frac{\text{m}}{\text{s}}$ ($v_2 > v_1$). Написати програм којим се одређује колико је растојање између њихових кућа, ако се зна да је после t секунди Мика био d метара иза Пера.

Улаз: Са стандардног улаза се уносе четири реална броја, сваки у посебној линији. Они редом представљају брзину кретања Пера (v_1) и брзину кретања Мике (v_2), изражене у $\frac{\text{m}}{\text{s}}$, потом број секунди (t) и растојање између Пера и Мике у метрима (d).

Издаз: На стандардном излазу приказати реалан број, на две децимале, који представља колико је растојање између њихових кућа.

Пример

Улаз	Издаз
1.6	35.00
2.1	
10	
30.0	

Задатак: Колона

Војници се крећу у колони дужине d метара, брзином $v \frac{\text{km}}{\text{h}}$. Командир који се налази на крају колоне, шаље курира на бициклу са поруком на чело колоне. Предавши поруку, курир се одмах враћа назад. Брзина курира бициклисте је $v_k \frac{\text{km}}{\text{h}}$ ($v_k > v$). Написати програм којим се одређује после колико времена се курир враћа на крај колоне.

Улаз: На стандардном улазу налазе се три позитивна реална броја, сваки у посебном реду, који редом представљају дужину колоне у метрима (m), и брзине војника и курира изражене у километрима на час ($\frac{\text{km}}{\text{h}}$).

Издаз: На стандардном излазу исписати време након кога се курир враћа на крај колоне. Време приказати у минутима на две децимале.

Пример

Улаз	Издаз
850	5.26
3.5	
20	

Задатак: Браћа и пас

Два брата налазе се на удаљености d метара, крећу се истим правцем, у супротним смеровима. Први креће се равномерно брзином од $v_1 \frac{\text{km}}{\text{h}}$ а други $v_2 \frac{\text{km}}{\text{h}}$. Пас који је бржи од оба брата равномерно трчи брзином од $v_p \frac{\text{m}}{\text{s}}$. Пас трчи од првог ка другом брату, када стигне до другог окреће се и трчи назад ка првом, када стигне окреће се и трчи ка другом и то понавља све док се браћа не сретну. Напиши програм који одређује колико је пас метара претрчао.

Улаз: На стандардном улазу налазе се четири реална броја, сваки у посебној линији, који редом представљају растојање између браће изражено у m , брзине два брата изражене у $\frac{\text{km}}{\text{h}}$, редом брзина првог брата па брзина другог брата, и брзину пса изражену у $\frac{\text{m}}{\text{s}}$.

Издаз: Исписује се један реалан број заокружен на две децимале који представља број метара које је пас претрчао.

Пример

Улаз	Издаз
500.00	1440.00
3.00	
2.00	
4.00	

1.3 Целобројно дељење

У овом делу збирке позабавићемо се различитим применама целобројног дељења.

1.3.1 Појам целобројног количника и остатка

Број q се назива **целобројни количник** а број r **остатак** при дељењу природних бројева a и b ($b \neq 0$) ако је $a = b \cdot q + r$ и ако је $0 \leq r < b$.

Целобројни количник бројева a и b обележаваћемо са $a \operatorname{div} b$ или са $\lfloor \frac{a}{b} \rfloor$ ($\lfloor \dots \rfloor$ означава заокруживање наниже односно највећи цео број који је мањи или једнак датом броју), док ћемо остатак означавати са $a \operatorname{mod} b$. У каснијим поглављима (види, на пример, задатак **Поклони**) доказаћемо и формално да је $a \operatorname{div} b = \lfloor \frac{a}{b} \rfloor$, тј. доказаћемо да је $q = a \operatorname{div} b$ највећи цео број q такав да је $q \cdot b \leq a$, што оправдава и коришћење ознаке $\lfloor \frac{a}{b} \rfloor$ за целобројни количник бројева a и b .

Ако су операнди цели бројеви, у језику C++ целобројни количник може се израчунати оператором $/$ а остатак при дељењу оператором $\%$.

Задатак: Разломак у мешовит број

За дате природне бројеве a и b написати програм којим се дати неправи разломак $\frac{a}{b}$ преводи у мешовит број $n \frac{c}{b}$, такав да важи да је $\frac{c}{b} < 1$.

Улаз: У првој линији стандардног улаза налази се природан број a који представља бројилац неправог разломка, а у другој линији природан број b различит од нуле који представља именилац разломка ($a \geq b$).

Издаз: Прва и једина линија стандардног издаза треба да садржи мешовити запис разломка, прецизније природан број, бројилац и именилац мешовитог броја међусобно раздвојени по једном празнином (бланком).

Пример

Улаз	Издаз
23	2 7 8
8	

Решење

Мешовит број $n \frac{c}{b}$ представља вредност $n + \frac{c}{b} = \frac{n \cdot b + c}{b}$. У задатку тражимо мешовит број $n \frac{c}{b}$ који је једнак неправом разломку $\frac{a}{b}$ и према томе мора да важи $a = n \cdot b + c$. При том важи и $0 \leq c < b$ (јер важи да је $\frac{c}{b} < 1$, и зато је $c < b$, а уз то важи и $c \geq 0$). Зато је, на основу дефиниције целобројног количника, цео део n мешовитог броја једнак целобројном количнику при дељењу бројоца a имениоцем b , бројилац c разломљеног дела мешовитог броја је остатак при дељењу a са b , док именилац b разломљеног дела мешовитог броја остаје исти као у полазном разломку.

```
int a, b;           // polazni brojilac i imenilac
cin >> a >> b;
int n = a / b;     // ceo deo
int c = a % b;    // novi brojilac (imenilac ostaje b)
cout << n << " " << c << " " << b << endl;
```

Задатак: Врста и колона

Спортисти на дресу имају бројеве $0, 1, 2, \dots$. У том редоследу се ређају за дефиле за свечано отварање такмичења у коме корачају организовани у врсте од по 5. Напиши програм који на основу редног броја дреса одређује у којој ће се врсти и колони спортиста налазити (врсте и колоне се броје од један).

Улаз: Са улаза се уноси цео број x ($0 \leq x \leq 1000$) који представља број дреса спортисте.

Излаз: На стандардни излаз исписати број врсте и колоне раздвојене размаком.

Пример

Улаз Излаз
17 4 3

Објашњење

0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
20 21 ...

Као што се са слике види, спортиста са дресом 17 је у врсти број 4 и колони број 3.

Решење

Ако се спортиста са редним бројем x налази у врсти v и колони k тада се испред њега налази $(v-1) \cdot 5$ спортиста из претходних врста и $k - 1$ спортиста из претходних колона текуће врсте. Зато је $x = 5(v - 1) + (k - 1)$. Пошто $k - 1$ може бити 0, 1, 2, 3 или 4, важи да је $k - 1 = x \bmod 5$ и да је $v - 1 = x \operatorname{div} 5$. Заиста, елементи прве колоне су бројеви 0, 5, 10 итд. који сви имају остатак 0 при дељењу са 5, елементи друге колоне су бројеви 1, 6, 11 итд. који сви имају остатак 1 при дељењу са 5 итд. Слично, елементи прве врсте су бројеви од 0 до 4 који при целобројном дељењу са 5 дају количник 0, елементи друге врсте су бројеви од 5 до 9 који при целобројном дељењу са 5 дају количник 1 итд.

```
int x;
cin >> x;
cout << x / 5 + 1 << " " << x % 5 + 1 << endl;
```

Задатак: Шаховска табла број црних поља

На шаховској табли ширине n и дужине m , горње лево поље је беле боје. Написати програм којим се одређује број црних поља на датој шаховској табли.

Улаз: На стандардном улазу налазе се два природна броја $n, m \leq 100$.

Излаз: Број црних поља на шаховској табли.

Пример

Улаз Излаз
5 7
3

Решење

На шаховској табли димензије $n \times m$ укупан број поља је $n \cdot m$. У сваком реду боја поља се наизменично мења. Редови почињу наизменично белом па црном бојом (први ред почиње белом бојом, други црном бојом, трећи ред белом, ...). Ако је укупан број поља паран онда црних и белих поља има једнак број, а ако је укупан број поља непаран број белих поља је за 1 већи од броја црних поља (јер први ред почиње белим пољем). Према томе број црних поља једнак је целобројном количнику укупног броја поља и броја 2 тј. $(n \cdot m) \operatorname{div} 2 = \lfloor \frac{n \cdot m}{2} \rfloor$. Подсетимо се да оператор / израчунава целобројни количник када су оба операнда цели бројеви.

```
int n, m;
cin >> n >> m;
int brCrnihPolja = (n * m) / 2;
cout << brCrnihPolja << endl;
```

Задатак: Икс-окс

Мирко је мали програмер који покушава да испрограмира игрицу икс-окс. Близу је завршетка, али му је потребна мала помоћ. Смислио је да корисник мишем одређује квадрат у који ће се његов симбол уписати. Поље за игру се састоји од 9 квадрата (распоређена у три врсте и три колоне) и сваки квадрат је димензије 100 пута 100 пиксела (укупно поље је димензије 300 пута 300 пиксела). Познат је положај пиксела на који је кликнуто мишем и потребно је одредити редни број квадрата у којем се тај пиксел налази. Положај пиксела је одређен редним бројевима (координатама) тог пиксела по хоризонтали и по вертикали, рачунајући од доњег

1.3. ЦЕЛОБРОЈНО ДЕЉЕЊЕ

левог угла поља (пиксели се броје од 1 до 300). Квадрати се броје од 1 до 9, врсту по врсту, почевши од доњег левог угла поља, како је приказано на следећој слици:

7	8	9
4	5	6
1	2	3

Улаз: Са стандардног улаза се учитавају два цела броја (сваки у посебном реду) x и y ($1 \leq x, y \leq 300$) који представљају x тј. y у координату пиксела на који је кликнуто мишем.

Излаз: На стандардни излаз се исписује један број од 1 до 9 који представља редни број квадрата.

Пример 1

<i>Улаз</i>	<i>Излаз</i>
1	1
1	

Пример 2

<i>Улаз</i>	<i>Излаз</i>
120	8
280	

Пример 3

<i>Улаз</i>	<i>Излаз</i>
100	7
201	

Пример 4

<i>Улаз</i>	<i>Излаз</i>
101	5
300	

Задатак: По два и по три

Људи иду путем у колони два по два. На месту где је тротоар широк, престројили су се тако да иду три по три. При томе је редослед људи остао исти ако се посматрају ред по ред слева на десно.

Напиши програм који одређује у ком реду и колони се после престројавања налази човек који је на почетку био у реду R и колони K (врсте и колоне се броје од један).

Улаз: У првом реду улаза је цео број R ($1 \leq R \leq 1000$), а у другом реду цео број K ($1 \leq K \leq 2$).

Излаз: На стандардни излаз исписати број врсте и колоне после престројавања, раздвојене размаком.

Пример 1

<i>Улаз</i>	<i>Излаз</i>
2	1 3
1	

Објашњење

a b	a b X
X d	d e f
e f	g h ...
g h	
...	

Човек који је био први у другом реду (означен словом X), након престројавања ће бити трећи у првом реду.

Пример 2

Улаз

5
2

Излаз

4 1

Објашњење

a b	a b c
c d	d e f
e f	g h i
g h	X k l
i X	...
k l	
...	

Човек који је био други у петом реду (означен словом X), након престројавања ће бити први у четвртном реду.

1.3.2 Позициони запис

У позиционом запису бројева допринос цифре укупној вредности броја не зависи само од вредности цифре, већ и од њене позиције у запису. Запис $c_n c_{n-1} \dots c_0$ у позиционом систему са основом b (то некад обележавамо са $(c_n c_{n-1} \dots c_0)_b$) одговара броју

$$c_n \cdot b^n + c_{n-1} \cdot b^{n-1} + \dots + c_1 \cdot b + c_0,$$

при чему за сваку цифру c_i важи $0 \leq c_i < b$.

На пример број 1234 у основи 10 једнак је $1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10 + 4$.

Најчешће коришћени бројевни систем јесте декадни систем, односно систем са основом 10. Број x се у декадном систему може представити у облику $x = c_n \cdot 10^n + c_{n-1} \cdot 10^{n-1} + \dots + c_1 \cdot 10 + c_0$, где је c_0 цифра јединица, c_1 цифра десетица, c_2 цифра стотина итд. и за сваку од њих важи $0 \leq c_i \leq 9$.

За запис времена и углова користи се позициони запис у основи 60 (сат тј. угао има 60 минута, док један минут има 60 секунди).

Вредност броја се може одредити и помоћу Хорнерове шеме

$$(\dots((c_n \cdot b + c_{n-1}) \cdot b + c_{n-2}) \cdot b + \dots + c_1) \cdot b + c_0.$$

На пример број 1234 у основи 10 једнак је $((1 \cdot 10 + 2) \cdot 10 + 3) \cdot 10 + 4$.

Последња цифра у декадном запису броја може се одредити операцијом израчунавања остатка при дељењу са 10. На пример, последња цифра броја 1234 је 4, што је управо остатак при дељењу тог броја са 10. Слично, последња цифра записа броја у основи b може се одредити операцијом израчунавања остатка при дељењу броја са b . Докажимо ово и формално. Број x , чије су цифре редом $c_n c_{n-1} \dots c_1 c_0$, представља се у облику $x = c_n \cdot b^n + c_{n-1} b^{n-1} + \dots + c_1 \cdot b + c_0$. Пошто су сви сабирци осим последњег дељиви са b , тј. број се може написати у облику $x = (c_n \cdot b^{n-1} + c_{n-1} \cdot b^{n-2} + \dots + c_1) \cdot b + c_0$ и пошто је $0 \leq c_0 < b$, на основу дефиниције целобројног количника и остатка важи да је $x \bmod b$ једнако c_0 .

Општије, цифру c_k уз коефицијент b^k у запису броја x можемо одредити као

$$(x \operatorname{div} b^k) \bmod b.$$

Докажимо и ово формално. Докажимо да важи да је $x \operatorname{div} b^k = c_n \cdot b^{n-k} + c_{n-1} \cdot b^{n-k-1} + \dots + c_{k+1} \cdot b + c_k$. Заиста, важи да је $x = (c_n \cdot b^{n-k} + c_{n-1} \cdot b^{n-k-1} + \dots + c_{k+1} \cdot b + c_k) \cdot b^k + c_{k-1} b^{k-1} + \dots + c_1 b + c_0$. Пошто за сваку цифру c_i важи $0 \leq c_i \leq b-1$, важи да је $c_{k-1} b^{k-1} + \dots + c_1 b + c_0 \leq (b-1) \cdot (b^{k-1} + \dots + b + 1) = b^k - 1$. Зато је $x \bmod b^k = c_{k-1} b^{k-1} + \dots + c_1 b + c_0$, док је $x \operatorname{div} b^k = c_n \cdot b^{n-k} + c_{n-1} \cdot b^{n-k-1} + \dots + c_{k+1} \cdot b + c_k$. Зато се до цифре c_k може доћи одређивањем остатка при дељењу овог броја са b (сви сабирци осим последњег су дељиви са b , док је $0 \leq c_k < b$).

Задатак: Збир цифара четвороцифреног броја

Напиши програм који учитава четвороцифрени број и исписује збир свих цифара тог броја.

Улаз: У једној линији стандардног улаза налази се четвороцифрени број.

Излаз: Збир цифара четвороцифреног броја.

Пример

Улаз	Излаз
1234	10

Решење

Збир цифара четвороцифреног броја се добија издвајањем цифара јединица, десетица, стотина и хиљада, које се затим саберу. Цифре броја можемо одредити на неколико различитих начина.

1.3. ЦЕЛОБРОЈНО ДЕЉЕЊЕ

Кључна опаска је увек то да се последња цифра броја може одредити операцијом израчунавања остатка при дељењу са 10. На пример, последња цифра броја 1234 је 4, што је управо остатак при дељењу тог броја са 10.

Посебан израз за сваку цифру

До цифре јединица c_0 броја n може се лако доћи као $n \bmod 10$. На пример, цифра јединица броја 1234 је 4 и она се лако добија као остатак при дељењу броја 1234 са 10.

Последња цифра броја се лако може уклонити тако што се израчуна целобројни количник тог броја са 10. На пример, целобројни количник при дељењу броја 1234 са 10 је 123. Цифру десетица полазног броја можемо добити као цифру јединица броја који се добија када му се избрише последња цифра, а управо смо видели да је то број $n \div 10$. На пример, цифра десетица број 1234 је цифра јединица броја 123 и она је једнака $(n \div 10) \bmod 10$.

Последње две цифре броја се могу уклонити тако што се израчуна количник тог броја при дељењу са 100. На пример, целобројни количник при дељењу броја 1234 са 100 је 12. Цифру стотина полазног броја можемо добити као цифру јединица броја који се добија када му се избришу последње две цифре, а управо смо видели да је то број $n \div 100$. На пример, цифра стотина броја 1234 је цифра јединица броја 12 и она је једнака $(n \div 100) \bmod 10$.

Последње три цифре броја се могу уклонити тако што се израчуна количник тог броја при дељењу са 1000. На пример, целобројни количник при дељењу броја 1234 са 1000 је 1. Цифру хиљада полазног броја можемо добити као цифру јединица броја који се добија када му се избришу последње три цифре, а управо смо видели да је то број $n \div 1000$. На пример, цифра хиљада броја 1234 је цифра јединица броја 1 и она је једнака $(n \div 1000) \bmod 10$.

Уочава се веома јасна правилност.

```
// polazni broj i njegovo ucitavanje
int broj;
cin >> broj;

// izracunavanje zbira cifara
int cifraJedinica = (broj / 1) % 10;
int cifraDesetica = (broj / 10) % 10;
int cifraStotina = (broj / 100) % 10;
int cifraHiljada = (broj / 1000) % 10;
int zbirCifara = cifraJedinica + cifraDesetica +
                cifraStotina + cifraHiljada;

// prikaz rezultata
cout << zbirCifara << endl;
```

Види групачија решења овој задатку.

Задатак: Јарди, стопе и инчи

У САД се дужина мери у јардима, стопама и инчима. Један јард има 3 стопе, а једна стопа има 12 инча. Написати програм који дужину унету у јардима, стопама и инчима прерачунава само у инче.

Улаз: Са стандардног улаза се уносе три цела броја, сваки у посебном реду. Број j ($0 \leq j \leq 10$) представља број јарди, број s ($0 \leq s \leq 2$) представља број стопа, а број i ($0 \leq i \leq 11$) представља број инча.

Излаз: На стандардни излаз исписати само један цео број који представља унету дужину изражену само у инчима.

Пример

Улаз	Излаз
7	281
2	
5	

Решење**Класична конверзија**

Класичан поступак конверзије подразумева да се израчуна колико у једном јарду има инча ($3 \cdot 12 = 36$) и да се тај број помножи бројем јарди, да се број инча у једној стопи (12) помножи бројем стопа и на крају да се на збир ова два броја дода број инча. Тиме добијамо формулу $j \cdot 36 + s \cdot 12 + i$.

```
int jardi, stope, inci;
cin >> jardi >> stope >> inci;
int uIncima = jardi * 36 + stope * 12 + inci;
cout << uIncima << endl;
```

Хорнерова схема

У основи претходне формуле лежи један поступак који нам омогућава да поступно извршимо конверзију. Прво можемо да се запитамо колико стопа има у уčitаној дужини. Пошто у једној јарди има 3 стопе у j јарди имамо $j \cdot 3$ стопа и поред тога имамо још s стопа. Укупан број стопа је зато једнак $j \cdot 3 + s$.

Сада се можемо запитати колико укупно имамо инча. У свакој од стопа које смо мало пре израчунали има 12 инча и имамо још преосталих i инча. Зато је укупан број инча једнак $(j \cdot 3 + s) \cdot 12 + i$.

Овакав запис се назива Хорнерова схема и она се често користи за вредности бројева и полинома. На пример, вредност броја $123 = 1 \cdot 100 + 2 \cdot 10 + 3$, се Хорнеровом схемом може израчунати као $((1 \cdot 10) + 2) \cdot 10 + 3$, па и као $((0 \cdot 10 + 1) \cdot 10 + 2) \cdot 10 + 3$. Осим што је понекада јаснија, предности Хорнерове схеме у односу на класичан поступак израчунавања биће јасно илустровано у поглављу о итеративним алгоритмима.

```
int jardi, stope, inci;
cin >> jardi >> stope >> inci;
int uIncima = ((jardi * 3) + stope) * 12 + inci;
cout << uIncima << endl;
```

Задатак: Октални бројеви

У рачунарству се често користе тзв. октални бројеви - бројеви записани у основи 8, коришћењем само цифара од 0 до 7. Напиши програм који врши конверзију четвороцифрених окталних бројева у декадне вредности и обратно.

Улаз: Са стандардног улаза се учитавају 4 окталне цифре (свака у посебном реду, почевши од цифре највеће тежине) и након тога декадно записан природан број n ($0 \leq n < 8^4$).

Издаз: На стандардном издазу у првој линији исписати декадну вредност броја формираног од учитаних окталних цифара, а у другој линији четвороцифрену окталну репрезентацију броја n .

Пример

Улаз	Издаз
1	668
2	2322
3	
4	
1234	

Решење

За вредности броја на основу цифара се може употребити алгоритам заснован на класичној дефиницији позиционог записа (тада број израчунавамо као $c_3 \cdot 8^3 + c_2 \cdot 8^2 + c_1 \cdot 8^1 + c_0 \cdot 8^0$), док цифру на позицији i можемо израчунати као $(n \div 8^i) \bmod 8$.

Приликом имплементације је могуће дефинисати и користити функцију која на основу броја одређује цифре и функцију која на основу цифара одређује вредност броја (наравно, то није неопходно, али једном када се те функције дефинишу оне могу бити употребљене у разним задацима). Функција која одређује цифре броја враћа више вредности. У језику C++ то је могуће постићи тако што излазне параметре (резултујуће цифре) преносимо по референци (непосредно иза типа наведемо &).

```
// odredjivanje vrednosti oktalnog zapisa (c3c2c1c0)8
int izOktalnog(int c3, int c2, int c1, int c0) {
```

1.3. ЦЕЛОБРОЈНО ДЕЉЕЊЕ

```
    return c3*8*8*8 + c2*8*8 + c1*8 + c0;
}

// prevodjenje broja n u oktalni zapis (c3c2c1c0)8
void uOktalni(int n, int& c3, int& c2, int& c1, int& c0) {
    c0 = (n / (1)) % 8;
    c1 = (n / (8)) % 8;
    c2 = (n / (8*8)) % 8;
    c3 = (n / (8*8*8)) % 8;
}

int main() {
    // izracunavanje vrednosti oktalno zapisanog broja
    int c3, c2, c1, c0;
    cin >> c3 >> c2 >> c1 >> c0;
    cout << izOktalnog(c3, c2, c1, c0) << endl;

    // izracunavanje cifara broja koji treba zapisati oktalno
    int n;
    cin >> n;
    uOktalni(n, c3, c2, c1, c0);
    cout << c3 << c2 << c1 << c0 << endl;
    return 0;
}
```

Види групачија решења овој задатка.

Задатак: Избаци цифру стотина

Напиши програм који из декадног записа броја избацује цифру стотина.

Улаз: Са стандардног улаза уноси се један природан број мањи од милијарду.

Излаз: На стандардни излаз исписује се број добијен избацивањем цифре стотина из декадног записа унетог броја.

Пример

Улаз	Излаз
123456	12356

Решење

Пре избацивања цифре, одредимо префикс броја пре те цифре (део записа броја испред ње) и суфикс броја након те цифре (део записа броја иза ње). На пример, ако је број 123456, префикс пре цифре стотина је 123, а суфикс након ње је 56. Префикс се може одредити као целобројни количник овог броја са 1000, а суфикс као остатак при дељењу са 100. На крају ћемо на префикс дописати цифре суфикса. Пошто је суфикс двоцифрен, то ћемо урадити тако што ћемо префикс помножити са 100 и сабрати са суфиксом.

Докажимо ово и формално. Претпоставимо да је n облика $(c_k c_{k-1} \dots c_3 c_2 c_1 c_0)_{10}$, тј.

$$n = c_k \cdot 10^k + \dots c_3 \cdot 10^3 + c_2 \cdot 10^2 + c_1 \cdot 10 + c_0$$

Овај број се може записати у облику $(c_k \cdot 10^{k-3} + \dots + c_3) \cdot 10^3 + c_2 \cdot 10^2 + c_1 \cdot 10 + c_0$ и пошто важи да је $0 \leq c_2 \cdot 10^2 + c_1 \cdot 10 + c_0 < 1000$ (јер су све цифре између 0 и 9), на основу дефиниције целобројног количника, важи да је $n \operatorname{div} 1000 = c_k \cdot 10^{k-3} + \dots + c_3$. Потпуно аналогно се доказује да је $n \operatorname{mod} 100 = c_1 \cdot 10 + c_0$. На крају, важи да је

$$100 \cdot (n \operatorname{div} 1000) + n \operatorname{mod} 100 = c_k \cdot 10^{k-1} + \dots c_3 \cdot 10^2 + c_1 \cdot 10 + c_0$$

што је број $(c_k c_{k-1} \dots c_3 c_1 c_0)_{10}$ који се добија од полазног брисањем цифре c_2 .

```
int n;
cin >> n;
int prefiks = n / 1000;
int sufiks = n % 100;
cout << 100*prefiks + sufiks << endl;
```

Задатак: Поноћ

Напиши програм који за дато време у току дана одређује колико је секунди протекло од претходне поноћи и колико је сати, минута и секунди остало до следеће поноћи.

Улаз: Са стандардног улаза уносе се три цела броја (сваки у посебном реду) који одређују један временски тренутак:

- h - сат ($0 \leq h < 24$)
- m - минут ($0 \leq m < 60$)
- s - секунд ($0 \leq s < 60$)

Израз: На стандардни излаз исписати две линије. У првој исписати цео број који представља број секунди протеклих од претходне поноћи. У другој исписати три цела броја раздвојена двотачкама - број сати h_p ($0 \leq h_p < 24$), број минута m_p ($0 \leq m_p < 60$) и број секунди s_p ($0 \leq s_p < 60$) до наредне поноћи.

Пример

Улаз	Израз
10	37432
23	13:36:8
52	

Решење

Изражавање времена преко броја секунди од почетка дана (претходне поноћи)

Један веома често коришћен начин је да се пре израчунавања све преведе у најмање јединице, а да се након израчунавања изврши прерачунавање у полазне јединице. То би подразумевало да се текући тренутак и наредна поноћ претворе у број секунди протекло од претходне поноћи, да се израчуна број секунди до поноћи и да се затим то време изражено само у секундама изрази у сатима, минутима и секундама.

Ако је време одређено бројевима h , m и s који представљају број сати, минута и секунди, тада се време протекло од почетка дана може израчунати тако што се саберу број сати помножен са $60 \cdot 60$ (јер у једном сату има 60 минута, а сваки од њих има 60 секунди), број минута помножен са 60 (јер у једном минуту има 60 секунди) и број секунди, тј. израчунавањем вредности $h \cdot 60^2 + m \cdot 60 + s$, што може да се уради било директно, било Хорнеровим поступком, како је објашњено у задатку **Јарди, стопе и инчи**. Приметимо да се овде ради о запису у бројевној основи 60. Хорнеров поступак би овде кретао прво од броја целих сати h , затим израчунавао број целих минута тако што би помножио број сати h са 60 и на то додао број m , а затим број секунди добио множећи тај број минута са 60 и сабирајући са s (тј. израчунао би вредност израза $(h \cdot 60 + m) \cdot 60 + s$). Наредна поноћ се може представити у облику времена 24:00:00 и она се може превести у секунде на потпуно исти начин (изразом $24 \cdot 60^2$).

Када се изврши одузимање, тада је добијени број секунди (рецимо да је то t_s) потребно претворити у сате, минуте и секунде. То се може урадити како је описано у задатку **Збир цифара четвороцифреног броја**, који укључују израчунавање остатака и количника при целобројном дељењу са бројевном основом која је у овом случају 60. Последња цифра (вредност s) може се одредити као остатак при дељењу броја t_s са 60 тј. $s = t_s \bmod 60$, претпоследња цифра m као остатак при дељењу са 60 целобројног количника бројева t_s и 60 тј. $m = (t_s \div 60) \bmod 60$, а прва цифра h као целобројни количник бројева t_s и 60^2 тј. $h = t_s \div 60^2$. Ако у полазном броју секунди има више од једног дана (у овом задатку то није могуће, али у неким другим задацима ће бити), тада је потребно још одредити остатак броја сати при дељењу са 24. Овим се, дакле, добија следећи поступак.

$$\begin{aligned} s &= t_s \bmod 60 \\ m &= (t_s \div 60) \bmod 60 \\ h &= (t_s \div 60^2) \bmod 24 \end{aligned}$$

```
// sat, minut i sekund
int h, m, s;
```

1.3. ЦЕЛОБРОЈНО ДЕЉЕЊЕ

```
cin >> h >> m >> s;

// broj sekundi od prethodne ponoci
int S = h*60*60 + m*60 + s;
cout << S << endl;

// broj sekundi od prethodne ponoci do sledece ponoci
int Sponoc = 24*60*60;
// broj sekundi preostalih do ponoci
int S_do_ponoci = Sponoc - S;
// prevodimo to vreme u sate, minute i sekunde
int s_do_ponoci = (S_do_ponoci / (1)) % 60;
int m_do_ponoci = (S_do_ponoci / (60)) % 60;
int h_do_ponoci = (S_do_ponoci / (60*60)) % 24;

// ispisujemo rezultat
cout << h_do_ponoci << ":" << m_do_ponoci << ":" << s_do_ponoci << endl;
```

Приликом имплементације је могуће дефинисати и користити функције конверзије сати, минута и секунди у секунде и из секунди у сате, минуте и секунде (наравно, то није неопходно, али једном када се те функције дефинишу оне могу бити употребљене у разним задацима). Функција која конвертује минуте у сате и минуте враћа више вредности. У језику С++ то је могуће постићи тако што излазне параметре (резултујући број сати и минута) преносимо по референци (иза типа наведемо &).

```
// za trenutak h:m:s odredjuje se broj sekundi S proteklih od
// prethodne ponoci
int uSekunde(int h, int m, int s) {
    return h*60*60 + m*60 + s;
}

// za dati broj sekundi S proteklih od prethodne ponoci,
// odredjuje se vremenski trenutak (sat h, minut m i sekund s)
void odSekundi(int S, int& h, int& m, int& s) {
    s = (S / (1)) % 60;
    m = (S / (60)) % 60;
    h = (S / (60*60)) % 24;
}

int main() {
    // dan, minut i sekund
    int h, m, s;
    cin >> h >> m >> s;

    // broj sekundi od prethodne ponoci
    int S = uSekunde(h, m, s);
    cout << S << endl;

    // broj sekundi od prethodne ponoci do sledece ponoci
    int Sponoc = uSekunde(24, 0, 0);
    // broj sati, minuta i sekundi do naredne ponoci
    int h_do_ponoci, m_do_ponoci, s_do_ponoci;
    odSekundi(Sponoc - S, h_do_ponoci, m_do_ponoci, s_do_ponoci);
    cout << h_do_ponoci << ":" << m_do_ponoci << ":"
        << s_do_ponoci << endl;
    return 0;
}
```

Види групачија решења овој задатка.

Задатак: Трајање вожње

Познати су сат, минут и секунд почетка и краја вожње аутобусом (вожња почиње и завршава се у једном дану). Написати програм који одређује колико сати, минута и секунди је трајала та вожња.

Улаз: Са стандардног улаза учитава се 6 бројева (сваки у засебном реду). Прво сат, минут и секунд почетка вожње, а затим сат, минут и секунд краја вожње. Сати су из интервала $[0, 24)$, а минути и секунди из интервала $[0, 60)$.

Излаз: На стандардни излаз се исписује један ред у коме су три броја раздвојена двотачком: h ($0 \leq h < 24$) - број сати, m ($0 \leq m < 60$) број минута и s ($0 \leq s < 60$) број секунди трајања вожње.

Пример

Улаз	Излаз
2	0:2:6
59	
8	
3	
1	
14	

Решење

Трајање вожње израчунава се тако што се од времена доласка одузме време поласка. Задатак, дакле, захтева израчунавање разлике између два временска тренутка за која знамо да су у оквиру једног дана.

Изражавање времена преко броја секунди од почетка дана

Један од уобичајених начина рада са временом је да се сваки временски тренутак изрази преко броја секунди протеклих од почетка тог дана (тј. од претходне поноћи). Ако је h број сати, m број минута и s број секунди, оба смера конверзије можемо реализовати као у задатку **Поноћ**.

```
// pocetak i kraj voznje
```

```
int hPocetak, mPocetak, sPocetak;
cin >> hPocetak >> mPocetak >> sPocetak;
int hKraj, mKraj, sKraj;
cin >> hKraj >> mKraj >> sKraj;
```

```
// trajanje voznje
```

```
int sPocetak = uSekunde(hPocetak, mPocetak, sPocetak);
int sKraj = uSekunde(hKraj, mKraj, sKraj);
int hTrajanje, mTrajanje, sTrajanje;
odSekundi(sKraj - sPocetak, hTrajanje, mTrajanje, sTrajanje);
```

```
cout << hTrajanje << ":" << mTrajanje << ":" << sTrajanje << endl;
```

Види групачија решења овој задатку.

Задатак: UNIX време

Од оперативног система UNIX време у рачунарима се изражава као број секунди протеклих од почетка епохе тј. од 1. јануара 1970. године. По узору на то, осмислили смо систем мерења времена у коме се време изражава бројем милисекунди протеклих од укључивања рачунара. У неком тренутку, на рачунару је пуштена песма. Ако је познато време када је песма пуштена и дужина трајања песме у милисекундама, напиши програм који одређује када је песма завршена.

Улаз: Са стандардног улаза учитавају се следећи цели бројеви (сваки у посебном реду):

- dan ($0 \leq dan \leq 10$), sat ($0 \leq sat < 24$), min ($0 \leq min < 60$), sek ($0 \leq sek < 60$), $mili$ ($0 \leq mili < 1000$) - број дана, сати, минута, секунди и милисекунди протеклих од тренутка укључивања рачунара до пуштања песме.
- $trajanje$ ($0 \leq trajanje \leq 1000000$) - број милисекунди колико траје песма.

Излаз: На стандардни излаз исписати следеће целе бројеве, раздвојене двотачкама:

1.3. ЦЕЛОБРОЈНО ДЕЉЕЊЕ

- dan ($0 \leq dan \leq 100$), sat ($0 \leq sat < 24$), min ($0 \leq min < 60$), sek ($0 \leq sek < 60$), $mili$ ($0 \leq mili < 1000$) - број дана, сати, минута, секунди и милисекунди протеклих од тренутка укључивања рачунара до завршетка песме.

Пример 1

Улаз	Израз
3	3:10:17:3:843
10	
15	
23	
843	
100000	

Пример 2

Улаз	Израз
4	5:0:0:11:862
23	
59	
59	
517	
12345	

Пример 3

Улаз	Израз
10	11:0:16:39:999
23	
59	
59	
999	
1000000	

Решење

Као и обично, увешћемо функције за конверзију времена датог у данима, сатима, минутима, секундама и милисекундама у број милисекунди од укључивања рачунара и броја милисекунди у време, све свести на милисекунде, извршити тражену операцију сабирања времена и на крају, милисекунде превести у време у данима, сатима, минутима, секундама и милисекундама. Сличну технику, али само за сате, минуте и секунде, применили смо, на пример, у задатку [Поноћ](#).

Најприроднији начин да време преведемо у милисекунде је да употребимо Хорнерову шему, коју смо описали у задатку [Јарди, стопе и инчи](#). Укупан број сати можемо добити тако што број дана помножимо са 24 (јер је толико сати у једном дану) и додамо број сати. Укупан број минута можемо добити тако што тако добијени број сати помножимо са 60 (јер је толико минута у једном сату) и додамо број минута. Укупан број секунди можемо добити тако што тако добијени број минута помножимо са 60 (јер је толико секунди у једном минуту) и додамо број секунди. На крају, укупан број милисекунди можемо добити тако што добијени број секунди помножимо са 1000 (јер је толико милисекунди у једној секунди) и додамо број милисекунди. Дакле, ако променљиве dan , sat , min , sek и $mili$ представљају број дана, сати, минута, секунди и милисекунди, тада конверзију можемо извршити помоћу израза

$$(((dan \cdot 24 + sat) \cdot 60 + min) \cdot 60 + sek) \cdot 1000 + mili$$

Ово је много лепше решење него да, на пример, унапред рачунамо колико милисекунди има у једном сату.

Конверзија у супротном смеру тече тако што се рачуна једна по једна цифра у запису у овој мешовитој основи (слично као у задацима [Збир цифара четвороцифреног броја](#) и [Поноћ](#)) (касније ћемо видети да и за ово постоји елегантније решење).

```
// преводи dane, sate, minute, sekunde i milisekunde u milisekunde
int uMilisekunde(int dan, int sat, int min, int sek, int mili) {
    // Hornerova sema
    return (((dan*24 + sat)*60 + min)*60 + sek)*1000 + mili;
}

// преводи milisekunde u dane, sate, minute, sekunde i milisekunde
void odMilisekundi(int v, int& dan, int& sat, int& min,
                  int& sek, int& mili) {
    mili = (v / 1) % 1000;
    sek = (v / 1000) % 60;
    min = (v / (1000*60)) % 60;
    sat = (v / (1000*60*60)) % 24;
    dan = (v / (1000*60*60*24));
}
```

Види групаџија решења овој задатку.

Задатак: Угао сатне казаљке

Ако је дат угао који сатна казаљка заклапа са вертикалном полуправом која спаја центар и врх сата (број 12), одредити време у сатима и минутима.

Улаз: Са стандардног улаза учитава се цео број који представља угао у степенима.

Излаз: На стандардни излаз исписати два цела броја h ($0 \leq h < 12$) и m ($0 \leq m < 60$) који представљају број сати и минута који су најближи положају сатне казаљке.

Пример		Пример 2	
Улаз	Излаз	Улаз	Излаз
90	3:0	200	6:40

Решење

За разлику од секундне казаљке која се креће дискретно (поскочи сваке секунде), сатна казаљка се обично креће непрекидно. Ипак, пошто у овом задатку не разматрамо секунде, већ само минуте, претпоставићемо да се сатна казаљка помера само једном у минути (на сваком пуном минути). Дакле, положај сатне казаљке ће бити одређен временом протеклим од претходног тренутка када је сатна казаљка била усмерена на горе (а то је претходна поноћ или претходно подне) израженим у целом броју минута.

Ако се зна колико је тренутно сати и минута, тај број се може лако израчунати (број сати се помножи са 60 и сабере се са бројем минута, слично као у задатку **Поноћ**). Пошто угао казаљке линеарно зависи од броја протеклих минута, и пошто на сваких 12 сати сатна казаљка направи пун круг, тј. опише угао од 360 степени, важи пропорција

$$\alpha^\circ : M = 360^\circ : 12 \text{ h},$$

где је M број минута протеклих од претходног тренутка када је сатна казаљка заклапала 0° (а то је претходно подне или поноћ). M се, дакле, може израчунати тако што се тих 12h (тј. 12 сати и 0 минута) претвори у број минута протеклих од претходне поноћи или поднева и реши се пропорција

$$M = \frac{\alpha^\circ \cdot (12 \cdot 60 \text{ min})}{360^\circ} = \frac{\alpha^\circ \cdot 720 \text{ min}}{360^\circ} = \alpha \cdot 2 \text{ min}.$$

Када се зна број минута протеклих од претходног поднева или поноћи, време се лако може израчунати (остатак при дељењу са 60 даје број минута, а количник број сати, слично као у задатку **Поноћ**).

```
// за dati broj sati i minuta odredjuje broj minuta od prethodnih
// 00:00 sati
int uMinute(int h, int m) {
    return 60*h + m;
}

// за dati broj minuta proteklih od prethodnih 00:00 sati odredjuje
// broj sati i minuta
void odMinuta(int M, int& h, int& m) {
    m = M % 60;
    h = M / 60;
}

int main() {
    // alfa - ugao kazaljke u celom broju stepeni
    int alfa;
    cin >> alfa;
    // M - broj minuta proteklih od prethodnih 00:00 sati
    int M = alfa * uMinute(12, 00) / 360;
    // broj sati i minuta koji odgovaraju tom broju minuta
    int h, m;
    odMinuta(M, h, m);
    // ispisi rezultata
    cout << h << ":" << m << endl;
    return 0;
}
```

Један начин да се дође до решења је да се закључи да, пошто за 12 сати сатна казаљка обиђе угао од 360 степени, она за један сат обиђе дванаестину тог угла тј. угао од $\frac{360^\circ}{12} = 30^\circ$. Пошто један сат има 60 минута,

1.3. ЦЕЛОБРОЈНО ДЕЉЕЊЕ

за један временски минут она обиђе шездесетину угла који пређе за сат времена, тј. угао од $\frac{30^\circ}{60} = 0.5^\circ$. Дакле, пошто за један временски минут казаљка пређе пола степена, једном степену одговара 2 минута времена. Број минута протеклих од поднева или поноћи се онда може израчунати множењем датог угла у степенима са 2 (минута по степену).

```
// ugao kazaljke u realnom broju stepeni
int alfa;
cin >> alfa;
// M - broj minuta proteklih od prethodnih 00:00 sati
// posto u 12 sati ima 12*60 = 720 minuta, vazi proporcija
// tj. M : alfa = 720 : 360 = 2 : 1
int M = alfa * 2;
// broj sati i minuta koji odgovaraju tom broju minuta
int h, m;
odMinuta(M, h, m);
// ispis rezultata
cout << h << ":" << m << endl;
```

Задатак: Угао између казаљки

Одредити угао у степенима и минутима између сатне и минутне казаљке ако је задато време у сатима и минутима. Угао је увек позитиван и мери се у смеру кретања казаљки на сату (зато може да буде и већи од 180 степени).

Улаз: Учитавају се два броја, сваки у посебној линији. У првој линији је број сати (између 0 и 12), а у другој линији је број минута (између 0 и 59).

Израз: Степен угла 0 до 359, и минут угла од 0 до 59, раздвојени двотачком тј. карактером `:`.

Пример 1

Улаз	Израз
1	135:0
30	

Пример 2

Улаз	Израз
12	165:0
30	

Решење

Нека је дати временски тренутак описан параметрима *sat* и *minut*.

Угао који минутна казаљка заклапа у односу на почетни положај од нула минута (такозвани угаони отклон минутне казаљке) једнак је $minut \cdot 360'$, где $'$ означава угаони минут. Заиста, на сваки минут времена минутна казаљка се помера за $\frac{360^\circ}{60} = 6^\circ$. Дакле, 6° је померање минутне казаљке у степенима по сваком временском минуту, а пошто у једном степену има 60 угаоних минута тј. пошто је $1^\circ = 60'$, за сваки минут времена минутна казаљка се помери за $6^\circ \cdot \frac{60'}{1^\circ} = 360'$, тј. за 360 угаоних минута.

Угао који у датом временском тренутку заклапа сатна казаљка са позитивним смером у осе рачунат је у задатку **Угао сатне казаљке**. Подсетимо се, угао у угаоним минутима који сатна казаљка заузима у односу на положај 12 h (угаони отклон сатне казаљке) једнак је $sat \cdot 30^\circ \cdot \frac{60'}{1^\circ} + minut \cdot 30'$. Заиста на сваки сат казаљка се помери за $\frac{360^\circ}{12} = 30^\circ$, а пошто у једном степену има 60', то је једнако $30^\circ \cdot \frac{60'}{1^\circ}$. На сваки минут времена сатна казаљка се помери додатно за $\frac{30^\circ}{60} = 30'$. Заиста, она се за један минут времена помери 12 пута мање него минутна казаљка, за коју смо установили да се за минут времена помери за 360'. Дакле сатна казаљка се за сваки минут времена помери за 30'.

Да би се израчунао (неоријентисани) угао између казаљки изражен у минутима потребно је одредити апсолутну вредност разлике у угаоним минутима. На крају је добијени резултат потребно превести у степене и минуте.

```
// učitavamo vreme
int sat, minut;
cin >> sat >> minut;
```

```
// sat svodimo na interval [0, 12)
sat %= 12;
```

```
// ugao u minutima koji satna kazaljka zauzima sa položajem 12h
```

```

int ugaoSatne = sat * 30 * 60 + minut * 30;
// ugao u minutima koji minutna kazaljka zauzima sa polozejem 12h
int ugaoMinutne = minut * 360;
// ugao izmedju satne i minutne kazaljki u minutima
int ugaoIzmedju = abs(ugaoSatne - ugaoMinutne);
// ugao izmedju kazaljki u stepenima i minutima
int ugaoIzmedjuStepeni = ugaoIzmedju / 60;
int ugaoIzmedjuMinuti = ugaoIzmedju % 60;

// ispis rezultata
cout << ugaoIzmedjuStepeni << ":" << ugaoIzmedjuMinuti << endl;

```

Задатак можемо решити и дефинисањем помоћних функција за превођење времена у минуте и назад и превођење угла у угаоне минуте и директном применом пропорције унутар програма.

Задатак: Размени цифре

Напиши програм којим се у датом природном броју размењује цифра јединица и цифра стотина. За број са мање од три цифре сматрамо да су недостајуће цифре једнаке 0.

Улаз: У првој линији стандардног улаза налази се природан број мањи од милијарде.

Излаз: На стандардни излаз исписати број добијен после размене цифре јединица и цифре стотина.

Пример

Улаз	Излаз
2349	2943

Задатак: Бројање оваца

Три пастира Јован, Дуле и Станоје броје стадо од највише 120 оваца на крају сваког дана, на прсте десне руке. Броје сва тројица на следећи начин:

- Јован броји једну по једну овцу на прсте десне руке. Сваки пут кад изброји пет оваца он каже Дулету да изброји још један прст на својој десној руци, а Јован крене бројање од почетка.
- Дуле броји један по један прст само кад му Јован то каже. Када Дуле изброји до пет он каже Станоју да изброји још један прст на својој десној руци, а Дуле крене бројање од почетка.
- Станоје броји један по један прст само кад му Дуле то каже.

Када заврше бројање сваки од њих прстима десне руке показује број између 0 и 4. Написати програм који пастирима помаже да на основу броја отворених прстију сва три пастира редом израчунају колико стадо има оваца.

Улаз: На стандардном улази налазе се 3 цела броја из интервала $[0, 4]$ која редом представљају број отворених прстију десне руке наших пастира Јована, Дулета и Станоја.

Излаз: На стандардном излазу у једној линији приказати број оваца у стаду.

Пример

Улаз	Излаз
3	83
1	
3	

Задатак: Обрни цифре

Са стандардног улаза се уноси природан број n . Напиши програм који на стандардни излаз исписује број који се добија када се обрне редослед последње 4 цифре броја n .

Улаз: Позитиван цео број мањи од 10^9 .

Пример 1

Улаз	Излаз
1234567	1237654

Пример 2

Улаз	Излаз
123	3210

1.3. ЦЕЛОБРОЈНО ДЕЉЕЊЕ

Задатак: Цифре слесна

Петар и Марко се играју игре меморије. Петар диктира Марку цифре једног броја, једну по једну, отпозади, кренувши од цифре јединица. Напиши програм који помаже Марку да одреди број чије цифре Петар диктира.

Улаз: Са стандардног улаза уноси се 6 декадних цифара, свака у посебном реду.

Излаз: На стандардни излаз исписати један цео број - број чије цифре су унете са стандардног улаза.

Пример 1

Улаз	Излаз
1	654321
2	
3	
4	
5	
6	

Пример 2

Улаз	Излаз
0	12210
1	
2	
2	
1	
0	

Задатак: Чекање

Два стара другара су договорили да се нађу у центру града. Познато је време (у облику сата, минута и секунде) када је свако од њих дошао на састанак (оба времена су у једном дану). Напиши програм који одређује колико дуго је онај који је први стигао чекао оног другог.

Улаз: Са стандардног улаза се уноси шест целих бројева (сваки је задат у посебном реду). Сат (између 0 и 23), минут (између 0 и 59) и секунд (између 0 и 59) доласка првог човека и сат, минут и секунд доласка другог човека. Подаци представљају исправно задате временске тренутке у оквиру једног дана.

Излаз: На стандардни излаз исписати број сати, минута и секунди колико је онај другар који је први дошао чекао другог. Бројеве исписати у једном реду, раздвојене двочаком.

Пример

Улаз	Излаз
3	3:38:30
45	
20	
7	
23	
50	

Задатак: Тркачи

Два тркача трче по кружној стази дужине s km, један брзином од $v_1 \frac{\text{km}}{\text{h}}$, други брзином од $v_2 \frac{\text{km}}{\text{h}}$. Колико ће времена требати бржем тркачу да за цео круг прстигне споријег?

Улаз: Подаци на две децимале:

- s - дужина кружне стазе ($1 \leq s \leq 10$)
- v_1 - брзина првог тркача ($1 \leq v_1 \leq 45$)
- v_2 - брзина другог тркача ($1 \leq v_2 \leq 45$)

Излаз: Целобројне вредности броја сати, минута и секунди, које представљају време потребно да спорији тркач за цео круг сустигне споријег. Секунде приказати заокружене на најближу целобројну вредност.

Пример

Улаз	Излаз
10.00	10
5.00	0
6.00	0

Задатак: Радијани

У математици се мера угла често изражава у радијанима. Угао чија је мера један радијан је онај угао чија је дужина лука једнака полупречнику. Напиши програм који за дати реалан број радијана израчуна најближи угао у степенима, минутима и секундама (целобројним).

Улаз: Са стандардног улаза уноси се позитиван реалан број радијана.

Изназ: На стандардни излаз исписати три цела броја $stepeni$ ($0 \leq stepeni < 360$), $minuti$ ($0 \leq minuti < 60$) и $sekundi$ ($0 \leq sekundi < 360$), раздвојена двотачкама, који представљају угао у степенима, минутима и секундама најближи датом углу у радијанима.

Пример 1

Улаз Изназ
1.57 89:57:16

Пример 2

Улаз Изназ
1000 55:46:46

Задатак: Гпс

Марија је нашла географске координате на којима се налази закопано благо, међутим њен GPS уређај подржава географске координате унете као децимални број степени, док су координате које она има дате у броју степени, минута и секунди. Напиши програм који ће Марији помоћи да преведе координате.

Улаз: Са стандардног улаза уносе се три цела броја:

- st ($0 \leq st < 360$) - број степени
- min ($0 \leq min < 60$) - број минута
- sek ($0 \leq sek < 60$) - број секунди

Изназ: На стандардни излаз исписати један реалан број који представља угао у степенима (толеранција грешке је 10^{-5}).

Пример

Улаз Изназ
18 18.32222
19
20

Задатак: Поклапање казаљки

Ако сат показује тачно x сати, написати програм који одређује после колико минута ће се први пут поклопити велика и мала казаљка.

Улаз: Број сати од 0 до 11.

Изназ: У минутима заокруженим на најближу целобројну вредност од 0 до 60.

Пример 1

Улаз Изназ
2 11

Пример 2

Улаз Изназ
11 60

Глава 2

Гранање

У овој глави приказаћемо како се могу решавати разни задаци у којима ток извршавања програма зависи од испуњености одређених услова. За такве програме кажемо да имају *разгранату структуру* и да се у њима врши *гранање*.

2.1 Елементи програмског језика

У наставку ћемо описати елементе језика C++ који се користе у програмима разгранате структуре.

2.1.1 Релацијски оператори

Често је потребно утврдити да ли су неке две вредности међусобно једнаке или за неке две вредности утврдити која је од њих већа. За поређење вредности променљивих или израза користе се **релацијски оператори**.

- Основни релацијски оператор је оператор провере једнакости `==`. На пример, ако желимо да испитамо да ли променљиве `a` и `b` имају исту вредност то се може постићи релацијским изразом `a == b`. Вредност овог израза је типа `bool` и најчешће се користи приликом гранања (о коме ће ускоро бити речи), али се, такође, вредност релацијског израза може и доделити променљивој типа `bool`. Дешава се да се приликом писања кода направи грешка и уместо оператора провере једнакости `==` искористи оператор доделе `=`. Напоменимо још и то да поређење два реална броја може произвести понашање које је другачије од очекиваног због непрецизности записа реалних вредности.
- Поред провере једнакости, можемо вршити проверу да ли су две вредности различите. То се постиже оператором `!=`. Дакле, услов да променљиве `a` и `b` имају различиту вредност записује се као `a != b`.
- За поређење да ли је једна вредност мања, мања или једнака, већа, већа или једнака од друге вредности користе се редом релацијски оператори `<`, `<=`, `>`, `>=`.

Очекивано, релацијски оператори су нижег приоритета у односу на аритметичке операторе, односно у изразу `2+3 == 6-1` би се најпре израчунале вредности `2+3` и `6-1` а тек онда би се проверавала једнакост ове две израчунате вредности. Такође оператори `<`, `<=`, `>`, `>=` су вишег приоритета од оператора `==` и `!=`. Сви релацијски оператори су лево асоцијативни.

2.1.2 Логички оператори

За запис сложених услова користимо **логичке операторе**. Логички оператори примењују се на операнде који су типа `bool` и дају резултат типа `bool`. Они су нижег приоритета у односу на релационе и аритметичке операторе.

- Оператор логичке конјункције `&&` користи се за утврђивање да ли истовремено важи неки скуп услова. На пример, вредност израза `2 < 3 && 2 > 1` је `true`, а вредност израза `2 < 3 && 2 < 1` је `false`.
- Други основни логички оператор је оператор логичке дисјункције `||`. Њиме се утврђује да ли је тачан бар један од датих услова. На пример, израз `2 < 3 || 2 < 1` има вредност `true`, а израз `2 > 3 || 2 < 1` вредност `false`.

- Оператор `!` даје логичку негацију. На пример, израз `!(1 < 3)` има вредност `false`, која је супротна од вредности `true` израза `1 < 3`.

Операције логичке конјункције и дисјункције дефинисане су следећим таблицама.

<code>&&</code>	<code>false</code>	<code>true</code>	<code> </code>	<code>false</code>	<code>true</code>	<code>!</code>	
<code>false</code>	<code>false</code>	<code>false</code>	<code>false</code>	<code>false</code>	<code>true</code>	<code>false</code>	<code>true</code>
<code>true</code>	<code>false</code>	<code>true</code>	<code>true</code>	<code>true</code>	<code>true</code>	<code>true</code>	<code>false</code>

Оператор логичке конјункције је вишег приоритета од оператора логичке дисјункције. Дакле у изразу `a || b && c` би се прво израчунала вредност израза `b && c`, а онда би се извршила операција логичке дисјункције променљиве `a` и вредности претходног израза. Оба бинарна логичка оператора су лево асоцијативна.

И за оператор конјункције и за оператор дисјункције карактеристично је **лењо израчунавање** – иако су поменути оператори бинарни, вредност другог операнда се не рачуна, уколико је вредност комплетног израза одређена вредношћу првог операнда. Дакле, приликом израчунавања вредности израза `A && B`, уколико је вредност израза `A` једнака `false`, не израчунава се вредност израза `B`; слично, приликом израчунавања вредности израза `A || B`, уколико је вредност израза `A` једнака `true`, не израчунава се вредност израза `B`.

2.1.3 Наредба `if`

Већина програмских језика, па и `C++`, располаже наредбом гранања. Циљ гранања јесте да се на основу испуњености (или неиспуњености) неког услова одреди коју наредну наредбу треба извршити (одатле потиче и назив гранање).

Основни облик наредбе гранања у језику `C++` је:

```
if (uslov)
    naredba1
else
    naredba2
```

У овом случају, ако је испуњен услов `uslov` биће извршена прва наредба, а ако услов није испуњен биће извршена друга наредба. Наравно, уместо појединачне наредбе, може се јавити и блок наредби наведен у витичастим заградама. На пример, исписивање да ли је дати број паран или непаран може имати следећи облик:

```
if (broj % 2 == 0)
    cout << "paran" << endl;
else
    cout << "neparan" << endl;
```

Ставка `else` није обавезан део наредбе гранања. Дакле, ако бисмо хтели да испишемо да је број паран ако јесте паран, а ако није да не исписујемо ништа, то бисмо могли да постигнемо наредном наредбом:

```
if (broj % 2 == 0)
    cout << "paran" << endl;
```

2.1.4 Условни израз

Уместо наредбе гранања некада је погодније искористити условни израз (израз гранања).

Условни израз, односно оператор `?:`, има следећу форму:

```
uslov ? rezultat_tacno : rezultat_netacno
```

Овај оператор је тернарни, односно има три аргумента: први је услов чију испуњеност проверавамо, други аргумент је вредност израза ако је услов испуњен, док се трећим аргументом задаје вредност израза ако услов није испуњен. На пример, исписивање парности задатог броја могло би да се реализује и коришћењем израза гранања:

```
cout << (broj % 2 == 0 ? "paran" : "neparan") << endl;
```

2.2. ЈЕДНОСТАВНО ГРАНАЊЕ

Оператор гранања је десно асоцијативан и нижег приоритета у односу на скоро све остале оперatore (приоритетнији је једино од оператора доделе).

2.2 Једноставно гранање

Задаци који следе у наставку захтевају гранање на основу појединачних услова тј. до решења се долази ако се испита да ли неки услов важи или не важи.

2.2.1 Релацијски оператори

За решавање наредне групе задатака довољно је формулисати и испитати просте услове коришћењем аритметичких и релацијских оператора.

Задатак: Јабукe

Пера и Мика су брали јабуке. Пера је убрао p , а Мика m јабука. Напиши програм који проверава да ли је Пера успео да набере више јабука него Мика.

Улаз: Са стандардног улаза уносе се два природна броја (сваки у посебном реду). У првом реду број јабука које је убрао Пера, а у другом број јабука које је убрао Мика.

Излаз: Исписати на стандардном излазу: DA - ако је Пера убрао више јабука од Мике, а NE - ако Пера није убрао више јабука од Мике.

Пример 1		Пример 2		Пример 3	
Улаз	Излаз	Улаз	Излаз	Улаз	Излаз
73	DA	48	NE	35	NE
10		48		56	

Решење

Задатак решавамо тако што проверимо да ли је број јабука које је убрао Пера већи од броја јабука који је убрао Мика. Ако јесте испишемо DA, а у супротном испишемо NE.

Један начин да се реализује гранање у програму је да се употреби наредба if-else.

```
int p, m; // број Perinih i Mikinih jabuka
cin >> p >> m;
if (p > m)
    cout << "DA" << endl;
else
    cout << "NE" << endl;
```

Гранање се у овом случају може реализовати и помоћу условног израза. Условни израз се реализује применом оператора ?:

```
int p, m; // број Perinih i Mikinih jabuka
cin >> p >> m;
cout << (p > m ? "DA" : "NE") << endl;
```

Задатак: Збир година браће и сестре

Пера, Мика и Лаза су три брата рођена у истом дану, а Ана је њихова 3 године старија сестра. Написати програм којим се проверава да ли унети број може бити збир њихових година.

Улаз: Са стандардног улаза уноси се један позитиван природан број мањи од 500.

Излаз: На стандардном излазу приказати реч да ако унети број може бити збир година Пера, Мике, Лазе и Ане, а ако не може приказати реч не.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
27	da	30	ne

Решење

Пера, Мика и Лаза су рођени исте године, па имају једнак број година. Обележимо са x број година сваког од браће. Ана је за 3 године старија од своје браће па је њен број година $x + 3$. Према томе збир њихових година је $3 \cdot x + (x + 3) = 4 \cdot x + 3$. Потребно је проверити да ли унети број n може бити збир њихових година тј. потребно је проверити да ли за неки ненегативан цео број x (број година је ненегативан цео број) важи једнакост $4 \cdot x + 3 = n$. Решење једначине је $x = \frac{n-3}{4}$, то је ненегативан цео број ако је $n - 3$ дељиво са 4 (није потребно проверавати да ли је $n - 3 \geq 0$ јер за природан број n ако је $n - 3 < 0$ онда $n - 3$ није дељиво са 4).

Проверу дељивости можемо извршити тако што израчунамо целобројни остатак при дељењу (оператором %) и проверимо да ли је једнак нули.

Гранање можемо извршити наредбом гранања `if-else`.

```
int n;    // zbir godina tri brata i sestre
cin >> n;
if ((n - 3) % 4 == 0)
    cout << "da" << endl;
else
    cout << "ne" << endl;
```

Гранање се у овом случају може реализовати и помоћу условног израза. Условни израз се реализује применом оператора `?:`.

```
int n;    // zbir godina tri brata i sestre
cin >> n;
cout << ((n - 3) % 4 == 0 ? "da" : "ne") << endl;
```

Задатак: Теме правоугаоника

Дате су координате три темена правоугаоника са целобројним координатама чије су странице паралелне координатним осама. Темена су дата у произвољном редоследу. Написати програм којим се одређују координате четвртог темена.

Улаз: Са стандардног улаза учитавају се целобројне координате три темена правоугаоника - у сваком реду по два податка (координата x и координата y), раздвојена размаком.

Излаз: x и y координата траженог темена раздвојене размаком.

Пример

Улаз	Излаз
3 5	3 9
7 5	
7 9	

Решење

Уочимо да када су странице правоугаоника паралелне координатним осама, његов леви пар темена има исту апсцису (x -координату) и десни пар темена има исту апсцису. Аналогно важи да горњи пар темена правоугаоника има исту ординату (y -координату) и доњи пар темена има исту ординату. Према томе, када провером једнакости установимо да два темена имају исту x -координату, тражено теме припада другом пару са истом x -координатом, па исписујемо x -координату темена које није учествовало у провери. На исти начин долазимо до непознате y -координате.

```
// koordinate tri poznata temena
int x1, y1, x2, y2, x3, y3;
cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;

// koordinate nepoznatog temena
int x4 = 0, y4 = 0;

// odredjujemo nepoznatu apscisu x4
if (x1 == x2) x4 = x3;
if (x1 == x3) x4 = x2;
if (x2 == x3) x4 = x1;
```

2.2. ЈЕДНОСТАВНО ГРАНАЊЕ

```
// odredjujemo nepoznatu ordinatu y4
if (y1 == y2) y4 = y3;
if (y1 == y3) y4 = y2;
if (y2 == y3) y4 = y1;

// ispisujemo trazene koordinate
cout << x4 << " " << y4 << endl;
```

Задатак: Једнакокрајни троугао датог обима

Дате су целобројне дужине трију страница троугла. Написати програм којим се проверава да ли постоји једнакокрајни троугао целобројне дужине страница истог обима као дати троугао и која му је дужина странице.

Улаз: На стандарном улазу налазе се 3 природна броја која представљају странице троугла.

Изназ: Једна линија стандарног излаза садржи реч *да* ако постоји тражени троугао, за којом одвојено празнином, следи дужина странице тог троугла, ако тражени троугао не постоји садржи реч *не*.

Пример 1		Пример 2	
Улаз	Изназ	Улаз	Изназ
3	да 4	4	не
4		6	
5		9	

2.2.2 Логички оператори

За решавање наредне групе задатака пожељно је искомбиновати одређен број услова коришћењем логичких оператора.

Задатак: Радно време

Радно време једне организације је између 9 и 17 часова. Одредити да ли је послати мејл стигао у току радног времена.

Улаз: Са стандардног улаза се уносе два цела броја, сваки у посебном реду, који представљају сат и минут приспећа мејла.

Изназ: На стандардни излаз исписати *да* ако је мејл стигао у току радног времена тј. *не* ако није (9:00 спада у радно време, док 17:00 не спада).

Пример 1		Пример 2	
Улаз	Изназ	Улаз	Изназ
14	да	17	не
53		01	

Решење

Мејл је стигао током радног времена ако и само ако је сат у интервалу $[9, 17)$ тј. ако је већи или једнак 9 и строго мањи од 17 (број минута је небитан).

Иако математичка нотација допушта да се запише услов $a \leq c < b$, у програмским језицима такав запис најчешће није исправан. То је случај и у језику C++ и тај услов треба записати као $a \leq c \ \&\& \ c < b$. Додатно, треба бити веома обазрив јер је израз $a \leq c < b$ синтаксички исправан, али даје погрешну вредност (овде се истинитосна вредност израза $a \leq c$ тумачи као бројевна вредност 0 или 1 и она се онда пореди са бројевном вредношћу b).

Наравно, слично бисмо поступили и у случају потпуно отворених или потпуно затворених интервала, једино треба правилно одабрати одговарајући релацијски оператор ($<$ или \leq).

```
int sat, minut;
cin >> sat >> minut;
if (9 <= sat && sat < 17)
    cout << "да" << endl;
```

```
else
```

```
    cout << "ne" << endl;
```

Задатак: Кућни ред

Кућни ред забрањује прављење буке пре 6 часова, између 13 и 17 часова и након 22 часа. Напиши програм који радницима говори да ли у неком датом тренутку могу да изводе бучније радове.

Улаз: Са стандардног улаза се уноси цео број између 0 и 23 који представља сат.

Излаз: На стандардни излаз исписати поруку `moze` ако је дозвољено изводити бучне радове тј. `ne moze` ако није.

Пример 1

<i>Улаз</i>	<i>Излаз</i>
5	ne moze

Пример 2

<i>Улаз</i>	<i>Излаз</i>
6	moze

Пример 3

<i>Улаз</i>	<i>Излаз</i>
13	ne moze

Решење

Једно решење можемо засновати на томе да су радови дозвољени ако и само ако сат припада интервалу $[6, 13)$ или $[17, 22)$. Припадност интервалу смо већ тестирали у задатку **Радно време**.

```
int sat;
cin >> sat;
if ((6 <= sat && sat < 13) ||
    (17 <= sat && sat < 22))
    cout << "moze" << endl;
else
    cout << "ne moze" << endl;
```

Друго решење можемо засновати на томе да радови нису дозвољени ако и само ако је сат мањи од шест, ако припада интервалу $[13, 17)$ или је већи или једнак 22.

```
int sat;
cin >> sat;
if (sat < 6 || (13 <= sat && sat < 17) || (sat >= 22))
    cout << "ne moze" << endl;
else
    cout << "moze" << endl;
```

Задатак: Постоји ли троугао датих дужина страница

Написати програм којим се проверава да ли постоји троугао са датим дужинама страница.

Улаз: На стандардном улазу налазе се три реална позитивна броја, сваки у посебној линији. Бројеви представљају дужине страница a , b , c .

Излаз: Једна линија стандардног излаза која садржи реч `da` ако постоји троугао, иначе садржи реч `ne`.

Пример

<i>Улаз</i>	<i>Излаз</i>
4.3	da
5.4	
6.7	

Решење

Познато је да је дужина сваке странице троугла мања од збира дужина друге две странице (ова особина се назива *неједнакост троугла*). Да би троугао са дужинама страница a , b и c постојао, довољно је да важи $a < b + c$, $b < a + c$ и $c < a + b$. Ако важе сва три услова постоји троугао са датим страницама, иначе не постоји такав троугао. Бројеви a , b и c су по претпоставци задатка позитивни, тако да тај услов није неопходно посебно проверавати.

Напоменимо и да се неједнакост троугла некада помиње и у облику у којем се тражи да је разлика дужина сваке две странице мања од дужине треће странице, но, тај услов није потребно посебно проверавати јер он следи из услова за збир страница (на пример, ако се покаже да је $a < b + c$, тада важи и да је $a - b < c$ и да је $a - c < b$).

2.2. ЈЕДНОСТАВНО ГРАНАЊЕ

Пошто сва три услова треба да важе, могуће је повезати их оператором логичке конјункције (оператором и тј. `&&`).

```
// proveravamo da li postoji trougao sa duzinama stranica a, b i c
if (a < b + c && b < a + c && c < a + b)
    cout << "da" << endl;
else
    cout << "ne" << endl;
```

Вежбе ради, проверу можемо издвојити и у засебну функцију (коју онда можемо употребити и у наредним програмима).

```
// proverava da li postoji trougao cije su stranice duzine a, b i c
bool postojiTrougao(double a, double b, double c) {
    return a + b > c && b + c > a && a + c > b;
}

int main() {
    // učitavamo duzine stranica
    double a, b, c;
    cin >> a >> b >> c;

    // proveravamo da li postoji trougao sa duzinama stranica a, b i c
    if (postojiTrougao(a, b, c))
        cout << "da" << endl;
    else
        cout << "ne" << endl;
    return 0;
}
```

Види групачија решења овој задатку.

Задатак: Преступна година

Напиши програм који проверава да ли је унета година преступна.

Улаз: Са стандардног улаза се уноси број године између 1900 и 2200.

Излаз: На стандардни излаз исписати да ако је година преступна тј. не ако није.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
2004	da	2017	ne

Решење

По грегоријанском календару, који је у званичној употреби, година је преступна ако је дељива са 4, а није дељива са 100 или је дељива са 400. Тиме се постиже да је на 400 година тачно 97 преступних и да је трајање једне године 365,2425 дана, што је веома блиску трајању астрономске године. Проверу дељивости вршимо израчунавајући остатак при дељењу и провером да ли је он једнак нули (као у задатку **Збир година браће и сестре**).

```
int godina;
cin >> godina;
if ((godina % 4 == 0 && godina % 100 != 0) || godina % 400 == 0)
    cout << "da" << endl;
else
    cout << "ne" << endl;
```

Проверу можемо издвојити и у засебну функцију, коју онда можемо употребљавати и у наредним програмима.

```
bool prestupna(int godina) {
    return (godina % 4 == 0 && godina % 100 != 0) || godina % 400 == 0;
}
```

```
int main() {
    int godina;
    cin >> godina;
    if (prestupna(godina))
        cout << "da" << endl;
    else
        cout << "ne" << endl;
    return 0;
}
```

Задатак: Два броја истог знака

Написати програм којим се проверава да ли су два цела броја истог знака.

Улаз: На стандардном улазу налазе се два цела броја a, b ($-10^4 \leq a, b \leq 10^4$), оба различита од нуле.

Издаз: Једна линија стандарног излаза која садржи реч *да* ако су дати бројеви истог знака, иначе садржи реч *не*.

Пример

Улаз	Издаз
234	ne
-34	

Решење

Бројеви a и b истог су знака ако су оба позитивна ($a > 0$ и $b > 0$) или оба негативна ($a < 0$ и $b < 0$), па задатак можемо решити постављањем тих услова. Подсетимо се да у језику C++ везник *и* можемо записати оператором логичке конјункције `&&`, а везник *или* можемо записати оператором логичке дисјункције `||`.

```
if ((a > 0 && b > 0) || (a < 0 && b < 0))
    cout << "da" << endl;
else
    cout << "ne" << endl;
```

Још елегантније решење је да се услов да су два броја истог знака исказе као $a > 0 \Leftrightarrow b > 0$ тј. да је број a позитиван ако и само ако је број b позитиван (то је оправдано јер бројеви нису једнаки нули). У програмским језицима не постоји посебан оператор еквиваленције, међутим, еквиваленција две логичке вредности се може заправо испитати њиховим поређењем оператором `==`. Једнакост два логичка израза ће бити тачна ако су оба израза тачна или ако су оба израза нетачна, што је управо исто оно што је семантика њихове еквиваленције.

```
if ((a > 0) == (b > 0))
    cout << "da" << endl;
else
    cout << "ne" << endl;
```

Један трик да лако испитамо да ли су два броја истог знака је да приметимо то да су бројеви, различити од нуле, истог знака ако и само ако је њихов производ већи од 0. Према томе довољно је проверити да ли је $a \cdot b > 0$. При томе морамо бити обазриви да ли производ бројева не доводи до прекорачења. У нашем задатку до прекорачења неће доћи јер су бројеви a, b из интервала $[-10^4, 10^4]$ па је њихов производ у интервалу $[-10^8, 10^8]$ што може да се региструје коришћењем целобројног типа податка (опсеге различитих типова и питање прекорачења разматраћемо детаљније у наредним деловима збирке).

```
if (a * b > 0)
    cout << "da" << endl;
else
    cout << "ne" << endl;
```

Види грујачија решења овој задатку.

Задатак: Исти квадрант

Написати програм којим се проверава да ли две тачке $A(x_1, y_1), B(x_2, y_2)$ припадају истом квадранту. Сматраћемо да тачке на позитивном делу x осе припадају првом и четвртном квадранту, тачке на негативном делу

2.2. ЈЕДНОСТАВНО ГРАНАЊЕ

x осе припадају другом и трећем квадранту, слично тачке на позитивном делу y осе припадају првом и другом квадранту, а на негативном делу y осе трећем и четвртном квадранту, а да координатни почетак припада свим квадрантима.

Улаз: Стандардни улаз садржи четири цела броја, сваки у посебној линији:

- x_1, y_1 ($-10^4 \leq x_1, y_1, \leq 10^4$) - координате тачке $A(x_1, y_1)$
- x_2, y_2 ($-10^4 \leq x_2, y_2, \leq 10^4$) - координате тачке $B(x_2, y_2)$

Израз: На стандардном излазу у једној линији приказати реч да ако тачке припадају истом квадранту у супрорном приказати реч не.

Пример

Улаз	Израз
12	не
-45	
15	
23	

Задатак: Тачка у правоугаонику и кругу

Напиши програм који за тачку у равни задату својим координатама испитује да ли припада задатом кругу и задатом правоугаонику чије су странице паралелне са координатним осама.

Улаз: Са стандардног улаза учитавају се следећи реални бројеви (бројеви у истом реду су раздвојени једним размаком):

- x, y - координате тачке,
- x_0, y_0 - координате заједничког центра круга и правоугаоника,
- r - полупречник круга,
- w, h - дужина и ширина страница правоугаоника.

Израз: На стандардни излаз исписати два линије текста. У првој линији треба да пише `jeste u krugu` ако тачка (x, y) припада кругу са центром (x_0, y_0) полупречника r односно `nije u krugu` ако тачка не припада кругу. У другој линији треба да пише `jeste u pravougaoniku` ако тачка (x, y) припада правоугаонику чији је центар (тежиште) у тачки (x_0, y_0) , чије су странице паралелне координатним осама и чија је дужина w тј. h , односно `nije u pravougaoniku` ако тачка не припада унутрашњости тог правоугаоника. Граница круга (кружница) и правоугаоника сматрају се њиховим делом.

Пример

Улаз	Израз
1 1	nije u krugu
0 0	jeste u pravougaoniku
1	
2 2	

Задатак: Статус објављен током школе

Милица је објавила три статуса на друштвеној мрежи. Напиши програм који проверава да ли је неки од тих статуса написан у школи (Милица је била у школи од 8:30 до 13:50, укључујући и та времена).

Улаз: Са стандардног улаза учитавају се три времена објаве статуса (свако време у посебном реду). За сваки статус се учитавају сати и минути (два цела броја раздвојена размаком).

Израз: На стандардни излаз написати да ако је неки статус објављен током боравка у школи тј. не у супротном.

Пример 1

Улаз	Израз
14 23	не
17 19	
22 14	

Пример 2

Улаз	Израз
7 23	да
8 45	
15 20	

Задатак: Да ли се две даме нападају

Напиши програм који проверава да ли се две даме (краљице) на шаховској табли међусобно нападају (краљице се нападају ако се налазе у истој врсти, истој колони или на истој дијагонали шаховске табле).

Улаз: Са стандардног улаза се читавају координате поља на којем се налази једна краљица (два броја између 0 и 7 раздвојена размаком) и у наредном реду координате поља на којем се налази друга краљица (поново два броја између 0 и 7 раздвојена размаком). Претпостављамо да се краљице налазе на различитим пољима.

Излаз: На стандардни излаз исписати текст `da` ако се краљице нападају тј. `ne` у супротном.

Пример 1		Пример 2		Пример 3	
Улаз	Излаз	Улаз	Излаз	Улаз	Излаз
5 3	da	5 3	ne	4 4	da
1 7		1 8		4 6	

2.3 Угнежђено гранање

У задацима који следе користи се угнежђено гранање (наредбе гранања су наведене унутар тела ширих наредби гранања).

2.3.1 Елементи програмског језика

Уколико је потребно извршити гранање на основу неколико међусобно искључујућих услова, то можемо да урадимо коришћењем *конструкције* `else-if`, која има следећи општи облик:

```
if (uslov1)
    naredba1
else if (uslov2)
    naredba2
else if (uslov3)
    naredba3
...
else
    naredbak
```

Приликом извршавања ове наредбе проверавају се редом услови `uslov1`, `uslov2` итд. све док се не наиђе на неки услов који је испуњен и чији се блок наредби извршава. Последња `else` ставка се извршава ако ниједан од услова није испуњен. Тај део је опцион и не мора се навести.

Напоменимо да би на основу уобичајеног назубљивања наредби `if-else` коџ требало да буде назубљен на следећи начин:

```
if uslov1
    naredba1
else
    if uslov2
        naredba2
    else
        if ...
            ...
        else
            naredbak
```

зато што је доња наредба `if-else` угнежђена у горњу `if-else` наредбу и део је њене `else` гране. Ипак, `else-if` конструкција се користи прилично често и назубљује на горе приказани начин да би се нагласило да су све три гране у њој равноправне.

Поменимо једну важну појаву познату као *вучеће else* (енг. `dangling else`). Уколико имамо угнежђено гранање облика:

```
if (uslov1)
    if (uslov2)
        naredba1
```

2.3. УГНЕЖЂЕНО ГРАНАЊЕ

```
else
    naredba2
```

поставља се питање за коју `if` наредбу се везује `else` ставка: за прву или другу? С обзиром да преводилац не може да погоди шта је програмер имао на уму, ово питање се регулише правилом које гласи: `else` ставка се везује за последњу (дакле најближу) `if` наредбу, те би у претходном случају исправно форматирање било:

```
if (uslov1)
    if (uslov2)
        naredba1
    else
        naredba2
```

Уколико желимо да постигнемо да се `else` веже за прву `if` наредбу, то можемо постићи увођењем витичастих заграда и одговарајућим груписањем:

```
if (uslov1) {
    if (uslov2)
        naredba1
}
else
    naredba2
```

Уколико је потребно извршити гранање на основу неких конкретних вредности можемо искористити наредбу `switch-case` која има наредни облик:

```
switch (izraz) {
    case vrednost_1:
        naredbe_1;
        break;
    case vrednost_2:
        naredbe_2;
        break;
    ...
    case vrednost_k;
        naredbe_k;
        break;
    default:
        naredbe_d;
}
```

Приликом извршавања ове наредбе, редом се проверава да ли је вредност израза `izraz` једнака некој од наведених вредности. Уколико израз има неку од наведених вредности, извршава се наредба (или блок наредби) наведен након те вредности и након тога се излази из наредбе `switch-case`. Ако бисмо заборавили да наведемо ставку `break` онда би се извршиле све наредбе које одговарају случају за дату вредност израза и након тога све наредне наредбе (за вредности наведене након те у наредби `switch-case`). Последња ставка `default` је опциона и означава случај када израз није једнак ниједној од наведених вредности.

Ако је блок наредби које треба извршити исти за неке различите вредности које су наведене унутар наредбе `switch-case`, онда се оне могу груписати на следећи начин:

```
switch (izraz) {
    case vrednost_1: vrednost_2: vrednost_3:
        naredbe_1;
        break;
    case vrednost_4: vrednost_5:
        naredbe_2;
        break;
    default:
        naredbe_d;
}
```


Израз чија се вредност тестира може бити целобројног типа, али и типа `bool`, типа `string` и набројивог типа `enum` (о тим типовима биће више речи касније).

2.3.2 Гранање на основу припадности интервалима

У наредним задацима потребно је донети одређену одлуку на основу тога којем од неколико надовезаних интервала бројевне праве припада унета вредност. На пример, агрегатно стање воде одређујемо у зависности од тога да ли температура припада интервалу $[-\infty, 0)$, $[0, 100]$ или $(100, +\infty)$, а успех ученика у зависности од тога да ли просек припада интервалу $[2.0, 2.5)$, $[2.5, 3.5)$, $[3.5, 4.5)$ или $[4.5, 5]$.

Иако је могуће направити серију независних гранања којима се за сваки интервал проверава да ли му вредност припада, ови задаци се најчешће решавају применом конструкције `else-if` (тима се смањује број поређења и могућност грешке).

Задатак: Агрегатно стање

Написати програм којим се на основу температуре воде одређује њено агрегатно стање. Ако је температура:

- виша од $0^\circ C$ и нижа од $100^\circ C$ - агрегатно стање је течно
- не виша од $0^\circ C$ - агрегатно стање је чврсто,
- не нижа од $100^\circ C$ - агрегатно стање је гасовито.

За температуру од тачно 0° сматра се да је агрегатно стање чврсто, а за тачно 100° да је гасовито.

Улаз: Температура - цео број од -100 до 200 .

Излаз: На стандардни излаз исписати једну од следећих речи: `cvrsto`, `tecno`, `gasovito`.

Пример 1

Улаз Излаз
-10 cvrsto

Пример 2

Улаз Излаз
99 tecno

Решење

Из формулације задатка произилази да имамо три агрегатна стања, која настају у следећим температурним интервалима:

- ако температура није виша од $0^\circ C$ - агрегатно стање је чврсто;
- ако је температура виша од $0^\circ C$ и нижа од $100^\circ C$ - агрегатно стање је течно;
- ако температура није нижа од $100^\circ C$ - агрегатно стање је гасовито.

Одавде произилази код са три међусобно независна услова, којима се у произвољном редоследу проверава припадност температуре једном од три интервала $(-\infty, 0]$, $(0, 100)$ и $[100, \infty)$.

```
int t; // Temperatura
cin >> t;
```

```
if (t <= 0)
    cout << "cvrsto" << endl;
if (t > 0 && t < 100)
    cout << "tecno" << endl;
if (t >= 100)
    cout << "gasovito" << endl;
```

Међутим, до решења се може доћи и уз коришћење такозване *конструкције* `else-if`, следећим поступком:

- ако температура “није виша” од $0^\circ C$ - агрегатно стање је чврсто;
- у противном (температура је виша од $0^\circ C$): ако је температура нижа од $100^\circ C$ (припада другом интервалу) - агрегатно стање је течно;
- у противном (температура је виша или једнака $100^\circ C$): агрегатно стање је гасовито.

```
int t; // Temperatura
cin >> t;
```

```
if (t <= 0)
    cout << "cvrsto" << endl;
```

2.3. УГНЕЖЂЕНО ГРАНАЊЕ

```
else if (t < 100)
    cout << "tecno" << endl;
else
    cout << "gasovito" << endl;
```

Аналогно, претходном решењу можемо проверавати припадност температуре интервалу, али идући слесна од интервала $[100, \infty)$.

```
int t; // Temperatura
cin >> t;

if (t >= 100)
    cout << "gasovito" << endl;
else if (t > 0)
    cout << "tecno" << endl;
else
    cout << "cvrsto" << endl;
```

Задатак: Успех ученика

Написати програм којим се на основу датог просека ученика приказује успех ученика. Одличан успех има ученик чији је просек већи или једнак 4,5. Врлодобар успех постиже ученик чији је просек већи или једнак 3,5, а мањи од 4,5, добар успех се постиже за просек који је већи или једнак 2,5 а мањи од 3,5, довољан успех за просек већи или једнак 2, а мањи од 2,5. Ако ученик има неку јединицу унеће се просек 1, а успех му је недовољан.

Улаз: Са стандардног улаза учитава се један реалан број из скупа $[2, 5] \cup \{1\}$ који представља просек ученика.

Излаз: На стандардни излаз приказати успех ученика (реч *odlican*, *vrلودobar*, *dobar*, *dovoljan* или *nedovoljan*).

Пример

Улаз	Излаз
3.75	vrلودobar

Решење

Гранање на основу припадности интервалима

Одређивање успеха врши се на основу припадности датог просека интервалима тако да се овај задатак може решити на сличан начин као и задатак [Агрегатно стање](#).

Један начин за решавање задатка је да за сваки успех, независно један од другог, проверимо да ли просек припада одговарајућем интервалу. Проверу да ли ученик има одличан успех вршимо утврђујући да ли је просек већи или једнак 4,5, за врло добар успех проверавамо да ли је просек већи или једнак 3,5 а мањи од 4,5, и слично за остале успехе.

```
double prosek;
cin >> prosek;
if (prosek >= 4.5)
    cout << "odlican" << endl;
if (prosek >= 3.5 && prosek < 4.5)
    cout << "vrلودobar" << endl;
if (prosek >= 2.5 && prosek < 3.5)
    cout << "dobar" << endl;
if (prosek >= 2 && prosek < 2.5)
    cout << "dovoljan" << endl;
if (prosek == 1)
    cout << "nedovoljan" << endl;
```

Можемо приметити неке недостатке таквог решења:

- када утврдимо да ученик има један успех, нема потребе да проверавамо да ли ученик постиже неки други успех;

- када утврдимо да ученик не постиже успех који смо проверавали онда то можемо искористити за проверу следећег успеха.

Наведене недостатке превазилазимо тако што провере не вршимо независно једну од друге. Проверавамо редом успехе од одличног до недовољног (можемо и од недовољног до одличног) и при томе у следећој провери увек користимо оно шта смо закључили у претходној, коришћењем конструкције `else-if`. Прво проверимо да ли је ученик постигао одличан успех тј. да ли му је просек већи или једнак 4,5, ако јесте прикажемо одговарајућу поруку, а ако није настављамо даљу проверу. Сада знамо да је просек ученика мањи од 4,5 па при провери да ли је ученик постигао врло добар успех тај услов не проверавамо већ само да ли је просек већи или једнак 3,5. На исти начин настављамо по потреби провере за добар и довољан успех, и на крају ако ниједан услов није испуњен ученик има недовољан успех.

```
double prosek; // prosek ocena ucenika
cin >> prosek;
if (prosek >= 4.5)
    cout << "odlican" << endl;
else if (prosek >= 3.5)
    cout << "vrlodobar" << endl;
else if (prosek >= 2.5)
    cout << "dobar" << endl;
else if (prosek >= 2)
    cout << "dovoljan" << endl;
else
    cout << "nedovoljan" << endl;
```

Види групачија решења овој задатку.

Задатак: Одмор на пола пута

Путник се кретао T_1 сати брзином $V_1 \frac{\text{km}}{\text{h}}$, а затим T_2 сати брзином $V_2 \frac{\text{km}}{\text{h}}$ и T_3 сати брзином $V_3 \frac{\text{km}}{\text{h}}$. Планирао је да се одмори на пола пута, израчунати после колико времена од почетка путовања је имао одмор.

Улаз: Са стандардног улаза учитава се 6 реалних бројева, сваки у засебном реду: $T_1, V_1, T_2, V_2, T_3, V_3$.

Изназ: На стандардни излаз исписати један реалан број који представља време потребно да путник стигне до пола пута. Допуштена грешка у израчунавању и приказу резултата је 10^{-3} .

Пример

Улаз	Изназ
1 10	3.667
2 20	
3 30	

Решење

Пошто се на свакој деоници путник кретао равномерно, пређени пут на свакој деоници може се израчунати на основу формула $s_1 = v_1 \cdot t_1$, $s_2 = v_2 \cdot t_2$, $s_3 = v_3 \cdot t_3$ (види задатак [Путовање](#)). Зато је половина пређеног пута на растојању $s_{pola} = (s_1 + s_2 + s_3)/2$. Да бисмо израчунали после колико времена је дошао до те половине, анализираћемо да ли та средина пута лежи на првој, другој или трећој деоници. Ако је на првој (ако је $s_{pola} \leq s_1$), тада се време може израчунати као s_{pola}/v_1 . Ако је на другој (ако је $s_1 < s_{pola} \leq s_1 + s_2$), тада се време може израчунати као $t_1 + (s_{pola} - s_1)/v_2$ (прву деоницу је прешао за време t_1 , а онда је у другој прешао пут $s_{pola} - s_1$ крећући се брзином v_2). На крају, ако је на трећој (ако је $s_1 + s_2 < s_{pola} \leq s_1 + s_2 + s_3$), тада се време може израчунати као $t_1 + t_2 + (s_{pola} - (s_1 + s_2))/v_3$ (прву и другу деоницу је прешао за време $t_1 + t_2$, а онда је у трећој деоници прешао пут $s_{pola} - (s_1 + s_2)$ крећући се брзином v_3). Одређивање на којој деоници је средина пута врши се одређивањем интервала ком израчуната вредност припада, што се може урадити конструкцијом `else-if` како је приказано у задатку [Агрегатно стање](#).

```
// izracunavamo duzinu svake deonice
double s1 = t1*v1;
double s2 = t2*v2;
double s3 = t3*v3;
// izracunavamo rastojanje do polovine puta
double sPola = (s1+s2+s3) / 2.0;
```

2.3. УГНЕЖЂЕНО ГРАНАЊЕ

```
// izracunavamo i ispisujemo vreme za koje se stiglo do pola puta
double tDoPola;
if (sPola <= s1)
    tDoPola = sPola / v1;
else if (sPola <= s1+s2)
    tDoPola = t1 + (sPola - s1) / v2;
else
    tDoPola = t1 + t2 + (sPola - (s1+s2)) / v3;

cout << fixed << showpoint << setprecision(3)
    << tDoPola << endl;
```

Задатак: Растојање од тачке до дужи

Одредити растојање дате тачке A од дате дужи A_0A_1 која лежи на x -оси (растојање тачке и дужи дефинише се као најкраће међу растојањима између те тачке и тачака те дужи).

Улаз: Са стандардног улаза уносе се 4 реална броја (сваки у посебном реду):

- x_0 ($-10 \leq x_0 < 10$) – x -координата тачке $A_0(x_0, 0)$
- x_1 ($x_0 < x_1 \leq 10$) – x -координата тачке $A_1(x_1, 0)$
- x, y ($-20 \leq x, y, \leq 20$) – координате тачке $A(x, y)$

Издаз: На стандардни издаз исписати један реалан број - најкраће растојање од тачке A до дужи A_0A_1 , заокружено на пет децимала.

Пример 1

Улаз	Издаз
0	1.41421
1	
-1	
1	

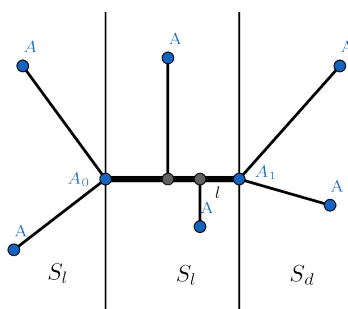
Пример 2

Улаз	Издаз
0	3.00000
1	
0.5	
3	

Решење

Посматрајмо дуж A_0A_1 за тачке $A_0(x_0, 0)$ и $A_1(x_1, 0)$ и поделу равни на три дисјунктна подскупа:

$$S_l = \{(x, y) \mid x < x_0\}, \quad S_s = \{(x, y) \mid x_0 \leq x \leq x_1\}, \quad S_d = \{(x, y) \mid x > x_1\}.$$



Слика 2.1: Растојање од тачке до дужи

Свим тачкама $A \in S_l$ најближа тачка дужи A_0A_1 је тачка A_0 . Докажимо ово тако што ћемо претпоставити супротно, тј. претпоставити да је нека тачка $A' \in A_0A_1$ ближа тачки A него тачка A_0 . У троуглу AA_0A' страница AA' која лежи наспрам тупог угла $\angle AA_0A'$ већа је од странице AA_0 која лежи наспрам оштрог угла $\angle AA'A_0$, што је контрадикција у односу на нашу претпоставку.

Аналогно, свим тачкама $A \in S_d$ најближа је тачка A_1 .

Тачкама $A \in S_s$ најближа је њихова нормална пројекција A_n на дуж A_0A_1 . Докажимо то тако што ћемо претпоставити супротно, да постоји нека тачка $A' \in A_0A_1$ која је ближа тачки A него тачка A_n . У троуглу

AA_nA' дуж AA' је хипотенуза и лежи наспрам правог угла, па мора бити дужа од катете AA_n која лежи наспрам оштрог угла, што је контрадикција.

Ако тачка A има координате (x, y) , тада је њено растојање од тачке A_0 једнако $\sqrt{(x - x_0)^2 + y^2}$, а од тачке A_1 једнако је $\sqrt{(x - x_1)^2 + y^2}$ (ово следи на основу Питагорине теореме, како смо то већ показали у задатку **Растојање тачака**). Пројекција A_n има координате $(x, 0)$ тако да је њено растојање од тачке A једнако $\sqrt{(x - x)^2 + (y - 0)^2} = |y|$. Припадност тачке скуповима S_l, S_s или S_d врши се гранањем на основу интервала реалне праве којима припада тачка x , обично конструкцијом `else-if` како смо приказали, на пример, у задатку **Агрегатно стање**.

```
// rastojanje tacaka (x1, y1) i (x2, y2)
double rastojanje_tacaka(double x1, double y1, double x2, double y2) {
    double dx = x2 - x1, dy = y2 - y1;
    return sqrt(dx*dx + dy*dy);
}

int main() {
// ...
// rastojanje tacke od duzi
double rastojanje;
if (x < x0) // x < x0
    rastojanje = rastojanje_tacaka(x, y, x0, 0);
else if (x <= x1) // x0 <= x <= x1
    rastojanje = abs(y); // rastojanje_tacaka(x, y, x, 0)
else // x > x1
    rastojanje = rastojanje_tacaka(x, y, x1, 0);

// ispisujemo rezultat
cout << fixed << setprecision(5) << showpoint
    << rastojanje << endl;
return 0;
}
```

Примена симетрија

Анализа би била једноставнија ако бисмо заједно померили (транслирали) тачку A и дуж A_0A_1 (не мењајући њихов међусобни положај), тако да се дуж нађе на x оси са средиштем у координатном почетку, а тачка A у првом квадранту или на његовој граници. Ово можемо да постигнемо у неколико корака.

- Најпре дуж A_0A_1 и тачку A померамо за вектор $(-x_0 - \frac{dx}{2}, 0)$, где је $dx = x_1 - x_0$ дужина дужи A_0A_1 . Ово чинимо тако што на координате крајњих тачака дужи A_0A_1 и тачке A додамо координате поменутог вектора. Тиме смо средиште дужи довели у координатни почетак.
- Ако је x координата тачке A након транслације негативна, можемо да пресликамо тачку A симетрично у односу на y осу. То постижемо тако што x координату тачке A заменимо њеном апсолутном вредношћу. Тиме се растојање тачке A од дужи A_0A_1 неће променити, а x координата ће бити већа или једнака нули.
- На крају, ако је y координата тачке A негативна, можемо да пресликамо тачку A симетрично у односу на x осу, тј. y координату тачке A заменимо њеном апсолутном вредношћу (растојање тачке и дужи ће и даље бити исто).

После описаних померања тачке и дужи, дате тачке имају нове координате: $A'_0 = (x'_0, y'_0) = (-\frac{dx}{2}, 0)$, $A'_1 = (x'_1, y'_1) = (\frac{dx}{2}, 0)$ и $A' = (x', y') = (|x - x_0 - \frac{dx}{2}|, |y|)$. Сада је анализа случајева једноставнија. Ако је $x' \leq x'_1$, тада је најближа тачка тачки A' њена нормална пројекција на дуж $A'_0A'_1$ и растојање је једнако y' , а ако је $x' > x'_1$ тада је најближа тачка тачка A'_1 и растојање се може израчунати као растојање између тачака A' и A'_1 .

```
// simetricno preslikavamo konfiguraciju tako da je srediste duzi u
// koordinatnom pocetku, a tacka u prvom kvadrantu
double dx = x1 - x0;
// koordinate u transformisanom koordinatnom sistemu
double x0t, x1t, xt, yt;
x0t = -dx/2; x1t = dx/2;
```

2.3. УГНЕЖЂЕНО ГРАНАЊЕ

```
xt = abs(x - x0 + (-dx/2)); yt = abs(y);  
  
// izracunavamo rastojanje  
double rastojanje;  
if (xt <= x1t)  
    rastojanje = yt;  
else  
    rastojanje = rastojanje_tacaka(xt, yt, x1t, 0);
```

Задатак: Школарина

У једној приватној школи уведено је правило којим се одређује износ попушта који остварују ученици приликом уписа у наредну школску годину. Ученици са одличним успехом остварују попуст од 40% укупног износа школарине, са врлодобрим 20% а са добрим 10%. Такође, ученици који су освојили награду на неком од државних такмичења остварују попуст од 30% укупног износа школарине. Уколико неки ученик испуњава два критеријума за попуст примењује се критеријум по коме је попуст већи. На основу пуног износа школарине, просечне оцене ученика и информације о наградама са такмичења одредити износ који ученик треба да плати при упису у наредну школску годину.

Улаз: У првој линији стандардног улаза налази се пун износ школарине (реалан број), у другој просечна оцена ученика (реалан број од 2.0 до 5.0) а у трећој 0 уколико ученик нема награду или 1 уколико је има.

Изназ: Износ школарине коју ученик треба да плати (заокружен на најближи цео број) наводи се у једној линије стандардног излаза.

Пример

Улаз	Изназ
4000	2400
4.65	
1	

Задатак: Солидарни порез

Порез је одређен на следећи начин. Првих 60 000 динара бруто плате се не опорезује. На део бруто плате између 60 000 и 100 000 динара, плаћа се 20% пореза док се на део бруто плате преко 100 000 динара плаћа порез од 25%. Напиши програм који за унуту бруто плату (плату пре одбијања пореза) израчунава нето плату (плату после одбијања пореза).

Улаз: Са стандардног улаза се учитава један реалан број – износ бруто плате.

Изназ: На стандардни излаз исписује се један реалан број – износ нето плате, заокружен на две децимале.

Пример

Улаз	Изназ
100000	92000.00

Задатак: Правоугаони прстен

У програму за рад са графиком потребно је имплементирати интеракцију мишем са правоугаонцима нацртаним на екрану, чије су ивице паралелне координатним осама. Правоугаоник се мишем превлачи по екрану тако што се ухвати за ивицу, а да би се то лакше радило допуштена је толеранција од по 5 пиксела са сваке стране ивица правоугаоника. Када је миш близу ивице правоугаоника, ивица се исцртава другом бојом. Такође, када је миш унутар правоугаоника (а када миш није близу ивице) цео правоугаоник се боји неком трећом бојом. Напиши програм који за дату позицију миша и дати правоугаоник одређује да ли је миш близу ивице (рачунајући дату толеранцију), или је унутар правоугаоника, довољно далеко од ивице или је ван правоугаоника, довољно далеко од ивице.

Улаз: Са стандардног улаза уноси се 6 целих бројева, сваки у посбној линији.

- x, y - координате миша
- x_0, y_0 - координате горњег левог темена правоугаоника
- w, h - ширина и висина правоугаоника.

Напомена: координата y у рачунарској графици расте одозго наниже.

Излаз: На стандардни излаз исписати UNUTRA ако се миш налази унутар (проширених) ивица правоугаоника, SPOLJA ако је миш ван (проширених) ивица правоугаоника, тј. NA IVICI ако се налази на (проширеној) иници правоугаоника.

Пример 1		Пример 2		Пример 3	
Улаз	Излаз	Улаз	Излаз	Улаз	Излаз
101	NA IVICI	106	UNUTRA	94	SPOLJA
101		106		94	
100		100		100	
100		100		100	
100		50		50	
100		100		100	

Задатак: Оцена на испиту

Оцена на испиту одређује се на основу броја освојених поена (може се освојити између 0 и 100 поена). Сви студенти који су добили мање од 51 поен аутоматски падају испит и добијају оцену 5. Оцена 6 се добија за број поена већи или једнак од 51, а мањи од 61, оцена 7 за број поена већи или једнак од 61, а мањи од 71, оцена 8 за број поена већи или једнак од 71, а мањи од 81, оцена 9 за број поена већи или једнак од 81 а мањи од 91, а оцена 10 за број поена већи или једнак од 91.

Улаз: Са стандардног улаза учитава се један цео број између 0 и 100 који представља број поена освојених на испиту.

Излаз: На стандардни излаз исписати један цео број - оцену на испиту.

Пример 1		Пример 2		Пример 3		Пример 4	
Улаз	Излаз	Улаз	Излаз	Улаз	Излаз	Улаз	Излаз
73	8	50	5	51	6	100	10

2.3.3 Гранање на основу коначног скупа вредности

У наредним задацима одлуку доносимо на основу испитивања неколико могућих вредности дате променљиве (на пример, редни број месеца на основу назива одређујемо провером 12 могућности).

Задатак: Редни број месеца

Напиши програм који на основу назива месеца одређује његов редни број.

Улаз: Са стандардног улаза се уноси назив месеца записан малим словима латинице.

Излаз: На стандардни излаз исписати један природан број који представља редни број месеца.

Пример

Улаз	Излаз
mart	3

Решење

Потребно је извршити класификацију на основу једне од 12 могућих вредности. Међутим, пошто је потребно унети назив месеца и класификацију извршити на основу тог текста, неопходно је употребити ниску карактера.

Једно могуће решење је решење помоћу гранања наредбама `if` тј. конструкцијом `else-if`.

```
string mesec;
cin >> mesec;

if (mesec == "januar")
    cout << 1 << endl;
else if (mesec == "februar")
    cout << 2 << endl;
// ...
```

2.3. УГНЕЖЂЕНО ГРАНАЊЕ

Језик C++ не допушта да се гранање помоћу наредбе `switch-case` врши на основу податка типа `string`.

Види групација решења овој задајци.

Задатак: Број дана у месецу

Напиши програм који за дати редни број месеца и годину одређује број дана у том месецу. Водити рачуна о томе да ли је година преступна (година је преступна ако је дељива са 4, а није дељива са 100, осим ако је дељива са 400, када јесте преступна).

Улаз: Са стандардног улаза учитавају се два броја:

- број месеца m ($1 \leq m \leq 12$) и
- број године g ($1900 \leq g \leq 2100$)

Излаз: На стандардни излаз исписати један цео број који представља број дана у задатом месецу.

Пример 1		Пример 2		Пример 3		Пример 4	
Улаз	Излаз	Улаз	Излаз	Улаз	Излаз	Улаз	Излаз
11	30	2	28	2	29	2	29
2014		2014		2016		2000	
Пример 5		Пример 6		Пример 7			
Улаз	Излаз	Улаз	Излаз	Улаз	Излаз		
2	28	4	30	8	31		
2100		2019		2018			

Решење

У овом задатку вршимо гранање на основу могућих (дискретних) вредности променљиве `mesec` (веома слично задатку [Назив месеца](#)). Тај облик гранања најлакше се имплементира наредбом `switch-case`. У случају месеца фебруара потребно је додатно извршити гранање на основу тога да ли је година преступна или није. Као што смо видели у задатку [Преступна година](#), година је преступна ако је дељива са 4 и није дељива са 100 или је дељива са 400.

```
// provera da li je data godina prestupna
bool prestupna(int godina) {
    // godina je prestupna ako je deljiva sa 4 i nije deljiva sa 100,
    // ili ako je deljiva sa 400
    return (godina % 4 == 0 && godina % 100 != 0) || (godina % 400 == 0);
}

int main() {
    // učitavamo mesec i godinu
    int mesec, godina;
    cin >> mesec >> godina;

    // odredjujemo broj dana u tom mesecu
    int brojDana = 0;
    switch(mesec) {
        // januar, mart, maj, jul, avgust, oktobar, decembar
        case 1: case 3: case 5: case 7: case 8: case 10: case 12:
            brojDana = 31;
            break;
        // april, jun, septembar, novembar
        case 4: case 6: case 9: case 11:
            brojDana = 30;
            break;
        // februar
        case 2:
            brojDana = prestupna(godina) ? 29 : 28;
            break;
    }
}
```



```
// ispisujemo rezultat
cout << brojDana << endl;
return 0;
}
```

Гранање је могуће остварити помоћу узастопних наредби `if` или још ефикасније помоћу конструкције `else-if` (слично као што се вршило гранање на основу припадности интервалима у задатку [Агрегатно стање](#)). Све месеце са истим бројем дана можемо груписати у исту грану.

```
// odredjujemo broj dana u tom mesecu
int brojDana = 0;
// januar, mart, maj, jul, avgust, oktobar, decembar
if (mesec == 1 || mesec == 3 || mesec == 5 || mesec == 7 ||
    mesec == 8 || mesec == 10 || mesec == 12)
    brojDana = 31;
// april, jun, septembar, novembar
else if (mesec == 4 || mesec == 6 || mesec == 9 || mesec == 11)
    brojDana = 30;
// februar
else if (mesec == 2)
    brojDana = prestupna(godina) ? 29 : 28;
```

Види груписања решења овог задатка.

Задатак: Назив месеца

Напиши програм који на основу унетог редног броја месеца исписује његов назив.

Улаз: Са стандардног улаза се уноси редни број месеца - природан број између 1 и 12.

Излаз: На стандардни излаз исписати назив месеца (исписан латиничким ASCII карактерима).

Пример

Улаз	Излаз
1	januar

2.3.4 Лексикографско поређење торки

Ако је потребно упоредити два временска тренутка, прво се пореде сати, а тек ако су сати једнаки, онда се пореде минути. Овакав начин поређења се у пракси често среће. Рећи ћемо да се уређене n -торке пореде *лексикографски*, ако се поредак одређује тако што се редом, пореди једна по једна позиција. Илуструјмо ово кроз неколико задатака.

Задатак: Пунолетство

Написати програм којим се испитује да ли ће особа чији је датум рођења познат бити пунолетна (имати пуних 18 година) неког задатог датума.

Улаз: У прве три линије се уноси датум рођења у редоследу дан, месец и година рођења. У следеће три задати датум у редоследу дан, месец и година. Оба датума су исправна.

Излаз: Исписати на стандардном излазу: DA - ако ће особа бити пунолетна, и NE - ако особа неће бити пунолетна. Ако је један задати датум тачно 18 година након датума рођења, особа се сматра пунолетном (без обзира на тачно време рођења).

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
1	DA	1	NE
5		5	
1986		1986	
1		30	
5		4	
2004		2004	

Решење

У задатку се захтева поређење два датума - датума који је тачно 18 година након године рођења и задатог датума у којем се испитује пунолетство. Ако се датуми представе уређеним тројкама облика (g, m, d) тада се поређење датума своди на лексикографско поређење ових уређених тројки. Постоје различити начини да се то уради.

Лексикографско поређење помоћу једног израза

Особа рођена на дан (d_r, m_r, g_r) биће пунолетна (имати пуних 18 година) на дан (d, m, g) :

- ако је $g_r + 18 < g$, или;
- ако је $g_r + 18 = g$, али и $m_r < m$, или;
- ако је $g_r + 18 = g$ и $m_r = m$, али и $d_r \leq d$.

Дакле, на основу овога могуће је формирати сложени логички израз који одређује да ли је особа пунолетна.

```
int d1, m1, g1; // datum rodjenja
int d2, m2, g2; // datum u kom se ispituje punoletstvo
cin >> d1 >> m1 >> g1
    >> d2 >> m2 >> g2;
if ((g2 > g1 + 18) ||
    (g2 == g1 + 18 && m2 > m1) ||
    (g2 == g1 + 18 && m2 == m1 && d2 >= d1))
    cout << "DA" << endl;
else
    cout << "NE" << endl;
```

Лексикографско поређење помоћу угнежђеног гранања

Постоји могућност да се поређење тројки реализује угнежђеним (хијерархијским) гранањем којим може да се утврди да ли је прва тројка лексикографски испред друге, да ли је иза ње или да ли су тројке једнаке. Такво лексикографско поређење почиње поређењем прве компоненте (у овом случају године). Ако је година у првом датуму мања од године у другом датуму онда је први датум пре другог, ако је година у првом датуму већа од године у другом датуму онда је први датум после другог, а ако су године једнаке, наставља се поређење наредних компоненти (месеци, евентуално и дана) по истом принципу. Приликом поређења могуће је постављати вредност променљиве која чува резултат поређења.

```
bool punoletan;
if (g2 > g1 + 18)
    punoletan = true;
else if (g2 < g1 + 18)
    punoletan = false;
else { // g1 == g2
    if (m2 > m1)
        punoletan = true;
    else if (m2 < m1)
        punoletan = false;
    else { // m1 == m2
        if (d2 >= d1)
            punoletan = true;
        else
            punoletan = false;
        // punoletan = d2 >= d1;
    }
}
```

Имплементација функције лексикографског поређења

Лексикографско поређење је могуће издвојити у посебну функцију. Ако се проверава само да ли је једна тројка лексикографски мања (или, слично, мања или једнака, већа, или већа или једнака) тада се резултат представља логичком вредношћу. Међутим, могуће је поређење организовати тако да је резултат целобројна вредност и то негативан ако је прва тројка лексикографски испред друге, нула ако су тројке једнаке, односно позитиван ако је прва тројка лексикографски иза друге. Оваква дефиниција је погодна јер се овакав резултат може добити простим одузимањем одговарајућих вредности.

```

// poredi datume i vraca:
// negativan rezultat ako je prvi datum pre drugog,
// nulu ako su datumi jednaki
// pozitivan rezultat ako je prvi datum posle drugog
int porediDatume(int d1, int m1, int g1,
                 int d2, int m2, int g2) {
    if (g1 != g2)
        return g1 - g2;
    if (m1 != m2)
        return m1 - m2;
    return d1 - d2;
}

int main() {
// ...
if (porediDatume(d2, m2, g2, d1, m1, g1 + 18) >= 0)
    cout << "DA" << endl;
else
    cout << "NE" << endl;
return 0;
}

```

Види групаџија решења овој задатку.

Задатак: Бољи у две дисциплине

Такмичари су радили тестове из математике и програмирања. За сваки предмет добили су одређени број поена (цео број од 0 до 50). Такмичари се рангирају по укупном броју поена из оба предмета. Ако два такмичара имају исти број поена, победник је онај који има више поена из програмирања. Потребно је написати програм који одређује победника такмичења.

Улаз: Учитавају се подаци за два такмичара. За сваког такмичара учитава се број поена из математике, а затим број поена из програмирања, сваки у посебном реду.

Излаз: Потребно је исписати редни број победника (1 или 2). Ако су два такмичара остварила потпуно исти успех, победник је такмичар 1 (јер је остварио више поена на квалификационом такмичењу).

Пример 1

Улаз	Излаз
37	1
45	
22	
47	

Објашњење

Први такмичар укупно има $37+45=82$, а други $22+47=69$ поена, тако да је победник први такмичар.

Пример 2

Улаз

43
40
40
43

Излаз

2

Објашњење

Први такмичар укупно има $43+40=83$, а други $40+43=83$ поена. Такмичари имају исти укупан број поена, али други такмичар има више поена из програмирања тако да је он победник.

Решење

Сваки такмичар је представљен уређеним паром бројева (поенима из математике и програмирања) и потребно је одредити већи од два уређена пара, при чему је релација поретка међу тим паровима одређена условима задатка (пореди се прво укупан број поена, па онда поени из програмирања). Ово је класичан пример релације лексикографског поређења.

Вежбе ради, ту релацију можемо имплементирати и сами, у засебној функцији која прихвата податке за два такмичара (четири броја) и враћа истинитосну (буловску) вредност `true` ако је први бољи од другог у овој релацији. Различити могући начини имплементације лексикографског поређења описани су у задатку [Пунолетство](#).

Један начин је да поређење реализујемо у функцији која проверава један по један услов и ако открије да је на основу неког услова могуће вратити резултат, помоћу `return` тај резултат враћа.

```
// Provera da li je takmicar A bolji od takmicara B
bool bolji(int matA, int progA, int matB, int progB) {
    int ukupnoA = matA + progA;
    int ukupnoB = matB + progB;
    if (ukupnoA > ukupnoB)
        return true;
    if (ukupnoA < ukupnoB)
        return false;
    return progA > progB;
}

int main() {
    // broj poena prvog takmicara iz matematike i programiranja
    int mat1, prog1;
    cin >> mat1 >> prog1;

    // broj poena drugog takmicara
    int mat2, prog2;
    cin >> mat2 >> prog2;

    // Redni broj pobednika
    int pobednik;

    // drugi je pobednik ako i samo ako je bolji od prvog
    if (bolji(mat2, prog2, mat1, prog1))
        pobednik = 2;
    else
        pobednik = 1;

    cout << pobednik << endl;
    return 0;
}
```

Поређење се може реализовати и помоћу сложеног логичког израза.

```
// Provera da li je takmicar A bolji od takmicara B
bool bolji(int matA, int progA, int matB, int progB) {
    int ukupnoA = matA + progA;
    int ukupnoB = matB + progB;
    return (ukupnoA > ukupnoB) ||
           (ukupnoA == ukupnoB && progA > progB);
}
```

Види групачија решења овој задајци.

Задатак: Ко је пре стигао

Два другара, Пера и Мика, су дошла у школу. Ако се за сваког зна сат, минут и секунд када је стигао, напиши програм који одређује који од њих је стигао пре.

Улаз: Са стандардног улаза учитава се 6 целих бројева. Прво сат, минут и секунд када је стигао Пера, а затим сат, минут и секунд када је стигао Мика (претпоставља се да бројеви представљају исправно задата времена).

Излаз: На стандардни излаз исписати једну линију текста: *Pera* ако је пре стигао Пера, *Mika* ако је пре стигао Мика или *istovremeno* ако су стигли истовремено.

Пример

Улаз	Излаз
14	Pera
23	
17	
15	
23	
11	

2.3.5 Хијерархија услова

Задаци који следе карактеристични су по томе што је угнежђено гранање организовано *хијерархијски* (некада се каже у облику *дрвета одлучивања*). У зависности од одговора на свако постављено питање, одређују се даља питања која ће се постављати, све док нас прикупљени одговори не доведу до једнозначног коначног решења.

Задатак: Два броја истог знака

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види тексты задатка.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

До решења се могло доћи и угнежђеним (хијерархијским) гранањем, без коришћења логичких оператора, међутим, такво решење даје веома гломазан програмски код и треба га избегавати.

```
bool istogZnaka;
```

```
if (a > 0)
    if (b > 0)
        istogZnaka = true;
    else // b < 0
        istogZnaka = false;
else // a < 0
    if (b > 0)
        istogZnaka = false;
    else // b < 0
        istogZnaka = true;
```

```
if (istogZnaka)
    cout << "da" << endl;
else
    cout << "ne" << endl;
```

Задатак: Линеарна једначина

Напиши програм који одређује број решења линеарне једначине $a \cdot x + b = 0$ за реалне вредности коефицијената a и b и решава је ако постоји јединствено решење.

Улаз: У првом реду коефицијент a , у другом коефицијент b .

Излаз: Ако има једно решење испис решења на две децимале, ако нема решења порука: *NEMA RESENJA*, ако има бесконачно много решења порука: *BESKONACNO RESENJA*.

2.3. УГНЕЖЂЕНО ГРАНАЊЕ

Пример 1

Улаз	Израз
1	-1.00
1	

Пример 2

Улаз	Израз
0	BESKONACNO RESENJA
0	

Пример 3

Улаз	Израз
0	NEMA RESENJA
1	

Решење

Број решења линеарне једначине облика $a \cdot x + b = 0$ зависи од тога да ли су коефицијенти једнаки или различити од нуле. Ако је

- $a \neq 0$, једначина има јединствено решење $x = -\frac{b}{a}$,
- $a = 0$ једначина се своди на једначину $0 \cdot x + b = 0$, која када је $b \neq 0$ нема решења. Ако је $b = 0$ једначина гласи $0 \cdot x + 0 = 0$, па је сваки реалан број њено решење.

Након учитавања бројева могуће је извршити гранање и исписати тражени резултат.

```
double a, b;    // koeficijenti jednacine
cin >> a >> b;

if (a != 0.0) {
    // a != 0: jednacina ima jedinstveno resenje
    double x = -b / a;
    cout << fixed << showpoint << setprecision(2) << x << endl;
} else {
    // jednacina je oblika 0x+b=0:
    //   ima beskonacno mnogo resenja ako je b = 0,
    //   nema resenja ako je b != 0
    if (b == 0.0)
        cout << "BESKONACNO RESENJA" << endl;
    else
        cout << "NEMA RESENJA" << endl;
}
```

Види групачија решења овој задатку.

Задатак: Икс-окс

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шексij задатку.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Пошто је у овом задатку број случајева релативно мали, можемо га решити и анализом случајева тј. гранањем.

Угнежђено гранање

Можемо засебно одредити ком интервалу припада координата x (гранањем на основу припадности надовезаним интервалима, као, на пример у задатку [Успех ученика](#)), а затим у свакој грани одредити ком интервалу припада координата y .

```
// koordinate piksela
int x, y;
cin >> x >> y;

// redni broj kvadrata
int kvadrat;

if (y <= 100) {
    if (x <= 100)
        kvadrat = 1;
    else if (x <= 200)
        kvadrat = 2;
```

```

else
    kvadrat = 3;
} else if (y <= 200) {
    if (x <= 100)
        kvadrat = 4;
    else if (x <= 200)
        kvadrat = 5;
    else
        kvadrat = 6;
} else {
    if (x <= 100)
        kvadrat = 7;
    else if (x <= 200)
        kvadrat = 8;
    else
        kvadrat = 9;
}
cout << kvadrat << endl;

```

Засебна анализа свих могућих случајева

Пошто је могућих случајева (квадрата) само 9, могуће је и извршити исцрпну проверу свих могућих случајева. На пример, тачка припада квадрату број 1 ако важи $1 \leq x \leq 100$ и $1 \leq y \leq 100$.

```

// koordinate piksela
int x, y;
cin >> x >> y;

// redni broj kvadrata
int kvadrat;

// analiziramo sve slucajeve
if (1 <= x && x <= 100 && 1 <= y && y <= 100)
    kvadrat = 1;
if (101 <= x && x <= 200 && 1 <= y && y <= 100)
    kvadrat = 2;
// ...

// ispisujemo resenje
cout << kvadrat << endl;

```

Задатак: Квадранти и осе

Дате су координате тачке. Напиши програм који одређује у ком квадранту или на којој осе се она налази.

Улаз: Са стандардног улаза уносе се два цела броја (сваки у посебном реду): x -координата и y -координата тачке ($-10 \leq x, y \leq 10$).

Излаз: На стандардни излаз исписати да ли се тачка налази у неком од четири квадранта, да ли је на неком делу неке координатне осе или је координатни почетак.

Пример 1

Улаз	Излаз
-1 2	2. kvadrant

Пример 2

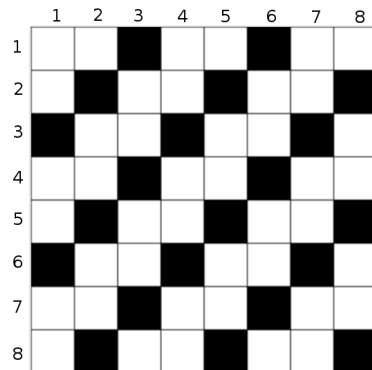
Улаз	Излаз
0 3	pozitivni deo y ose

Пример 3

Улаз	Излаз
0 0	koordinatni pocetak

Задатак: Размакнуте дијагонале

На једној необичној шаховској табли црна поља су распоређена по споредним дијагоналама тако да су у првој врсти прва два поља бела, затим иде једно црно поље, затим два бела, па црно и тако у круг. Једна таква табла је приказана на слици. Напиши програм који одређује боју поља на основу унетих координата.



Слика 2.2: Размакнуте дијагонале

Улаз: Са стандардног улаза се читавају редни бројеви врсте и колоне три поља v ($1 \leq v \leq 100$) и k ($1 \leq k \leq 100$), раздвојене једним размаком (за свако поље у посебном реду).

Излаз: На стандардни излаз исписати боју поља (црно или бело), за свако поље у посебном реду.

Пример

Улаз	Излаз
1 5	belo
2 4	belo
3 4	crno

Задатак: Положај две праве

Дате су две праве својим имплицитним једначинама (једначинама облика $Ax + By + C = 0$). Одредити њихов међусобни положај (поклапају се, паралелне су или се секу у тачки...)

Улаз: У 6 линија стандардног улаза налази се у свакој по један реалан број. Прва три броја редом представљају коефицијенте A , B и C једне праве, а друга 3 коефицијенте друге праве.

Излаз: У једној линији стандардног излаза исписује се текст `poklapaju se` за праве које се поклапају, `parallelne su` за паралелне праве или `seku se` за праве које се секу. Ако се праве секу у другој линији стандардног излаза исписују се координате пресечне тачке (2 реална броја са две децимале).

Пример

Улаз	Излаз
5	seku se
0	2.00 -2.00
-10	
1	
1	
0	

2.3.6 Минимум и максимум два броја

Максимум два броја је већи, а минимум је мањи од њих. Њихово одређивање се може веома једноставно извршити гранањем. На пример, одређивање максимума бројева a и b може се извршити на следећи начин:

```

if (a >= b)
    maks = a;
else
    maks = b;

```


Уместо наредбе гранања можемо употребити и условни израз.

```
maks = a >= b ? a : b;
```

Ипак, програми у којима је потребно одредити минимум или максимум бивају једноставнији ако се уместо гранања употребе библиотечке функције за одређивање минимума и максимума. Подсетимо се, у језику C++ то су функције `min` и `max` декларисане у заглављу `<algorithm>`.

У наставку ћемо видети неколико задатака у којима проналажење минимума и максимума парова бројева доводи до решења.

Задатак: Куглице

У свакој од две кутије налазе се само црвене и плаве куглице. Напиши програм који за дати број црвених и плавих куглица у свакој од кутија одређује најмањи број куглица које је потребно преместити из кутије у кутију да би у једној кутији биле само црвене, а у другој само плаве куглице.

Улаз: Са стандардног улаза учитавају се четири цела броја (сваки у посебном реду): број црвених куглица у првој кутији, број плавих куглица у првој кутији, број црвених куглица у другој кутији и број плавих куглица у другој кутији.

Излаз: На стандардни излаз исписати један цео број – укупан број куглица које треба преместити из кутије у кутију.

Пример

Улаз	Излаз
10	12
5	
7	
8	

Решење

Претпоставимо да је број црвених и плавих куглица у првој и другој кутији c_1, p_1 тј. c_2, p_2 . Прва могућност је да се све црвене куглице из прве кутије пребаце у другу, а да све плаве куглице из друге кутије пребаце у прву, чиме се добија $c_1 + p_2$ премештања. Друга могућност је да се све плаве куглице из прве кутије пребаце у другу, а да све црвене куглице из друге кутије пребаце у прву, чиме се добија $p_1 + c_2$ премештања. Решење се добија одређивањем мањег од ова два броја тј. њиховог *минимума*.

Одређивање минимума два броја је у програмском језику C++ најбоље урадити коришћењем библиотечке функције `min`, декларисане у заглављу `<algorithm>`.

```
int crvene1, plave1, crvene2, plave2;
cin >> crvene1 >> plave1 >> crvene2 >> plave2;
cout << min(crvene1 + plave2, crvene2 + plave1) << endl;
```

Минимум је могуће одредити а могуће је имплементирати и коришћењем гранања, на пример, коришћењем наредбе `if-else`.

```
// poredimo premestanje svih crvenih u drugu, a plavih u prvu
// i premestanje svih plavih u drugu i crvenih u prvu
int iz1U2 = crvene1 + plave2, iz2U1 = crvene2 + plave1;
int najmanjiBrojPremestanja;
if (iz1U2 < iz2U1)
    najmanjiBrojPremestanja = iz1U2;
else
    najmanjiBrojPremestanja = iz2U1;
cout << najmanjiBrojPremestanja << endl;
```

Гранање којим се одређује минимум је могуће имплементирати и коришћењем условног израза.

```
// poredimo premestanje svih crvenih u drugu, a plavih u prvu
// i premestanje svih plavih u drugu i crvenih u prvu
int iz1U2 = crvene1 + plave2, iz2U1 = crvene2 + plave1;
int najmanjiBrojPremestanja = iz1U2 < iz2U1 ? iz1U2 : iz2U1;
cout << najmanjiBrojPremestanja << endl;
```

Задатак: Проја

Кукурузни хлеб, проја, се прави мешањем кукурузног и пшеничног брашна и воде (наравно, по жељи можете додати јаја, со, сир, кајмак и слично, али ми ћемо разматрати само неопходне састојке). Неопходно је да употребљена количина кукурузног брашна буде двоструко већа од употребљене количине пшеничног брашна. Количина воде која се додаје у мешавину брашна је двоструко мања од укупне количине брашна. Одредити коју количину воде треба додати да бисмо умесили највећу могућу проју од кукурузног и пшеничног брашна којим располажемо.

Улаз: У првој линији стандардног улаза налази се количина кукурузног, а у другој количина пшеничног брашна којом располажемо (реални бројеви).

Излаз: Количину воде коју треба додати у мешавину брашна (реални број приказан на две децимале) исписати у јединој линији стандардног излаза.

Пример

Улаз	Излаз
1524	1143.00
800	

Решење

Да би се од постојећих намирница направила највећа могућа количина хлеба потребно је задовољити све услове рецепта који захтева да кукурузно брашно чини два дела смеше а пшенично један. Укупна количина брашна је збирна количина та три дела а потребна количина воде коју треба одредити је пола укупне количине брашна.

Дакле, ако са M означимо количину једног дела смеше, количина пшеничног брашна је M , кукурузног брашна је $2M$, укупна количина брашна је $M + 2 \cdot M = 3 \cdot M$, а воде је $\frac{3 \cdot M}{2}$. Ако је расположива количина пшеничног брашна P_b , а кукурузног K_b , потребно је наћи највећу вредност M такву да важи $M \leq P_b$ и $2 \cdot M \leq K_b$, тј. $M \leq \frac{K_b}{2}$. Пошто је M мање или једнако од вредности P_b и $\frac{K_b}{2}$, оно је мање или једнако од мање од те две вредности тј. од $\min(P_b, \frac{K_b}{2})$ и то је уједно највећа могућа вредност за M . Када се одреди вредност $M = \min(P_b, \frac{K_b}{2})$, израчунава се и исписује количина воде $\frac{3 \cdot M}{2}$.

Најбољи начин да се одреди минимум два броја је да се употреби библиотечка функција. У језику C++ могуће је употребити функцију `min`, декларисану у заглављу `<algorithm>`.

```
double kb, pb;
// ucitavanje postojece kolicine kukuruznog brasna
cin >> kb;
// ucitavanje postojece kolicine psenicnog brasna
cin >> pb;
// odredjivanje najvece moguće kolicine jednog dela mesavine M
double M = min(kb / 2.0, pb);
// odredjivanje i ispis potrebne kolicine vode
double v = 3.0 * M / 2.0;
cout << fixed << setprecision(2) << showpoint << v << endl;
```

Генерално, највеће x које задовољава систем неједначина $x \leq a_1, \dots, x \leq a_n$ је $x = \min(a_1, \dots, a_n)$. Дуално, најмање x које задовољава систем неједначина $x \geq a_1, \dots, x \geq a_n$ је $x = \max(a_1, \dots, a_n)$.

Задатак: Скупови спортиста

У одељењу има n ученика, од чега k ученика уме да игра кошарку, а f уме да игра фудбал. На спортско такмичење повешће се сви ученици који умеју да играју оба спорта. Колико ће најмање, а колико највише ученика ићи на такмичење?

Улаз: Са стандардног улаза читавају се три цела броја:

- n ($20 \leq n \leq 30$) - укупан број ученика у одељењу,
- k ($0 \leq k \leq n$) - број ученика који играју кошарку,
- f ($0 \leq f \leq n$) - број ученика који играју фудбал.

Излаз: На стандардни излаз исписати два цела броја (сваки у посебном реду):

- најмањи број ученика који ће ићи на такмичење,

- највећи број ученика који ће ићи на такмичење.

Пример

Улаз	Изназ
27	7
15	15
19	

Решење

На основу принципа укључивања-искључивања важи да је:

$$|K \cup F| = |K| + |F| - |K \cap F|,$$

где $|K \cup F|$ означава број елемената уније скупа кошаркаша и фудбалера, $|K|$ означава број кошаркаша и једнак је датом броју k , $|F|$ број фудбалера и једнак је датом броју f , а $|K \cap F|$ означава број оних који играју и фудбал и кошарку. Пошто су и фудбалери и кошаркаши ученици одељења у коме има n ученика важи да је $|K \cup F| \leq n$. Зато је $|K| + |F| - |K \cap F| \leq n$, тј. $|K \cap F| \geq k + f - n$. Са друге стране, важи да је $|K \cap F| \geq 0$. Зато је

$$|K \cap F| \geq \max(k + f - n, 0).$$

Докажимо да се ова доња граница увек може достићи, тј. да је најмања вредност броја ученика који су добри у оба спорта управо $\max(k + f - n, 0)$. Заиста, ако је $k + f \leq n$ могуће је да се фудбалери и кошаркаши не преклапају. Са друге стране, ако је $k + f > n$ тада је могуће да њих $k + f - n$ тренирају оба спорта, да $n - f$ игра само кошарку, а $n - k$ игра само фудбал (сви ови бројеви су ненегативни, па је могуће направити баш овакав распоред).

Са друге стране важи да је $K \cap F \subseteq K$, $K \cap F \subseteq F$, па је $|K \cap F| \leq |K| = k$, $|K \cap F| \leq |F| = f$, тј.

$$|K \cap F| \leq \min(k, f).$$

Та вредност се може постићи ако је $K \subseteq F$ или је $F \subseteq K$ тј. ако сви они који тренирају онај спорт којег тренира мање ђака, уједно тренирају и онај други спорт.

Одређивање минимума два броја је у програмском језику C++ најбоље урадити коришћењем библиотеке функције `min`, а максимума коришћењем библиотеке функције `max`.

```
int n, f, k;
cin >> n >> k >> f;
// најмањи број ученика који су добри у оба спорта
cout << max(k+f-n, 0) << endl;
// највећи број ученика који су добри у оба спорта
cout << min(k, f) << endl;
```

Задатак: Домине

Зоран има две домине. На свакој домини су тачкицама представљене две цифре од 1 до 9. Напиши програм који одређује највећи број који Зоран може написати помоћу своје две домине (сваку од њих може окренути како жели и домине може поређати како жели).

Улаз: Први ред стандардног улаза садржи бројеве на првој домини (раздвојене размаком), а други ред садржи бројеве на другој домини.

Изназ: На стандардни излаз исписати тражени највећи број.

Пример 1		Пример 2		Пример 3	
Улаз	Изназ	Улаз	Изназ	Улаз	Изназ
2 3	6132	1 9	9143	6 2	6362
1 6		3 4		3 6	

Решење

2.3. УГНЕЖЂЕНО ГРАНАЊЕ

Сваку од две домине је потребно окренути тако да већа цифра на тој домини буде испред мање (ако су цифре једнаке, свеједно је како се домина окреће). Редослед две домине се одређује тако да број записан на првој домини буде већи од броја записаног на другој.

Прву цифру сваке од две домине одређујемо као максимум две цифре на тој домини, а другу као њихов минимум. Два двоцифрена броја можемо упоредити и лексикографски (слично као у задатку Пунолетство): прва домина иде испред друге ако је њена прва цифра већа од прве цифре друге домине или су те цифре једнаке, а друга цифра прве домине је већа од друге цифре друге домине.

```
// odredjujemo vecu i manju cifru na prvoj domini
int m11 = max(d11, d12), m12 = min(d11, d12);
// odredjujemo vecu i manju cifru na drugoj domini
int m21 = max(d21, d22), m22 = min(d21, d22);
// odredjujemo bolji redosled domina leksikografskim poredjenjem parova
// (m11, m12) i (m21, m22)
if (m11 > m21 || m11 == m21 && m12 > m22)
    cout << m11 << " " << m12 << " " << m21 << " " << m22 << endl;
else
    cout << m21 << " " << m22 << " " << m11 << " " << m12 << endl;
```

Једно могуће решење је и да експлицитно формирамо двоцифрене бројеве множењем прве цифре са 10 и додавањем друге, да их затим упоредимо и да тражени четвороцифрени број добијемо множењем већег двоцифреног са 100 и додавањем мањег (ово решење је исправно јер смо сигурни да се не јавља цифра 0). Када је број формиран, цифре можемо да издвојимо коришћењем целобројног дељења са 10 (као у задатку Збир цифара четвороцифреног броја).

```
// veci dvocifreni broj koji se moze napraviti pomocu prve domine
int d1 = d11 > d12 ? 10*d11 + d12 : 10*d12 + d11;
// veci dvocifreni broj koji se moze napraviti pomocu druge domine
int d2 = d21 > d22 ? 10*d21 + d22 : 10*d22 + d21;
// najveći četvorocifreni broj koji se moze napraviti od domina
int d = d1 > d2 ? 100*d1 + d2 : 100*d2 + d1;
// ispisujemo njegovu jednu po jednu cifru
cout << (d / 1000) % 10 << " "
    << (d / 100) % 10 << " "
    << (d / 10) % 10 << " "
    << (d / 1) % 10 << endl;
```

Задатак: Интервали

Дата су два затворена интервала реалне праве $[a_1, b_1]$ и $[a_2, b_2]$. Написати програм којим се одређује: њихов покривач (најмањи интервал реалне праве који садржи дате интервале), пресек (највећи интервал реалне праве који је садржан у оба интервала ако постоји) и дужину њихове уније (дела реалне праве који ти интервали покривају).

Улаз: Са стандардног улаза учитавају се четири реална броја: a_1, b_1, a_2 и b_2 , при чему важи $a_1 < b_1$ и $a_2 < b_2$.

Израз: На стандардни излаз исписује се 5 реалних бројева (заокружених на две децимале): у првом реду исписују се леви и десни крај покривача раздвојени размаком, у наредном леви и десни крај пресека раздвојени размаком или текст `preseka ne postoji` ако се интервали не секу и у трећем реду се исписује дужина уније.

Пример 1

Улаз	Израз
1	1.00 4.00
3	2.00 3.00
2	3.00
4	

Пример 2

Улаз	Израз
0.5	-2.50 4.50
4.5	preseka ne postoji
-2.5	5.00
-1.5	

Решење

Почетак покривача два интервала је мањи од два лева краја та два интервала (тј. $a_{\text{покривач}} = \min(a_1, a_2)$),

а крај покривача је већи од два десна краја (тј. $b_{\text{pokrivac}} = \max(b_1, b_2)$). Напоменимо и да покривач увек постоји (пробај да објасниш зашто). Почетак евентуалног пресека је већи од два лева краја (тј. $a_{\text{preseka}} = \max(a_1, a_2)$), а крај евентуалног пресека је мањи од два десна краја (тј. $b_{\text{preseka}} = \min(b_1, b_2)$). Пресек постоји ако и само ако је $b_{\text{preseka}} > a_{\text{preseka}}$. На крају, дужину уније можемо једноставно одредити тако што од збира дужина једног и другог интервала одузме дужина пресека (дужину интервала рачунамо као разлику између његовог десног и левог краја).

```
// odredjujemo i ispisujemo pokrivac intervala
double aPokrivac = min(a1, a2); // prvi (levlji) pocetak
double bPokrivac = max(b1, b2); // drugi (desnji) kraj
cout << fixed << showpoint << setprecision(2)
    << aPokrivac << " " << bPokrivac << endl;

// odredjujemo i ispisujemo preseka intervala i njegovu duzinu
double aDesni = max(a1, a2); // drugi (desnji) pocetak
double bLevi = min(b1, b2); // prvi (levlji) kraj
double duzinaPreseka;
if (bLevi >= aDesni) {
    cout << fixed << showpoint << setprecision(2)
        << aDesni << " " << bLevi << endl;
    duzinaPreseka = bLevi - aDesni;
} else {
    cout << "preseka ne postoji" << endl;
    duzinaPreseka = 0.0;
}

// odredjujemo i ispisujemo duzinu unije intervala
double duzina1 = b1 - a1, duzina2 = b2 - a2;
double duzinaUnije = duzina1 + duzina2 - duzinaPreseka;
```

Задатак: Позитиван део интервала

На целобројној бројевној правој дат је интервал $[a, b]$. Напиши програм који одређује дужину дела тог интервала који припада позитивном делу праве.

Улаз: Са стандардног улаза учитавају се два цела броја a и b ($-10^5 \leq a \leq b \leq 10^5$).

Излаз: На стандардни излаз исписати један цео број који представља тражену дужину позитивног дела интервала.

Пример 1		Пример 2		Пример 3	
Улаз	Излаз	Улаз	Излаз	Улаз	Излаз
-3	5	2	3	-3	0
5		5		-1	

Решење

Један начин да одредимо дужину дела интервала $[a, b]$ који лежи на позитивном делу праве је да применимо технику проналажења пресека два интервала (приказану у задатку [Интервали](#)), при чему је први интервал $[a, b]$, а други је $[0, +\infty)$. Пресек одређујемо тако што одредимо већи од два почетка (то је број $\max(a, 0)$) и леви од два краја (пошто је један крај $+\infty$ то је број b), и затим ако је крај десно од почетка та два броја одузмемо, док је у супротном резултат нула.

```
// ucitavamo interval
int a, b;
cin >> a >> b;
// izracunavamo i ispisujemo duzinu pozitivnog dela tog intervala
int p = max(b - max(0, a), 0);
cout << p << endl;
```

Још један начин је да одредимо десни и леви крај пресека и да их одузмемо. Десни крај је већи од бројева b и 0 , док је леви крај већи од бројева a и 0 . Зато је тражена дужина једнака $\max(b, 0) - \max(a, 0)$.

2.3. УГНЕЖЂЕНО ГРАНАЊЕ

```
// izracunavamo i ispisujemo duzinu pozitivnog dela tog intervala
int p = max(0, b) - max(0, a);
cout << p << endl;
```

Задатак се може решити и тако што се примени угнежђено (хијерархијско) гранање. Ако је $b \leq 0$ тада цео интервал лежи лево од координатног почетка и дужина његовог позитивног дела је 0. У супротном, ако је $a \geq 0$ тада цео интервал лежи десно од координатног почетка и дужина позитивног дела је $b - a$, а у супротном, ако је $a < 0$ тада је позитивни део интервала интервал $[0, b]$ чија је дужина b .

```
// izracunavamo i ispisujemo duzinu pozitivnog dela tog intervala
int p;
if (b <= 0) // ceo interval lezi na nepozitivnom delu prave
    p = 0;
else // desni kraj intervala je pozitivan
    if (a >= 0) // pocetak pozitivnog dela intervala je levi kraj
        p = b - a;
    else // pocetak pozitivnog dela intervala je 0
        p = b - 0;
cout << p << endl;
```

Задатак: Део правоугаоника у првом квадранту

Дат је правоугаоник чије су странице паралелне координатним осама. Одредити површину дела тог правоугаоника који припада првом квадранту.

Улаз: Са стандардног улаза читавају се 4 цела броја који представљају координате темена једне дијагонале правоугаоника.

Излаз: На стандардни излаз исписује се један цео број - површина дела правоугаоника у првом квадранту.

Пример

Улаз	Излаз
-3	4
5	
2	
7	

Решење

Ако одредимо доње лево и горње десно теме (што можемо урадити применом минимума и максимума), тада површину можемо израчунати анализом разних случајева. Прво, ако се горње десно теме налази ван првог квадранта, тада је површина једнака нули. У супротном анализирамо положај доњег левог темена и у зависности од квадранта коме оно припада одређујемо површину.

```
// odredjujemo levu i desnu ivicu
int xl = min(x1, x2), xd = max(x1, x2);
// odredjujemo gornju i donju ivicu
int yd = min(y1, y2), yg = max(y1, y2);

// površina pravougaonika
int P;
if (xd <= 0 || yg <= 0) // gornje desno teme nije u 1. kvadrantu
    P = 0;
else if (xl <= 0 && yd <= 0) // donje levo teme je u 3. kvadrantu
    P = xd * yg;
else if (xl > 0 && yd <= 0) // donje levo teme je u 4. kvadrantu
    P = (xd - xl) * yg;
else if (xl <= 0 && yd > 0) // donje levo teme je u 2. kvadrantu
    P = xd * (yg - yd);
else // donje levo teme je 1. kvadrantu
    P = (xd - xl) * (yg - yd);

cout << P << endl;
```

Задатак једноставније можемо решити тако што одредимо дужину дела горње странице правоугаоника и дужину дела десне странице правоугаоника који припадају првом квадранту (површина дела правоугаоника у првом квадранту је онда производ те две дужине). То се своди на проналажење дела дужи на x -оси тј. y -оси који припадају позитивном делу те осе, што је проблем који смо већ решавали у задатку **Позитиван део интервала**. Једина разлика је то што не знамо у ком редоследу су задате координате x_1 и x_2 нити у ком редоследу су задате координате y_1 и y_2 , тако да се пре одређивања дужине позитивног дела сваког од њих врши одређивање мањег и већег од њих (ако су a и b бројеви који представљају крајње тачке неког интервала, онда се помоћу $l = \min(a, b)$ и $d = \max(a, b)$ могу одредити леви и десни крај тог интервала).

```
// a i b su krajnje tacke duzi koja lezi na koordinatnoj osi,
// odredjuje se deo te duzi koji pripada pozitivnom delu te ose
int pozitivanDeoDuzi(int a, int b) {
    // levi i desni kraj duzi
    int l = min(a, b);
    int d = max(a, b);
    return max(d, 0) - max(l, 0);
}

int main() {
    // koordinate temena dve dijagonale pravougaonika
    int x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;

    // duzina dela desne ivice koji je desno od y-ose
    int ax = pozitivanDeoDuzi(x1, x2);
    // duzina dela gornje ivice koji je iznad x-ose
    int by = pozitivanDeoDuzi(y1, y2);
    // izracunavamo i ispisujemo površinu
    cout << ax * by << endl;
    return 0;
}
```

Задатак: Темена правоугаоника

Библиотека за цртање омогућава цртање правоугаоника тако што се задају прво координате његовог горњег левог темена, а затим координате његовог доњег десног темена. Нама су познате координате крајњих тачака неке од његових дијагонала. Напиши програм који на основу тих координата одређује координате потребне за цртање. Координатни систем је организован тако да x -координата расте са лева на десно, док y -координата расте одозго наниже.

Улаз: У првом реду стандардног улаза налазе се координате темена дијагонале (свако теме у посебном реду, при чему су координате раздвојене размаком).

Излаз: На стандардни излаз исписати координате горњег левог и доњег десног темена (свако теме у посебном реду, при чему су координате раздвојене размаком).

Пример

Улаз	Излаз
40 80	40 30
90 30	90 80

Задатак: H₂SO₄

Колико се молекула супорне киселине H_2SO_4 може направити од датог броја атома водоника H , сумпора S и кисеоника O ?

Улаз: У свакој од две линије стандардног улаза налази се цео број који представља дати број атома, редом, водоника, сумпора, па кисеоника.

Излаз: У једној линији стандардног излаза исписати цео број који представља број могућих молекула сумпорне киселине.

Пример

2.3. УГНЕЖЂЕНО ГРАНАЊЕ

Улаз	Изназ
75	26
58	
106	

Објашњење

Датих 75 атома водоника даје довољно атома водоника за 37 молекула H_2SO_4 и један додатан неискоришћени атом водоника, датих 58 атома сумпора за тачно 58 молекула H_2SO_4 , док датих 106 атома кисеоника даје довољно атома кисеоника за 26 молекула H_2SO_4 и два додатна неискоришћена атома кисеоника. Ограничавајући фактор је најмањи од ових бројева молекула, а то је 26.

Види групација решења овој задатку.

Задатак: Сијалице

У соби постоје плава и жута сијалица. За сваку сијалицу је познато време (у облику сат, минут, секунд) када је укључена и када је искључена. Напиши програм који одређује колико времена је соба била осветљена плаво, колико времена је била осветљена зелено и колико времена је била осветљена жуто (соба је осветљена зелено док су истовремено укључене плава и жута сијалица).

Улаз: Са стандардног улаза се уносе четири реда са по три броја раздвојена размацама (сат, минут и секунд укључивања плаве сијалица, сат, минут и секунд њеног искључивања, сат, минут и секунд укључивања жуте сијалице и сат, минут и секунд њеног искључивања).

Изназ: У првој линији стандардног излаза исписати три броја раздвојена двотачкама која представљају време (у сатима, минутима и секундама) које је соба била обојена плаво, у наредној линији у истом облику исписати време које је соба била обојена зелено и у последњој линији у истом облику исписати време које је соба била обојена жуто.

Пример 1		Пример 2	
Улаз	Изназ	Улаз	Изназ
13 15 0	0:30:0	10 50 20	1:49:51
14 20 0	0:35:0	12 40 11	0:0:0
13 45 0	1:10:0	8 45 30	1:44:50
15 30 0		10 30 20	

Задатак: Краљево растојање

Напиши програм који одређује колико је потеза краљу на шаховској табли потребно да дође на дато поље. Краљ се у сваком потезу може померити за једно поље у било ком од 8 смерова (горе, доле, лево, десно и дијагонално).

Улаз: Са стандардног улаза се читавају четири броја између 1 и 8 (у свакој линији по два, раздвојена размаком). Прва два броја представљају координате полазног, а друга два броја координате долазног поља.

Изназ: На стандардни излаз исписати један цео број који представља најмањи број потеза потребан краљу да са полазног стигне до долазног поља.

Пример 1		Пример 2	
Улаз	Изназ	Улаз	Изназ
1 2	2	8 3	6
3 4		2 5	

Задатак: Део квадрата у првом октанту

Дат је квадрат чије су ивице паралелне координатним осама. Одредити запремину дела тог квадрата који припада првом октанту (део простора у коме су све координате позитивне).

Улаз: Са стандардног улаза читава се 6 целих бројева који представљају редом 3 координате једног темена, затим 3 координате другог темена исте просторне дијагонала квадрата.

Изназ: На стандардни излаз исписује се један цео број - запремина дела квадрата у првом октанту.

Пример 1

Улаз	Изназ
-2	36
4	
1	
3	
2	
7	

Пример 2

Улаз	Изназ
9	0
2	
-5	
1	
6	
-2	

2.3.7 Разни задаци са гранањем**Задатак: Сутрашњи датум**

Напиши програм који за унети датум одређује датум наредног дана.

Улаз: Са стандардног улаза се уносе три позитивна цела броја (сваки у засебном реду) која представљају дан, месец и годину једног исправног датума.

Изназ: На стандардни излаз исписати три цела броја која представљају дан, месец и годину сутрашњег датума. Сви бројеви се исписују у једном реду, а иза сваког броја наводи се тачка.

Пример 1

Улаз	Изназ
1	2.1.2016.
1	
2016	

Пример 2

Улаз	Изназ
28	29.2.2016.
2	
2016	

Пример 3

Улаз	Изназ
28	1.3.1900.
2	
1900	

Пример 4

Улаз	Изназ
31	1.1.2018.
12	
2017	

Решење

Као и увек у раду са датумима, пожељно је на располагању имати функцију која одређује број дана у сваком месецу (за то је потребно имати могућност провере и да ли је година преступна), коју смо показали у задатку [Број дана у месецу](#).

Да бисмо одредили сутрашњи дан, прво ћемо увећати дан. У највећем броју случајева то је довољно, међутим, ако се након увећања дана добије непостојећи датум тј. ако увећани дан прелази број дана у том месецу, тада се прелази на први дан наредног месеца (тако што се месец увећа за један). Опет је у највећем броју (преосталих) случајева то довољно. Једини случај који још није покривен настаје ако је данашњи дан 31. децембар тј. ако се увећањем броја месеца добије месец који је већи од 12. У том случају се прелази на први јануар наредне године (тако што се месец постави на један, а година увећа за један).

Једно решење је такво да се за представљање сутрашњег дана користе посебне променљиве, независне од оних за представљање данашњег дана.

```
// ucitavamo danasnji datum
int dan_danas, mesec_danas, godina_danas;
cin >> dan_danas >> mesec_danas >> godina_danas;

// izracunavamo sutrasnji datum
int dan_sutra, mesec_sutra, godina_sutra;
if (dan_danas + 1 <= brojDanaUMesecu(mesec_danas, godina_danas)) {
    dan_sutra = dan_danas + 1;
    mesec_sutra = mesec_danas;
    godina_sutra = godina_danas;
} else {
    // danas je poslednji dan u mesecu
    if (mesec_danas + 1 <= 12) {
        dan_sutra = 1;
        mesec_sutra = mesec_danas + 1;
```

2.3. УГНЕЖЂЕНО ГРАНАЊЕ

```
    godina_sutra = godina_danas;
} else {
    // danas je poslednji dan u poslednjem mesecu (31. 12.)
    dan_sutra = 1;
    mesec_sutra = 1;
    godina_sutra = godina_danas + 1;
}
}

// ispisujemo sutrasnji datum
cout << dan_sutra << "." << mesec_sutra << "."
    << godina_sutra << "." << endl;
```

Види групачија решења овој задајци.

Задатак: Јучерашњи датум

Напиши програм који за унети датум одређује датум претходног дана.

Улаз: Са стандардног улаза се уносе три позитивна цела броја (сваки у засебном реду) која представљају дан, месец и годину једног исправног датума.

Изназ: На стандардни излаз исписати три цела броја која представљају дан, месец и годину јучерашњег датума. Сви бројеви се исписују у једном реду, а иза сваког броја наводи се тачка.

Пример 1		Пример 2		Пример 3	
Улаз	Изназ	Улаз	Изназ	Улаз	Изназ
1	29	1	28.2.1900.	1	31.12.2013.
3	2	3		1	
2016	2016	1900		2014	

Задатак: Врста троугла на основу углова

Дата су три конвексна угла изражена у степенима и минутима. Написати програм којим се проверава да ли то могу бити углови троугла, и ако могу какав је то троугао у односу на углове (оштроугли, правоугли или тупоугли).

Улаз: Са стандардног улаза уноси се шест целих бројева, сваки у посебној линији. Бројеви представљају три угла изражена у степенима. Подаци представљају исправно задате углове, степени и минути одређују исправно записане углове у интервалу од 0 степени и 0 минута, до 180 степени и 0 минута (минути су увек број између 0 и 59).

Изназ: Једна линија стандардног излаза која садржи реч `ostrougli`, `pravougli` или `tupougli`, ако троугао са датим угловима постоји, у зависности од врсте троугла, или реч `ne` ако не постоји троугао са датим угловима.

Пример

```
Улаз      Изназ
35        tupougli
23
92
37
52
0
```

Задатак: Врста троугла на основу страница

Напиши програм који на основу дужина страница троугла проверава да ли такав троугао постоји и да ли је једнакостранични, једнакокраки или разностранични.

Улаз: Стандардни улаз садржи три природна броја a , b и c , сваки у посебном реду.

Изназ: На стандардни излаз исписати `trougao ne postoji` ако не постоји троугао чије су дужине страница a , b и c , односно `jednakostranichni`, `jednakokraki` или `raznostranichni`, ако су ти бројеви дужине страница одговарајућег троугла.

Пример 1

<i>Улаз</i>	<i>Излаз</i>
3	raznostranicni
4	
5	

Пример 2

<i>Улаз</i>	<i>Излаз</i>
3	jednakokraki
4	
5	

Пример 3

<i>Улаз</i>	<i>Излаз</i>
1	trougao ne postoji
5	
1	

Задатак: Тенис

Напиши програм који на основу броја освојених гема два играча одређује исход сета у тенису (сет није последњи и при резултату 6:6 се игра тај-брејк).

Улаз: Са стандардног улаза се уносе два природна броја (раздвојена размаком) који представљају број освојених гема сваког играча редом.

Излаз: Ако је први играч освојио сет на стандардни излаз исписати поруку `robedio prvi`. Ако је други играч освојио сет исписати `robedio drugi`. Ако унети резултат неисправан исписати `neispravno`. Ако сет још није завршен исписати `ni je zavrшено`.

Пример 1

<i>Улаз</i>	<i>Излаз</i>
7 5	robedio prvi

Пример 2

<i>Улаз</i>	<i>Излаз</i>
3 7	neispravno

Глава 3

Итерација

Итерација подразумева понављање одређене процедуре, тако што се она у сваком кораку примењује на резултате њеног претходног извршавања, све док се не дође до жељеног резултата.

Један типичан пример итеративног поступка може бити израчунавање збира више бројева. На пример, када израчунавамо збир бројева 1, 5, 4, 8 и 2, крећемо од збира $1 + 5 = 6$, затим увећавамо тај збир за 4 тј. израчунавамо збир $6 + 4 = 10$, затим увећавамо тај збир за 8 тј. израчунавамо збир $10 + 8 = 18$ и на крају увећавамо тај збир за 2 тј. израчунавамо збир $18 + 2 = 20$.

У овом поглављу упознаћемо се са најчешће коришћеним итеративним поступцима и могућим начинима њиховог имплементирања.

3.1 Основни алгоритми над малим серијама елемената

Рачунари су јако добри у понављању истих поступака велики број пута и итеративни алгоритми се често примењују за обраду великих количина података (на пример, рачунар може да сабере милион бројева у делићу секунде). Итеративни алгоритми се у том случају програмирају уз помоћ **петљи** (некада се каже и **циклуса**). Ипак, да бисмо вам олакшали разумевање неколико најчешће коришћених итеративних поступака, у почетку ћемо их испрограмирати у варијанти у којој се понављање врши мали број пута (3-5) и такве програме можемо написати и без коришћења петљи (навођењем суштински идентичног кода неколико пута).

Најчешће коришћени итеративни алгоритми служе за анализу **бројевних серија**, које се понекад називају и *низови* или *секвенце* бројева. Овде ће термин *низ* бити коришћен само за серије бројева који су у програму смештени у једној променљивој сложеног типа (о овоме више касније).

3.1.1 Ажурирање вредности променљивих

Дефинисањем променљиве у програму ми јој додељујемо неку вредност. У досадашњим задацима смо могли да променљивама додељујемо вредност само једном, приликом њиховог дефинисања. На тај начин променљиве све време имају исту вредност током рада програма. У итеративним поступцима које ћемо сада упознати, биће потребно да променљивама мењамо вредности током извршавања програма (тима се и оправдава њихов назив - променљиве). У наставку ћемо описати елементе програмског језика који се тичу измене (ажурирања) постојећих вредности променљивих.

Елементи програмског језика

Променљива може променити своју вредност тако што јој се **додели** нова вредност. На пример, наредни програм исписује 1, а затим 2, јер се током његовог извршавања вредност променљиве x мења са 1 на 2.

```
x = 1;
cout << x << endl;
x = 2;
cout << x << endl;
```

Често нова вредност променљиве зависи од њене старе вредности. Чест случај јесте повећање вредности неке променљиве за 1 (на пример, приликом пребројавања колико има објеката који задовољавају неки скуп

услова), а наравно могуће је повећати вредност променљиве и за неку другу вредност која је различита од 1 (на пример, приликом рачунања суме бројева из неког датог скупа суму ћемо увећавати за вредност сваког од елемената тог скупа). Некад је потребно смањити текућу вредност променљиве, повећати је одређени број пута или смањити одређен број пута. Оваква измена променљиве x може се постићи изразом облика $x = x \text{ op } v$, где је op неки од наведених оператора $+$, $-$, $*$, $/$, а v вредност за коју се променљива мења. На пример, наредбом `zbir = zbir + a` вредност променљиве `zbir` се увећава за вредност променљиве `a`, док се наредбом `stepen = stepen * 2` вредност променљиве `stepen` увећава два пута. Ове наредбе не треба мешати са сличним записима из математике, који имају сасвим другачије значење (нпр. $x = x + 1$ би у математици могло да представља једначину која нема решења).

Наредбе додељивања вредности облика $x = x \text{ op } v$ могу се скраћено записати као $x \text{ op} = v$ и у пракси се често користи скраћени запис. На пример, уместо писања `zbir = zbir + x`, можемо скраћено записати `zbir += x`. Оператори доделе `+=`, `-=`, `*=`, `/=` називају се **сложени оператори доделе**. Поред наведених оператора као сложени оператор може се користити и оператор `%=`, али је његова употреба ређа од осталих сложених оператора доделе.

На располагању имамо и унарни **оператор инкрементирања** `++` којим се вредност неке променљиве повећава за 1 и унарни **оператор декрементирања** `--` којим се вредност променљиве смањује за 1. Ови оператори имају највиши приоритет од свих оператора. Оно што је важно поменути јесте да оба оператора имају две варијанте: **префиксну** и **постфиксну**. Оне се разликују у следећем: вредност израза `x++` је вредност променљиве `x` пре увећања њене вредности, док је вредност израза `++x` увећана вредност променљиве `x`. Дакле, наредни фрагмент кода:

```
int x = 5;
cout << x << endl;
cout << x++ << endl;
cout << x << endl;
cout << ++x << endl;
```

би исписивао редом вредности 5 (иницијална вредност променљиве), 5 (вредност израза `x++`), 6 (вредност променљиве `x` након операције инкрементирања) и 7 (вредност израза `++x`). У решењима у овој збирци обично се нећемо ослањати на вредност израза са операторима инкрементирања и декрементирања (користићемо их обично изоловано, само да би се повећала или смањила вредност променљиве).

Улазно-излазни параметри функција

Измена вредности променљивих некада се врши и у функцијама. Функција као улазни параметар може примити стару, а као резултат вратити нову жељену вредност променљиве и измена вредности променљиве се тада врши на месту позива функције (изразом облика $x = f(x)$). На пример,

```
int PovecajZaDva(int x) {
    return x + 2;
}

int main() {
    int x = 0;
    cout << "Pocetna vrednost x je " << x << endl;
    x = PovecajZaDva(x);
    cout << "Nova vrednost x je " << x << endl;
    return 0;
}
```

Променљиве се у функцију преносе **по вредности**, што значи да се у функцијама праве копије променљивих и измена вредности променљивих у функцијама се не преноси аутоматски на вредности променљивих коришћених у позиву. Илуструјмо ово примером.

```
void PovecajZaDva(int x) {
    cout << "Vrednost x na ulazu u funkciju je " << x << endl;
    x = x + 2;
    cout << "Vrednost x na izlazu iz funkcije je " << x << endl;
}

int main() {
```

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

```
int x = 0;
cout << "Pocetna vrednost x je " << x << endl;
PovecajZaDva(x);
cout << "Nova vrednost x je " << x << endl;
}
```

На почетку главне функције `main` променљива `x` има вредност 0. Она се преноси у функцију тако што функција има своју копију променљиве `x` која се такође иницијализује на 0 (јер је то вредност наведена у позиву). У функцији се та копија увећава за два и на излазу из функције исписује се да је вредност променљиве `x` једнака 2. Међутим, по повратку у главну функцију копија нестаје, а оригинално `x` и даље има вредност 0.

Видели смо раније да функција може имати улазне и излазне параметре. Међутим, понекад имамо потребу да дефинишемо функције које имају улазно-излазне параметре: њима се преносе почетне вредности у саму функцију, али се након извршавања функције путем њих враћају и резултујуће вредности. То је могуће урадити **преносом по референци**. Нећемо у овом тренутку детаљно објашњавати како референце функционишу, већ нам је довољно да знамо да их користимо тако што иза типа параметра наведемо симбол `&`. На пример, уколико функција треба да повећа за два вредност свог аргумента, то можемо урадити посредством улазно-излазних параметара.

```
void PovecajZaDva(int& x) {
    x = x + 2;
}
```

Позив ове функције се врши на следећи начин.

```
int main() {
    int x = 0;
    cout << "Pocetna vrednost x je " << x << endl;
    PovecajZaDva(x);
    cout << "Nova vrednost x je " << x << endl;
}
```

Улазно-излазни аргументи морају бити иницијализовани пре позива функције.

У овом случају се у функцији не прави копија променљиве `x` већ се ради са променљивом прослеђеном у позиву (имена променљивих не морају бити једнака), тако да функција даје жељени ефекат и програм исписује да је почетна вредност променљиве `x` нула, а да је нова вредност два.

Једна од честих употреба референци у функцијама јесте за потребе размене вредности аргумената функције. На пример, можемо дефинисати функцију која размењује вредност две задате променљиве.

```
void razmeni(int& a, int& b) {
    int tmp = a;
    a = b;
    b = tmp;
}
```

У језику C++, размена се може извршити библиотечком функцијом `swap` декларисаном у заглављу `<algorithm>`.

Задатак: Цена хлеба

Хлеб је прво поскупео 10%, па је затим појефтинио 10%. Ако је позната почетна цена хлеба, напиши програм који одређује цену након тих промена.

Улаз: Са стандардног улаза се учитава почетна цена хлеба (реалан број заокружен на две децимале).

Издаз: На стандардни издаз исписати крајњу цену хлеба (реалан број заокружен на две децимале).

Пример

Улаз	Издаз
35.50	35.15

Решење

Једно решење у складу са свим претходним у овој збирци било би да се уведу три променљиве (једна за почетну цену, једна за цену после поскупљења и једна за цену после појевтињења).

```
double pocetna_cena;
cin >> pocetna_cena;
double cena_posle_poskupljenja = 1.1 * pocetna_cena;
double cena_posle_pojeftinjenja = 0.9 * cena_posle_poskupljenja;
cout << fixed << showpoint << setprecision(2)
    << cena_posle_pojeftinjenja << endl;
```

Задатак можемо да решимо и коришћењем само једне променљиве за цену, мењајући вредност те променљиве током извршавања програма.

```
double cena;
cin >> cena;
cena = 1.1 * cena;
cena = 0.9 * cena;
cout << fixed << showpoint << setprecision(2) << cena << endl;
```

Задатак: Сутрашњи датум

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види текстови задатка.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Једно могуће решење је да се мењају вредности променљивих које репрезентују данашњи дан, чиме се можда мало штеди меморија (мада је то занемариво) и добија мало краћи програмски код, али се губи информација о данашњем дану након одређивања сутрашњег, што је лоше.

```
// učitavamo datum
int dan, mesec, godina;
cin >> dan >> mesec >> godina;

// uvecavamo dan za jedan
dan++;
if (dan > brojDanaUMesecu(mesec, godina)) {
    // taj dan ne postoji u ovom mesecu, pa prelazimo na prvi dan
    // narednog meseca
    dan = 1;
    mesec++;
    if (mesec > 12) {
        // taj mesec ne postoji, pa prelazimo na januar naredne godinu
        mesec = 1;
        godina++;
    }
}

// ispisujemo sutrasnji datum
cout << dan << "." << mesec << "." << godina << "." << endl;
```

Задатак: Подела јабука

Аца је добио n јабука и на једнаке делове (по трећину) поделио браћи Бранку, Цанету, и Душану. Преостале јабуке појео је сам. Бранко је своје јабуке поделио браћи (Аци, Цанету и Душану), а што је преостало појео је сам. Тако је затим поступио и Цане, а онда и Душан. Колико јабука после овакве поделе има свако од њих (занемарујући оне које су поједене).

Улаз: Број јабука n које је добио Аца.

Излаз: Број јабука које на крају има Аца (први ред), Бранко (други ред), Цане (трећи ред) и Душан (четврти ред).

Пример

Улаз	Изназ
100	44
	33
	19
	0

Објашњење

Аца је добио 100 јабука. По 33 је дао Бранку, Цанету и Душану, а једну је појео сам, тако да су након прве поделе Аца, Бранко, Цане и Душан имали редом 0, 33, 33, 33 јабуке. Након тога, Бранко је поделио браћи по 11 јабука и ништа му није остало, тако да је број јабука након друге поделе 11, 0, 44, 44. Након тога, Цане је делио јабуке тако што је браћи дао по 14 јабука, а две је сам појео, тако да је након треће поделе стање 25, 14, 0, 58. На крају, Душан је браћи дао по 19 јабука, а једну преосталу је сам појео. Зато је на крају стање било 44, 33, 19, 0.

Решење

Основни корак поделе (који се понавља 4 пута) је да нека особа која пре тог корака има x јабука, даје по трећину својих јабука особама које имају y , z и w јабука. Након тог корака поделе особе чија је количина јабука била y , z и w увећавају своју количину јабука за целобројни количник броја x са 3 тј. $x \text{ div } 3$, док особи чија је количина јабука била x и која дели јабуке остаје 0 (што је нова вредност за x), јер све јабуке или подели или поједе. Увећавање променљиве x за вредност y се може постићи или доделом $x = x + y$ или, мало краће, оператором $+=$ тј. наредбом $x += y$.

У првом кораку поделе особа x је Аца, док су остале три особе (y , z и w) Бранко, Цане и Душан, у другом је особа x Бранко, док су остале три особе Аца, Цане и Душан и тако даље.

На почетку Аца има дати број јабука n , док сви остали имају 0 јабука.

Могуће решење је да програм организујемо као низ од 16 наредби које одговарају сваком кораку њихове поделе.

```
// Аца у почетку добија све јабуке, а остали немају ништа
int Аца,
    Бранко = 0,
    Цане = 0,
    Душан = 0;
cin >> Аца;

// Аца даје по трећину Бранку, Цанету и Душану - остатак поједе;
Бранко += Аца / 3;
Цане += Аца / 3;
Душан += Аца / 3;
Аца = 0;

// Бранко даје по трећину Аци, Цанету и Душану - остатак поједе;
Аца += Бранко / 3;
Цане += Бранко / 3;
Душан += Бранко / 3;
Бранко = 0;

// Цане даје по трећину Аци, Бранку и Душану - остатак поједе;
Аца += Цане / 3;
Бранко += Цане / 3;
Душан += Цане / 3;
Цане = 0;

// Душан даје по трећину Аци, Бранку и Цанету - остатак поједе.
Аца += Душан / 3;
Бранко += Душан / 3;
Цане += Душан / 3;
```



```
Dusan = 0;
```

```
// ispisujemo preostali broj jabuka za svakoga
cout << Aca << endl << Branko << endl << Cane << endl << Dusan << endl;
```

Ако направимо функцију која реализује један корак поделе, ту функцију можемо позвати четири пута и то тако што у првом позиву параметар x везујемо за Ацин број јабука, у другом за Бранков, у трећем за Цанетов, а у четвртном за Душанов, док се остали параметри (y , z и w) везују за број јабука остале браће.

Параметри те функције треба да буду улазно-излазни (преко њих се преносе почетне улазне вредности у функцију, али се и враћају резултујуће вредности након поделе). Да би се реализовали улазно-излазни параметри, потребно је да се пренос врши преко референци које се у језику C++ обележавају тако што се непосредно након типа параметра у декларацији функције наведе симбол &.

```
void podela(int &x, int &y, int &z, int &w) {
    y += x / 3; // Osoba x osobama y, z i w
    z += x / 3; // daje po trecinu svojih jabuka,
    w += x / 3; // a ono sto ostane pojede sam.
    x = 0;
}

int main() {
    int Aca,
        Branko = 0,
        Cane = 0,
        Dusan = 0;
    cin >> Aca;
    // Aca daje po trecinu Branku, Canetu i Dusanu - ostatak pojede
    podela(Aca, Branko, Cane, Dusan);
    // Branko daje po trecinu Aci, Canetu i Dusanu - ostatak pojede
    podela(Branko, Aca, Cane, Dusan);
    // Cane daje po trecinu Aci, Branku i Dusanu - ostatak pojede
    podela(Cane, Aca, Branko, Dusan);
    // Dusan daje po trecinu Aci, Branku i Canetu - ostatak pojede
    podela(Dusan, Aca, Branko, Cane);
    cout << Aca << endl << Branko << endl << Cane << endl << Dusan << endl;
    return 0;
}
```

Задатак: Јучерашњи датум

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текстови задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

3.1.2 Основне статистике (збир, производ, просек, минимум, максимум)

Најчешће коришћени итеративни алгоритми служе за израчунавање статистика бројевних серија. На пример, једну такву серију могу чинити бројеви 1, 2, 3, 4 и 5. Најзначајније статистике су:

- збир елемената (збир бројева из примера је 15),
- производ елемената (производ бројева из примера је 120),
- минимум тј. најмањи од свих елемената (минимум бројева из примера је 1),
- максимум тј. највећи од свих елемената (максимум бројева из примера је 5).

Одређивање збира и производа

На примеру четворчлане серије елемената, прикажимо итеративне алгоритме за одређивање збира и производа. Збир четири броја се најједноставније израчунава изразом

```
broj1 + broj2 + broj3 + broj4
```

што је због леве асоцијативности сабирања (у програмирању) идентично изразу

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

```
((broj1 + broj2) + broj3) + broj4
```

Други начин да се израчунавање збира реализује је да се уведе помоћна променљива `zbir` која би се увећавала за текући елемент.

```
zbir = broj1;
zbir = zbir + broj2;
zbir = zbir + broj3;
...
```

За разлику од полазног решења у ком се збир израчунава само помоћу једног израза, ово решење је *итеративно*, и резултат се добија узастопном применом корака истог облика. Истакнимо још једном врло важну особину итеративних алгоритама, а то је да се током њиховог извршавања мења вредност неке променљиве (у овом случају то је променљива `zbir`).

Уместо да збир иницијализујемо првим елементом, можемо га иницијализовати нулом, а затим увећавати за један по један елемент.

```
zbir = 0;
zbir = zbir + broj1;
zbir = zbir + broj2;
...
```

Производ се одређује по веома сличном принципу, осим што уместо иницијализације нулом користимо иницијализацију јединицом (као неутралним елементом за множење) и што уместо сабирања користимо множење.

```
proizvod = 1;
proizvod = proizvod * broj1;
proizvod = proizvod * broj2;
...
```

У задацима је често потребно израчунати *просек* тј. *аритметичку средину* серије елемената, која је једнака количнику збира и броја елемената. Приликом имплементације потребно је једино обратити пажњу да се пре дељења осигура да је било збир, било број елемената серије реалан број, да не би дошло до целобројног дељења.

Одређивање минимума и максимума

Потпуно аналогно израчунавању збира неколико бројева, може се израчунати минимум, једино што се уместо операције сабирања два броја користи операција одређивања минимума два броја. У сваком кораку се рачуна минимум минимума свих претходних бројева и текућег броја.

Реално је претпоставити да на располагању имамо функцију којом се израчунава минимум два броја. Таква функција је обично део стандардне библиотеке, а када није, једноставно се може дефинисати. У језику C++ минимум два броја може се израчунати библиотечком функцијом `min` која је декларисана у заглављу `<algorithm>`. Минимум четири броја онда можемо одредити наредним изразом.

```
min(min(min(broj1, broj2), broj3), broj4)
```

Максимум израчунавамо по потпуно истом принципу, једино што уместо минимума два броја у сваком кораку користимо максимум два броја.

И у случају налажења минимума уместо једног израза можемо дефинисати и итеративни алгоритам.

```
minimum = broj1;
minimum = min(minimum, broj2);
minimum = min(minimum, broj3);
...
```

Ако би се уместо функције за рачунање минимума два броја употребило гранање (овај пут изражено условним изразом), дошло би се до следећег кода.

```
minimum = broj1;
minimum = broj2 < minimum ? broj2 : minimum;
minimum = broj3 < minimum ? broj3 : minimum;
...
```

Ако би се уместо условног израза користила наредба гранања, добио би се следећи облик кода.

```
minimum = broj1;
if (broj2 < minimum)
    minimum = broj2;
else
    minimum = minimum;
if (broj3 < minimum)
    minimum = broj3;
else
    minimum = minimum;
...
```

Јасно је да гране `else` немају никакав ефекат и могу се изоставити. Тиме добијамо уобичајени алгоритам одређивања минимума.

```
minimum = broj1;
if (broj2 < minimum)
    minimum = broj2;
if (broj3 < minimum)
    minimum = broj3;
...
```

Дакле, променљиву `minimum` иницијализујемо вредношћу првог броја, а затим је редом поредимо са другим, трећим, четвртим и петим бројем и када год је неки од тих бројева мањи од дотадашње вредности минимума тј. вредности променљиве `minimum`, ажуриромо вредност те променљиве и постављамо је на број за који смо открили да је мањи од ње.

Можемо и мало прецизније да анализирамо претходни код и да докажемо његову коректност. У првом кораку вредност променљиве `minimum` биће минимум једночланог скупа `{broj1}`, у другом биће минимум двочланог скупа `{broj1, broj2}`, након трећег биће минимум скупа `{broj1, broj2, broj3}` и тако даље. Такав услов, дакле, важи у сваком кораку нашег програма и назива се *инваријанца*. Ако је, на пример, вредност променљиве `minimum` вредност минимума скупа `{broj1, broj2, broj3}` и ако је `broj4` мањи од вредности те променљиве, тада је `broj4` мањи од најмање вредности у скупу `{broj1, broj2, broj3}`, што значи да је мањи од свих вредности тог скупа и да је минимум скупа `{broj1, broj2, broj3, broj4}` управо `broj4`. Пошто се вредност променљиве `minimum` ажурира на вредност `broj4` одржава се услов да је вредност променљиве `minimum` управо минимум скупа `{broj1, broj2, broj3, broj4}`, тј. инваријанта је одржана. Са друге стране, ако услов не важи, то значи да је најмањи од бројева из скупа `{broj1, broj2, broj3}` мањи или једнак вредности `broj4` и то је уједно минимум скупа `{broj1, broj2, broj3, broj4}`. Пошто се вредност променљиве `minimum` не мења, инваријанта је опет одржана.

Као што се збир може иницијализовати нулом, а производ јединицом, што су неутралне вредности за сабирање тј. множење, тако се и минимум може иницијализовати вредношћу $+\infty$, а максимум вредношћу $-\infty$. Ако се ради са целобројним типом, ове вредности није могуће представити тако да се уместо њих могу употребити вредности `numeric_limits<int>::max()` и `numeric_limits<int>::min()` (највећа и најмања вредност које се могу представити типом `int`).

Рецимо и да се понављање кода може избећи ако би се употребила петља, што ћемо примењивати када будемо радили са дужим серијама елемената. Сви принципи алгоритма су исти и зато рад са малим серијама бројева представља одличан увод за рад са дужим серијама и петљама.

Низови и библиотечке функције

Уместо ручне имплементације одређивања минимума четири променљиве, можемо употребити низ и библиотечке функције за одређивање максимума низа. О разним врстама низова биће речи у поглављу о низовима, а до тада ћемо разматрати само низове који садрже мали број елемената и који су иницијализовани директним набрајањем њихових елемената.

У језику C++, почевши од стандарда C++11 уведена је тзв. листа иницијализатора (енгл. *initializer list*) која омогућава да се неким функцијама проследи анониман низ сачињен од неколико елемената наведених у витичастим заградама. За наш задатак важна је функција `min` која може да прими такву листу. Тако се изразом `min({broj1, broj2, broj3, broj4})` може веома једноставно одредити најмањи од четири дата броја.

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

Друга могућност је да се бројеви сместе у класичан низ или вектор и да се употреби библиотечка функција `min_element` (више речи о овоме биће у поглављу о низовима).

Задатак: Најлошији контролни

Ученици током године раде 5 контролних задатака. Наставник је обећао да ће приликом закључивања оцена сваком ученику занемарити најслабију оцену (ако постоји више таквих, занемариће само једну). Напиши програм који учитава 5 оцена једног ученика и исписује просечну оцену када се занемари најслабије урађени контролни.

Улаз: У свакој од пет линија стандардног улаза налази се по једна оцена (цео број између 1 и 5) коју је ученик добио.

Изназ: Просечна оцена без најлошијег контролног, заокружена на две децимале.

Пример

Улаз	Изназ
5	4.50
4	
4	
5	
3	

Решење

Одређивање збира и минимума

Просек свих оцена се може добити тако што се збир свих оцена подели бројем свих оцена. У нашем случају најлошија оцена не улази у просек и зато је потребно да израчунамо збир свих оцена без најлошије. Најједноставнији начин да то урадимо је да израчунамо збир свих пет оцена, а затим да од њега одуземо најмању од тих пет оцена. Дакле, након што учитамо 5 оцена, израчунаћемо њихов збир и њихов минимум, одузети минимум од збира и поделити са 4 (један начин да обезбедимо да дељење буде реално је да употребимо константу 4.0).

```
// Funkcija koja izracunava najmanji od 5 datih brojeva
int Minimum5Brojeva(int broj1, int broj2, int broj3,
                    int broj4, int broj5) {
    return min(min(min(broj1, broj2), broj3), broj4), broj5);
}

int main() {
    // Ulazni podaci i njihovo učitavanje
    int ocena1, ocena2, ocena3, ocena4, ocena5;
    cin >> ocena1 >> ocena2 >> ocena3 >> ocena4 >> ocena5;

    // Izracunavanje trazenog proseka
    int zbirSvihOcena = ocena1 + ocena2 + ocena3 + ocena4 + ocena5;
    int najlosijaOcena = Minimum5Brojeva(ocena1, ocena2, ocena3,
                                         ocena4, ocena5);
    int zbirOcenaBezNajlosije = zbirSvihOcena - najlosijaOcena;
    double prosekOcenaBezNajlosije = zbirOcenaBezNajlosije / 4.0;
    // broj kojim se deli mora biti realan jer bi inace bilo primenjeno
    // celobrojno deljenje

    // Prikaz rezultata
    cout << fixed << showpoint << setprecision(2)
         << prosekOcenaBezNajlosije << endl;
    return 0;
}
```

Збир и минимум можемо одредити и итеративним алгоритмима.

```
// Funkcija koja izracunava zbir 5 datih brojeva
```

```

int zbir5Brojeva(int broj1, int broj2, int broj3,
                int broj4, int broj5) {
    int zbir = broj1;
    zbir = zbir + broj2;
    zbir = zbir + broj3;
    zbir = zbir + broj4;
    zbir = zbir + broj5;
    return zbir;
}

// Funkcija koja izracunava najmanji od 5 datih brojeva
int minimum5Brojeva(int broj1, int broj2, int broj3,
                    int broj4, int broj5) {
    int minimum = broj1;
    minimum = min(minimum, broj2);
    minimum = min(minimum, broj3);
    minimum = min(minimum, broj4);
    minimum = min(minimum, broj5);
    return minimum;
}

int main() {
    // Ulazni podaci i njihovo ucitavanje
    int ocena1, ocena2, ocena3, ocena4, ocena5;
    cin >> ocena1 >> ocena2 >> ocena3 >> ocena4 >> ocena5;

    // Izracunavanje trazene proseka
    int zbirSvihOcena = zbir5Brojeva(ocena1, ocena2, ocena3,
                                    ocena4, ocena5);
    int najlosijaOcena = minimum5Brojeva(ocena1, ocena2, ocena3,
                                        ocena4, ocena5);
    int zbirOcenaBezNajlosije = zbirSvihOcena - najlosijaOcena;
    double prosekOcenaBezNajlosije = zbirOcenaBezNajlosije / 4.0;
    // broj kojim se deli mora biti realan jer bi inace bilo primenjeno
    // celobrojno deljenje

    // Prikaz rezultata
    cout << fixed << showpoint << setprecision(2)
          << prosekOcenaBezNajlosije << endl;
    return 0;
}

```

Итеративни алгоритам за одређивање минимума може бити заснован на гранању.

```

// Funkcija koja izracunava najmanji od 5 datih brojeva
int Minimum5Brojeva(int broj1, int broj2, int broj3,
                    int broj4, int broj5) {
    int minimum = broj1;
    if (broj2 < minimum)
        minimum = broj2;
    if (broj3 < minimum)
        minimum = broj3;
    if (broj4 < minimum)
        minimum = broj4;
    if (broj5 < minimum)
        minimum = broj5;
    return minimum;
}

```

Низови и библиотечке функције

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

Задатак је могуће решити и уз помоћ низова и библиотечких функција за сабирање и одређивање минимума.

```
// Ulazni podaci i njihovo ucitavanje
int ocena1, ocena2, ocena3, ocena4, ocena5;
cin >> ocena1 >> ocena2 >> ocena3 >> ocena4 >> ocena5;

// Izracunavanje trazene proseka
int zbirSvihOcena = ocena1 + ocena2 + ocena3 + ocena4 + ocena5;
int najlosijaOcena = min({ocena1, ocena2, ocena3, ocena4, ocena5});
int zbirOcenaBezNajlosije = zbirSvihOcena - najlosijaOcena;
double prosekOcenaBezNajlosije = zbirOcenaBezNajlosije / 4.0;
// broj kojim se deli mora biti realan jer bi inace bilo primenjeno
// celobrojno deljenje

// Prikaz rezultata
cout << fixed << showpoint << setprecision(2)
      << prosekOcenaBezNajlosije << endl;
```

Види груписања решења овој задајци.

Задатак: Најтоплији дан

Дате су дневне температуре редом за сваки дан једне седмице. Написати програм којим се одређује редни број најтоплијег дана те седмице (ако их има више исписати први). Дани у седмици су нумерисани бројевима од 1 до 7, почев од понедељка.

Улаз: Са стандардног улаза уноси се 7 целих бројева из интервала $[-50, 50]$. Сваки број је у посебном реду. Бројеви редом представљају температуре измерене у понедељак, уторак, среду, четвртак, петак, суботу и недељу.

Излаз: На стандардном излазу исписати тражени редни број најтоплијег дана.

Пример

Улаз	Излаз
27	2
32	
28	
27	
32	
31	
29	

Решење

Алгоритам одређивања позиције максимума

У овом задатку тражи се позиција првог појављивања максимума у оквиру серије целих бројева. Дакле, потребно је одредити прву максималну вредност у оквиру серије и запамтити њену позицију.

Један од начина да се одреди позиција максимума неке (непразне) серије је да се мало прилагоди алгоритам одређивања вредности максимума који смо приказали у задатку **Најлошији контролни**. Поред вредности максимума одржава се и позиција на којој је та вредност први пут пронађена. Вредност максимума се иницијализује првим елементом серије, а позиција се иницијализује јединицом. Затим се један по један елемент серије учитава и пореди са текућом вредношћу максимума па се, ако је то потребно, ажурирају вредности максимума и његове позиције. Редни број текућег елемента у серији пратимо променљивом коју увећавамо за 1 сваки пут кад пређемо на нови дан. Ажурирање максимума вршимо само ако је текућа вредност строго већа од досадашњег максимума, чиме одређујемо прво појављивање максимума. Да бисмо одредили његово последње појављивање, ажурирање максимума би требало да вршимо када год се појави елемент који је већи или једнак од текућег максимума.

```
int t, rbr; // temperatura i redni broj tekuceg dana
int maxT, rbrMax; // temperatura i redni broj najtoplјeg dana

cin >> t; rbr = 1; // ponedeljak
```

```

// pretpostavljamo da je prvi dan najtopliji
maxT = t; rbrMax = 1;

cin >> t; rbr++; // utorak
// ako je dan topliji od prethodno najtoplijeg, azuriramo
// vrednost i redni broj najtoplijeg dana
if (t > maxT) {
    maxT = t; rbrMax = rbr;
}

cin >> t; rbr++; // sreda
// ako je dan topliji od prethodno najtoplijeg, azuriramo
// vrednost i redni broj najtoplijeg dana
if (t > maxT) {
    maxT = t; rbrMax = rbr;
}
// ...
cout << rbrMax << endl;

```

Види групачија решења овој задатку.

Задатак: Најбољи у две дисциплине

Три такмичара су радила тестове из математике и програмирања. За сваки предмет добили су одређени број поена (цео број од 0 до 50). Такмичари се рангирају по укупном броју поена из оба предмета. Ако два такмичара имају исти број поена, бољи је онај који има више поена из програмирања. Ако су два такмичара остварила идентичан успех, бољи је онај који има мањи редни број (јер су редни бројеви одређени на основу успеха на квалификационом такмичењу).

Потребно је написати програм који одређује победника такмичења.

Улаз: Учитавају се подаци за три такмичара. За сваког такмичара учитава се број поена из математике, а затим број поена из програмирања, сваки у посебном реду.

Излаз: Потребно је исписати редни број победника. Бројање почиње од 1.

Пример

Улаз	Излаз
38	3
42	
43	
40	
40	
43	

Решење

Овај задатак је уопштење задатка **Бољи у две дисциплине**, тако што се не тражи максимум два, већ максимум три такмичара. Решење које ће овде бити приказано се веома лако може уопштити и на већи број такмичара.

Када би се победник одређивао само на основу укупног броја поена онда би овај задатак захтевао примену алгорита одређивања позиције највећег од неколико датих бројева (види задатак **Најтоплији дан**). Међутим, сваки такмичар је представљен уређеним паром бројева (поенима из математике и програмирања) и потребно је одредити позицију највећег елемента међу три уређена пара, при чему је релација поретка међу тим паровима одређена условима задатка (пореди се прво укупан број поена, па онда поени из програмирања). Ово је класичан пример релације поређења узастопном применом неколико критеријума који се примењују један за другим, кажемо лексикографски.

Најелегантније решење је да ту релацију имплементирамо у засебној функцији која прихвата податке за два такмичара (четири броја) и враћа истинитосну (буловску) вредност `true` ако и само ако је први бољи од другог у односу на ову релацију. Када имамо функцију поређења на располагању, позицију максималног елемента одређујемо применом уобичајеног алгорита за то.

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

Различити могући начини имплементације лексикографског поређења баш какво је потребно у овом задатку су описани у задатку **Бољи у две дисциплине**.

Један начин је да поређење реализујемо у функцији која проверава један по један услов и ако открије да је на основу неког услова могуће вратити резултат, помоћу `return` тај резултат враћа.

```
// provera da li je takmicar A bolji od takmicara B
bool bolji(int matA, int progA, int matB, int progB) {
    int ukupnoA = matA + progA;
    int ukupnoB = matB + progB;
    if (ukupnoA > ukupnoB)
        return true;
    if (ukupnoA < ukupnoB)
        return false;
    return progA > progB;
}

int main() {
    // broj poena prvog takmicara iz matematike i programiranja
    int mat1, prog1;
    cin >> mat1 >> prog1;

    // broj poena drugog takmicara
    int mat2, prog2;
    cin >> mat2 >> prog2;

    // broj poena treceg takmicara
    int mat3, prog3;
    cin >> mat3 >> prog3;

    // redni broj pobednika
    int pobednik;
    // broj poena pobednika iz matematike i programiranja
    int matMax, progMax;

    // algoritam za odredjivanje pozicije maksimuma
    // pretpostavljamo da je pobednik prvi takmicar
    pobednik = 1;
    matMax = mat1; progMax = prog1;
    // ako je drugi bolji od dosadasnjeg najboljeg, pobednik je on
    if (bolji(mat2, prog2, matMax, progMax)) {
        pobednik = 2;
        matMax = mat2; progMax = prog2;
    }
    // ako je treci bolji od dosadasnjeg najboljeg, pobednik je on
    if (bolji(mat3, prog3, matMax, progMax)) {
        pobednik = 3;
        matMax = mat3; progMax = prog3;
    }

    cout << pobednik << endl;
    return 0;
}
```

Види групачија решења овој задајци.

Задатак: Најбољи такмичари

На такмичењу у пливању слободним стилем на 100 метара учествовало је 4 такмичара са старним бројевима од 1 до 4. Познати су резултати свих такмичара изражени у секундама. Написати програм који исписује

стартне бројеве свих оних такмичара који су постигли најбољи резултат (најмање време) на том такмичењу.

Улаз: У четири линије стандардног улаза налазе се четири реална броја из интервала $[40, 100]$, сваки број у посебној линији. Бројеви представљају резултате такмичара редом од такмичара са стартним бројем 1 до такмичара са стартним бројем 4.

Излаз: На стандардном излазу исписују се стартни бројеви такмичара који су постигли најбољи резултат. Сваки стартни број је у посебној линији, старни бројеви су приказани од најмањег до највећег.

Пример

Улаз	Излаз
54.7	2
50.3	3
50.3	
58.6	

Решење

У овом задатку се тражи да се одреди минимум задате серије резултата такмичења у пливању и позиције свих елемената серије који имају минималну вредност. Одређивање позиције најмањег елемента разматрали смо у задатку **Најтоплији дан** међутим, за разлику од тог задатка у случају више појављивања минималне вредности овде се захтева проналажење свих позиција, док је тамо било довољно наћи само прву. Решење ове варијанте задатка могуће је урадити на неколико начина.

Провера минималности

Један начин да се то уради је да се за сваки резултат провери да ли он представља минимум серије, тј. да ли је мањи или једнак од осталих резултата. У оваквом решењу услов гранања је сложен, па се лако може десити да се пропусти неки од критеријума. Такође, за велики број такмичара ово решење би било прилично неефикасно.

```
double t1, t2, t3, t4;
// učitavamo vremena takmicara
cin >> t1 >> t2 >> t3 >> t4;

// za svakog takmicara proveravamo da li je bolji od ostalih
if (t1 <= t2 && t1 <= t3 && t1 <= t4)
    cout << 1 << endl;
if (t2 <= t1 && t2 <= t3 && t2 <= t4)
    cout << 2 << endl;
if (t3 <= t1 && t3 <= t2 && t3 <= t4)
    cout << 3 << endl;
if (t4 <= t1 && t4 <= t2 && t4 <= t3)
    cout << 4 << endl;
```

Одређивање минимума и упоређивање са њим

Најједноставнији и уједно најефикаснији начин је да се прво одреди минимум серије, а да се затим један по један резултат такмичара пореди са том минималном вредношћу, при чему се полази од првог такмичара. У случају да је резултат такмичара једнак са минимумом, стартни број тог такмичара се приказује на стандардном излазу. Овим се заправо комбинује поступак одређивања најмање вредности серије који смо детаљно описали у задатку **Најлошији контролни** са накнадним поступком филтрирања серије који је описан у задатку **Јутарње температуре**.

```
// učitavamo vremena takmicara
double t1, t2, t3, t4;
cin >> t1 >> t2 >> t3 >> t4;

// pronalazimo najbolje vreme
double minT = min({t1, t2, t3, t4});

// za svakog takmicara proveravamo da li je postigao to najmanje vreme
if (t1 == minT)
    cout << 1 << endl;
```

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

```
if (t2 == minT)
    cout << 2 << endl;
if (t3 == minT)
    cout << 3 << endl;
if (t4 == minT)
    cout << 4 << endl;
```

Види групација решења овој задатка.

Задатак: Најјефтинији за динар

У једној продавници је у току акција у којој се купцима који купе три производа нуди да најјефтинији од та три производа добију за један динар. Напиши програм који одређује снижену цену три производа.

Улаз: Са стандардног улаза уносе се три цела броја из интервала од 50 до 5000 који представљају цене у динарима за три купљена производа.

Излаз: На стандардни излаз исписати један цео број који представља укупну снижену цену та три производа.

Пример

Улаз	Излаз
2499	6099
3599	
899	

Задатак: Просек скијаша

На такмичењу у скијашким скоковима поред удаљености коју скакач прескочи оцењује се и стил скакача. Пет судија оцењују стил скакача оценама од 0 до 20. Затим се од свих добијених оцена избришу једна најмања и једна највећа. Коначна оцена стила скакача рачуна се као просек преосталих оцена. Напиши програм којим се на основу оцена 5 судија одређује коначна оцена стила скакача.

Улаз: У свакој од пет линија стандардног улаза налази се по једна оцена стила скакача (цео број између 0 и 20) коју је скијаш добио од пет судија.

Излаз: Коначна оцена стила скакача приказана на две децимале.

Пример

Улаз	Излаз
3	3.00
5	
0	
1	
5	

Задатак: Врста троугла на основу углова

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види текснi задатка.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Види групација решења овој задатка.

Задатак: Растојање тачка правоугаоник

Деца се играју јурке у школском дворишту правоугаоног облика и око њега. Правила игре су таква да је ученик безбедан када се ухвати за ограду. Напиши програм који одређује колико је најмање растојање које Тока треба да претрчи да би био безбедан (он се у почетку може налазити и у дворишту и ван њега).

Улаз: Са стандардног улаза читава се 6 реалних бројева (сваки у посебном реду):

- x_0, y_0 ($0 \leq x_0, y_0 < 1000$) - координате доњег левог темена правоугаоника који представља школско двориште у координатном систему постављеном тако да су ивице правоугаоника паралелне координатним осама.
- x_1, y_1 ($x_0 < x_1 \leq 1000, y_0 < y_1 \leq 1000$) координате горњег десног темена правоугаоника.

- x, y ($0 \leq x_0, y_0 \leq 1000$) - координате тачке на којој се налази Тока.

Излаз: На стандардни излаз исписати један реалан број - најкраће растојање Токе до ограде у метрима (допуштена је толеранција је један центиметар, тј. резултат исписати заокружен на две децимале).

Пример 1

Улаз	Излаз
100	50.00
100	
200	
200	
50	
170	

Пример 2

Улаз	Излаз
100	111.80
100	
200	
200	
50	
300	

Задатак: Позиција највећег угла троугла

Дата су два угла разностраничног троугла у степенима, минутима и секундама. Одредити који од углова троугла је највећи, први по редоследу уноса, други по редоследу уноса или трећи, одређен у програму.

Улаз: У свакој од шест линија стандардног улаза налази се по један цео број. Бројеви редом представљају степене, минуте (мање од 60) и секунде (мање од 60), прво једног, па другог угла троугла. Збир дата два угла је мањи од 180 степени.

Излаз: У једној линији стандардног излаза исписати број 1, 2 или 3.

Пример

Улаз	Излаз
75	3
30	
14	
23	
15	
45	

Задатак: Најближи просеку

Написати програм који за четири дата броја одређује који је од та четири броја најближи аритметичкој средини датих бројева (ако су два броја једнако близу исписати први).

Улаз: У четири линије стандардног улаза налазе се четири реална броја.

Излаз: У једној линији стандардног излаза приказати један од унетих реалних бројева, на две децимале, најближи њиховој аритметичкој средини.

Пример

Улаз	Излаз
24.3	22.70
20.2	
23.5	
22.7	

Задатак: Први и други максимум

Написати програм којим се одређују највећа два различита броја од пет датих целих бројева.

Улаз: На стандардном улазу налазе се 5 целих бројева, сваки у посебној линији.

Излаз: Прва линија стандардног излаза садржи највећи број од датих 5 бројева. Друга линија стандардног излаза садржи други по величини цео број од датих 5 бројева. Ако су сви унети бројеви једнаки друга линија треба да садржи само знак '-'.
 -

Пример 1

Улаз	Израз
2	7
7	3
-4	
7	
3	

Пример 2

Улаз	Израз
12	12
12	-
12	
12	
12	

Задатак: Аутомобил у вођству

Три аутомобила крећу са стартне позиције у тренуцима $0 \leq t_1 \leq t_2 \leq t_3$ константним брзинама v_1, v_2 и v_3 . Приказати стартне бројеве аутомобила који су на водећој позицији у тренутку t ($t \geq 0$). Ако ни један аутомобил још није кренуо, сва три су у водећој позицији.

Улаз: Са стандардног улаза учитава се седам целих бројева, сваки у засебној линији $t_1, v_1, t_2, v_2, t_3, v_3, t$ где t_1, t_2, t_3, t представљају времена у секундама, а v_1, v_2, v_3 брзине у метрима по секунди.

Израз: На стандардни излаз исписати један или више целих бројева (сваки у посебном реду) који представљају редне бројеве аутомобила (1, 2 или 3) који су на водећој позицији. Ако је више аутомобила истовремено у вођству, њихове редне бројеве исписати у растућем редоследу.

Пример

Улаз	Израз
10	1
5	2
5	
4	
0	
2	
30	

3.1.3 Филтрирање

Филтрирање подразумева да се из серије елемената издвоје они који задовољавају неки задати услов. Често нас занима да одредимо статистике само таквих елемената (њихов број, збир, просек итд.).

Задатак: Јутарње температуре

Дате су јутарње температуре за 4 дана T_1, T_2, T_3 и T_4 . Одредити број дана од та 4 дана када је јутарња температура била испод просечне температуре за та 4 дана.

Улаз: Са стандардног улаза се учитавају четири реална броја (сваки је у посебној линији) који представљају јутарње температуре за четири дана.

Израз: У првој линији стандардног улаза исписује се просечна температура за та четири дана, заокружена на две децимале.

У другој линији стандардног излаза исписује се број дана када је температура била испод просечне.

Пример

Улаз	Израз
27.3	23.20
20.2	2
23.5	
21.8	

Решење

Потребно је прво одредити аритметичку средину четири унете температуре (просечну температуру) као количник њиховог збира и броја 4 (тј. реалног броја 4.0), слично као, на пример, у задатку **Најлошији контролни**. Бројач дана чија је температура била испод просечне треба иницијално поставити на 0. Затим, за сваку температуру треба проверити да ли је нижа од просечне, па ако јесте бројач треба увећати за 1.

Приказани алгоритам је класичан пример алгоритма израчунавање статистике *филтриране серије* тј. алгоритма у којем се траже да се обраде (у овом случају преброје) елементи који задовољавају неки дати услов.

```

// unosimo cetiri temperature
double T1, T2, T3, T4;
cin >> T1 >> T2 >> T3 >> T4;

// odredjujemo njihovu aritmeticku sredinu
double prosekT = (T1 + T2 + T3 + T4) / 4.0;

// brojimo one dane u kojima je temperatura ispod proseka
int brojac = 0;
if (T1 < prosekT) brojac++;
if (T2 < prosekT) brojac++;
if (T3 < prosekT) brojac++;
if (T4 < prosekT) brojac++;

cout << fixed << showpoint << setprecision(2) << prosekT << endl;
cout << brojac << endl;

```

Задатак: Квалификациони праг

Ако се знају поени 4 такмичара и квалификациони праг на такмичењу, одредити колико такмичара је освојило довољно поена и квалификовало се у следећи круг такмичења, а колики је просек поена оних који се нису квалификовали.

Улаз: У пет линија стандардног улаза налазе се пет целих бројева из интервала $[0, 100]$, сваки број у посебној линији. Прва четири броја представљају освојене поене такмичара, а пети број је квалификациони праг.

Излаз: На стандардном излазу исписује се:

- у првој линији број такмичара који су се квалификовали,
- у другој линији просек поена (на две децимале) за групу такмичара која се није квалификовала, а у случају да су се сви квалификовали у другој линији исписати знак ‘-’.

Пример

Улаз	Излаз
0	2
30	1.00
2	
99	
30	

Решење

Потребно је одредити број такмичара који су се квалификовали и просек поена такмичара који се нису квалификовали. Просек поена такмичара који се нису квалификовали добијамо као количник збира њихових поена и броја таквих такмичара. Иако је потребно да знамо и број такмичара који су се квалификовали, и број оних који нису, нема потребе да за то уводимо два бројача, јер ако уведемо само бројач за такмичаре који се нису квалификовали, онда број такмичара који су се квалификовали можемо одредити као разлику укупног броја такмичара (у нашем примеру 4) и броја такмичара који се нису квалификовали (слично би се радило када би се увео бројач за такмичаре који су се квалификовали). У нашем решењу одређујемо број такмичара који се нису квалификовали и збир њихових поена. Бројач такмичара који се нису квалификовали и збир њихових поена поставимо на 0. Затим, за сваког такмичара проверимо да ли је његов број поена мањи од квалификационог прага, па ако јесте бројач увећамо за 1, а збир за поене такмичара. Пошто и за бројач и за збир користимо целобројне променљиве, приликом одређивања просека такмичара који се нису квалификовали потребно је бар једну од њих конвертовати у реалну (или оператором експлицитне конверзије, или множењем са 1.0). Напоменимо да се у овом задатку, слично као у задатку **Јутарње температуре** врши израчунавање статистика филтриране серије (тражи се просек, тј. збир и број свих елемената серије који задовољавају дати услов).

```

// poeni takmicara
int p1, p2, p3, p4;
cin >> p1 >> p2 >> p3 >> p4;
// kvalifikacioni prag

```

```

int prag;
cin >> prag;

// izracunavamo zbir poena i broj onih koji se nisu kvalifikovali
int br = 0, suma = 0;
if (p1 < prag) {
    br++;
    suma += p1;
}
if (p2 < prag) {
    br++;
    suma += p2;
}
// ...

// ispisujemo broj onih koji su se kvalifikovali
cout << 4 - br << endl;

// ako postoji bar neko ko se nije kvalifikovao
if (br > 0) {
    // izracunavamo i ispisujemo prosek svih onih koji se nisu
    // kvalifikovali (posto su i suma i br celobrojne, potrebna je
    // eksplicitna konverzija)
    double prosek = (double)suma / (double)br;
    cout << fixed << setprecision(2) << showpoint << prosek << endl;
}
else
    cout << '-' << endl;

```

Задатак: Претицање

Три аутомобила стартних бројева 1, 2, 3 крећу са исте стартне позиције редом у тренуцима t_1, t_2, t_3 ($0 \leq t_1 < t_2 < t_3$), и крећу се константним брзинама v_1, v_2, v_3 . Написати програм којим одређујемо тренутак кад се деси последње претицање, или ако нема претицања исписати текст *нема*.

Улаз: Са стандардног улаза учитава се 6 реалних бројева. Прва линија садржи бројеве t_1, v_1 , друга бројеве t_2, v_2 , а трећа бројеве t_3, v_3 .

Издаз: На стандардни издаз исписати један реалан број који представља време последњег претицања, заокружен на две децимале или текст *нема* ако претицања није било.

Пример 1

Улаз	Издаз
1.6 20	нема
2 12	
7.4 2.1	

Пример 2

Улаз	Издаз
6.3 9	47.57
15 18	
19 13	

Решење

Ако је прво возило кренуло у тренутку t_a брзином v_a , а друго у каснијем тренутку t_b брзином v_b , тада ће друго возило престићи прво ако и само ако је од њега брже тј. ако је $v_b > v_a$. Пошто крећу са истог старта, претицање настаје у оном тренутку када оба возила пређу исти пут. Пошто се креће равномерно, пут који у неком тренутку t пређе прво возило је $s_a = (t - t_a) \cdot v_a$, док друго возило прелази пут $s_b = (t - t_b) \cdot v_b$ (веза између брзине, времена и пређеног пута описана је у задатку [Путовање](#)). Пошто ти путеви треба да буду једнаки важи да је $(t - t_a) \cdot v_a = (t - t_b) \cdot v_b$ одакле је

$$t = \frac{v_b \cdot t_b - v_a \cdot t_a}{v_b - v_a}.$$

Пошто у нашој трци учествују три аутомобила, могућа су три потенцијална претицања. Зато разматрамо могућност да друго возило претекне прво, да треће возило претекне прво и да треће возило претекне друго.

Тако добијамо малу серију бројева које чине времена претицања и којих има између 0 и 3. Ако не постоји ни једно време претицања (ако је свако следеће возило спорије или једнако брзо као претходно) тада треба пријавити да претицања нема, а у супротном треба пронаћи последње време претицања као максимум те серије времена.

У овом задатку, дакле, тражимо максимум филтриране серије, која може бити и празна. Начини да се то уради представљају модификације начина да се одреди максимум тј. минимум серије бројева који су описани у задатку **Најлошији контролни**.

Алгоритам одређивања максимума филтриране серије

Један начин да ово реализујемо је да памтимо последње време претицања до тада и податак о томе да ли је пронађено бар једно претицање. Упоредимо наведене парове возила и да када наиђемо на ново време претицања, време последњег претицања ажурирамо или ако је ово прво пронађено претицање или ако ово претицање наступа после до тада последњег претицања.

```
// времена u коме su tri vozila krenula sa starta
double t1, t2, t3;
// brzine ta tri vozila
double v1, v2, v3;

// ...

// da li je pronadjeno jedno preticanje
bool biloPreticanja = false;
// vremenski trenutak u kom se desilo poslednje preticanje
double vremePoslednjegPreticanja;

// proveravamo da li drugi automobil moze preteci prvi
if (v2 > v1) {
    double vremePreticanja = (v2*t2 - v1*t1) / (v2 - v1);
    if (!biloPreticanja ||
        vremePreticanja > vremePoslednjegPreticanja)
        vremePoslednjegPreticanja = vremePreticanja;
    biloPreticanja = true;
}
// proveravamo da li treci automobil moze preteci prvi
// ...
// proveravamo da li treci automobil moze preteci drugi
// ...

// Ako je bilo preticanja prijavljujemo vreme poslednjeg preticanja
if (biloPreticanja)
    cout << fixed << showpoint << setprecision(2)
         << vremePoslednjegPreticanja << endl;
else
    cout << "nema" << endl;
```

Понављање сличног кода се може избећи или коришћењем петље или издвајањем ажурирања максимума у помоћну функцију (при чему је петља много природније решење). Коришћење логичке променљиве у којој памтимо да ли је до тада детектовано претицање може се избећи ако се вредност времена последњег претицања иницијализује нулом (тима се постиже да када се наиђе на прво претицање дође до ажурирања времена последњег претицања, јер свако претицање наступа сигурно у тренутку $t > 0$, пошто аутомобили не крећу истовремено са старта).

Види грујачија решења овој задајци.

Задатак: Берза

Трговац на берзи је током једне радне недеље сваки дан или остваривао зараду или је губио новац. Од свих дана у којима је нешто зарадио, одредити дан у коме је најмање зарадио и вредност коју је тог дана зарадио (ако је више таквих дана, пријавити први) или пријавити да је свих дана губио новац.

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

Улаз: Са стандардног улаза уноси се 5 целих бројева из интервала $[-10000, 10000]$ (износи које је трговац остварио у понедељак, уторак, среду, четвртак и петак), сваки у посебном реду.

Изназ: Ако је трговац у неком дану остварио зараду (износ је строго већи од нуле), на стандардни излаз у првом реду исписати најмањи износ зараде који је остварен током недеље и у другом реду ознаку дана (PON, UTO, SRE, CET или PET) у коме је остварен тај најмањи профит. Ако ниједан дан није остварена зарада, исписати ред који садржи само карактер -.

Пример 1		Пример 2	
Улаз	Изназ	Улаз	Изназ
3200	1350	-4700	-
-420	CET	-360	
-10		-1000	
1350		-1550	
5670		-3245	

3.1.4 Претрага

Претрага подразумева испитавање да ли у серији постоји неки елемент који задовољава дати услов (на пример, да ли је неки од ученика у одељењу постигао максималан број поена на контролном задатку). Такође, могуће је испитивати да ли сви елементи задовољавају услов, тражити позицију првог или позицију последњег елемента који задовољава или не задовољава дати услов и слично.

Алгоритам претраге се своди на итеративно израчунавање дисјункције или конјункције. Постоји велика сличност између алгоритма претраге и алгоритма израчунавања збира, производа, минимума и максимума серије које смо већ разматрали. На пример, провера да ли је неки од бројева a_1 , a_2 или a_3 негативан се своди на то да се провери да ли је први број негативан, да ли је други број негативан и тако даље. Потребно је дакле израчунати вредност логичког израза

$$(a_1 < 0) \ || \ (a_2 < 0) \ || \ (a_3 < 0)$$

Веома слично као и у случају израчунавања збира, производа или минимума/максимума, резултујућу променљиву постављамо на неутралну вредност (`false` у случају дисјункције тј. `true` у случају конјункције), а онда је у сваком кораку ажурирамо применом одговарајућег оператора.

```
postoji = false;
postoji = postoji || a1 < 0;
postoji = postoji || a2 < 0;
postoji = postoji || a3 < 0;
```

Даље, може се приметити да ако је услов $a_i < 0$ испуњен, онда вредност променљиве `postoji` постаје `true` а ако није испуњен, онда остаје онаква каква је и била раније. На основу тога, претходни код се може написати у следећем облику:

```
postoji = false;
if (a1 < 0) postoji = true;
if (a2 < 0) postoji = true;
if (a3 < 0) postoji = true;
```

Оператор `||` има лењу семантику тј. у изразу $x \ || \ y$ вредност израза y се не израчунава ако израз x има вредност `true`. Зато сваки наредни услов можемо проверавати само ако претходни није задовољен.

То постижемо тиме што уместо независних провера користимо конструкцију `else-if` (илустрована у задатку [Агрегатно стање](#)) и сваки наредни услов проверавамо само ако претходни није испуњен.

```
postoji = false;
if (a1 < 0) postoji = true;
else if (a2 < 0) postoji = true;
else if (a3 < 0) postoji = true;
```

Алгоритам линеарне претраге је нарочито значајан код дужих серија, и о њему ће више бити речи у задатку [Негативан број](#).

Задатак: Чудно радно време

Паја има обичај да да у сред дана напусти радно место и оде негде на кратко. Због тога је објавио необично радно време, које се састоји од 4 интервала, са три паузе (четворократно радно време). Напиши програм који проверава да ли Паја ради у тренутку t . Сматра се да сваки од ових интервала садржи свој почетак, а не садржи свој крај.

Улаз: На стандардном улазу у прва четири реда је задат по један интервал Пајиног радног времена. Сваки интервал је задат сатом и минутом почетка и сатом и минутом краја (цели бројеви s_1, m_1, s_2, m_2 раздвојени размаком, $0 \leq s_i \leq 23, 0 \leq m_i \leq 59$, за $i \in \{1, 2\}$). Интервали су дисјунктни и поређани хронолошки. У петом реду стандардног улаза налазе се цели бројеви S, M раздвојени зарезом, $0 \leq S \leq 23, 0 \leq M \leq 59$, који представљају тренутак t , за који се проверава да ли је у Пајином радном времену.

Излаз: На стандардни излаз написати да ако је t у Пајином радном времену тј. не у супротном.

Пример 1

Улаз	Излаз
9 0 11 30	ne
12 15 14 45	
15 0 17 0	
17 15 19 0	
14 45	

Решење

Да бисмо утврдили да ли је тренутак t у Пајином радном времену, можемо да користимо логичку променљиву *otvoreno*.

На почетку овој променљивој доделимо вредност `false`, а затим за сваки интервал радног времена редом проверавамо да ли садржи тренутак t . Ако је за неки од интервала услов испуњен, променљивој *otvoreno* додељујемо вредност `true`. На тај начин, по завршетку поступка променљива *otvoreno* саопштава да ли тренутак t припада бар једном од дата 4 интервала.

```
// време s:m se ucitava i prevodi u minute
int UcitajVremeUMinutima() {
    int s, m;
    cin >> s >> m;
    return s * 60 + m;
}

int main() {
    // ucitavamo intervale radnog vremena
    int a1 = UcitajVremeUMinutima();
    int b1 = UcitajVremeUMinutima();
    int a2 = UcitajVremeUMinutima();
    int b2 = UcitajVremeUMinutima();
    int a3 = UcitajVremeUMinutima();
    int b3 = UcitajVremeUMinutima();
    int a4 = UcitajVremeUMinutima();
    int b4 = UcitajVremeUMinutima();

    // ucitavamo trenutak dolaska
    int t = UcitajVremeUMinutima();

    // proveravamo da li je dolazak u nekom od intervala radnog vremena
    bool otvoreno = false;
    if (a1 <= t && t < b1)
        otvoreno = true;
    if (a2 <= t && t < b2)
        otvoreno = true;
    if (a3 <= t && t < b3)
```

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

```
    otvoreno = true;
if (a4 <= t && t < b4)
    otvoreno = true;

if (otvoreno)
    cout << "da" << endl;
else
    cout << "ne" << endl;
return 0;
}
```

Задатак: Постоји ли троугао датих дужина страница

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Једна могућност је да се сваки услов проверава посебно и на основу његове провере поставља вредност логичке променљиве. Логичка променљива се на почетку поставља на вредност `true` (претпостављамо да троугао постоји), а затим испитујемо један по један услов и ако је услов нарушен, вредност променљиве постављамо на `false`.

Приметимо да када се установи да неки услов није испуњен, није неопходно вршити проверу даље, тако да је уместо три независне наредбе `if` могуће употребити конструкцију `else-if`.

```
// proveravamo da li postoji trougao sa duzinama stranica a, b i c
bool postojiTrougao = true;
if (a + b <= c)
    postojiTrougao = false;
else if (b + c <= a)
    postojiTrougao = false;
else if (a + c <= b)
    postojiTrougao = false;
```

Проверу је могуће реализовати и у функцији у којој се проверава један по један услов и чим се наиђе на неки који није испуњен, враћа се вредност `false`. (наредбом `return`). На крају функције, ако су сви услови били испуњени враћа се вредност `true`. Пошто се у свакој грани јавља наредба `return` извршавање функције бива прекинуто када се уђе у било коју од грана. Зато није неопходно користити `else` (мада није ни погрешно).

```
// provera da li postoji trougao cije su stranice duzine a, b i c
bool postojiTrougao(double a, double b, double c) {
    if (a + b <= c)
        return false;
    if (b + c <= a)
        return false;
    if (a + c <= b)
        return false;

    return true;
}
```

Задатак: Статус објављен током школе

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Непливачи

Деда Раде жели да упише својих четворо унука непливача у школу пливања. Инструктор му је рекао да је остао само један слободан термин, за који могу да се пријаве само деца виша од 110 сантиметара. Деда Раде

не жели да раздваја унуке, па ће их уписати у школу пливања само ако сви испуњавају услов. Написати програм који одређује да ли деда Раде већ сада може да упише свих четворо унука у школу пливања.

Улаз: Са стандардног улаза се уносе четири цела позитивна броја не већа од 180, сваки у посебном реду – висине деда Радетових унука.

Излаз: На стандардни излаз исписати реч SVI ако је свако од четворо деце више од 110cm, а реч NIKO ако бар једно дете није више од 110 cm.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
131	SVI	131	NIKO
111		110	
128		128	
117		117	

3.1.5 Сортирање и примене

Због својих великих примена **сортирање** је један од најбоље проучених проблема у рачунарству. Задатак сортирања је да се дата серија уреди у неком траженом поретку (на пример, да се бројеви уреду по величини). Сортирање је у *распућеном* поретку ако је сваки наредни елемент строго већи од претходног а у *неопадајућем* поретку ако је сваки наредни елемент већи или једнак од претходног. Аналогно се дефинише сортирање у *опадајућем* и *нераспућеном* поретку.

Сортирање селекцијом

Један од могућих приступа сортирању три променљиве a , b и c јесте тај да прво покушамо да разменама променљивих осигурамо да се најмања вредност налази у променљивој a . За то су нам довољна два поређења и највише две размене. Прво упоређујемо вредности a и b и размењујемо их само ако је b мање од a , чиме осигуравамо да је тренутна вредност a мања или једнака вредности b . Након тога, тренутну вредност променљиве a (то је мања од оригиналних вредности a и b) упоређујемо и са вредношћу c и само ако је c мања од ње, размењујемо их. Тиме је осигурано да тренутна вредност променљиве a буде мања или једнака тренутним вредностима променљивих b и c , при чему ниједна од вредности није изгубљена јер су све време коришћене само размене. На крају, пореде се вредности b и c , и само ако је c мања од вредности b , оне се размењују, чиме се постиже да су све три променљиве сортиране. Дакле овај поступак се може имплементирати на следећи начин (после сваког корака наведен је услов за који сигурно знамо да у том тренутку важи).

```

if (a > b)
    razmeni(a, b);
// a <= b
if (a > c)
    razmeni(a, c);
// a <= b i a <= c
if (b > c)
    razmeni(b, c);
// a <= b i a <= c i b <= c

```

Овај алгоритам сортирања назива се *сортирање избором најмањег елемента* (енгл. *selection sort*).

Сортирање уметањем

Сортирање уметањем заснива се на идеји да се елементи један по један уметају у сортирани префикс низа. Први елемент a сигурно представља сортирани једночлани низ. Да бисмо обезбедили да прва два елемента представљају сортиран низ, потребно је да заменимо a и b ако је b мање од a . Када су a и b исправно сортирани, тада елемент c треба убацити на његово место. Он се прво пореди са елементом b и ако је већи или једнак од њега, три елемента су већ сортирана како треба. У супротном, размењујемо b и c . Након тога, потребно је још проверити да ли је b сада можда поново мањи од a и ако јесте, разменити a и b . Дакле овај поступак се може имплементирати на следећи начин (после сваког корака наведен је услов за који сигурно знамо да у том тренутку важи).

```

if (b < a)
    razmeni(a, b);
// a <= b
if (c < b) {

```

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

```
    razmeni(b, c);  
    // b <= c, a <= c  
    if (b < a)  
        razmeni(a, b);  
}  
// a <= b, b <= c, a <= c
```

Овај алгоритам сортирања назива се *сортирање уметањем* (енгл. *insertion sort*).

Сортирање низа библиотечком функцијом

Најједноставнији начин да се три броја сортирају је да се они сместе у низ и да се примени библиотечка функција за сортирање елемената низа. О низовима и њиховом коришћењу биће више речи у каснијим поглављима ове збирке.

На пример, ако желимо да сортирамо четири бројевне вредности *a*, *b*, *c* и *d*, можемо формирати четворочлани *низ* и иницијализовати га на те четири вредности.

```
int[] niz = {a, b, c, d};
```

Вредности у низу можемо и учитати са стандардног улаза и тада је приликом декларације низа потребно да означимо да ће низ бити четворочлан.

```
int niz[4];  
cin >> niz[0] >> niz[1] >> niz[2] >> niz[3];
```

Након што је низ попуњен вредностима, можемо га сортирати библиотечком функцијом `sort`, на следећи начин.

```
sort(niz, niz+4);
```

Алтернативно, могуће је навести и

```
sort(begin(niz), end(niz));
```

Задатак: Уреди три броја

Милица купује патике. Допала су јој се три пара. Напиши програм који исписује цене та три пара патика од најјефтинијих до најскупљих.

Улаз: Са стандардног улаза уносе се три цела броја из интервала [1000, 10000], сваки у посебном реду.

Излаз: На стандардни излаз исписују се бројеви уређени од најмањег до највећег.

Пример

Улаз	Излаз
2300	1800
1800	2300
2799	2799

Решење

У задатку се тражи да се три броја уреде (сортирају) по величини и то у неоппадајући поредак (тако да ниједан наредни није мањи од претходног - нагласимо да два суседна елемента могу да буду једнака).

Сортирање можемо извршити селекцијом. Само сортирање је пожељно издвојити у засебну функцију (чиме се добија на прегледности програма, а та функција се може користити и у наредним програмима).

У језику C++ за размену две променљиве може се користити библиотечка функција `swap` која је декларисана у заглављу `<algorithm>`. Пренос аргумената у функције које врше размену или сортирање је потребно вршити по референци (то су улазно-излазни параметри тих функција), и параметре приликом декларације функције потребно је означити знаком `&`.

```
// uredjivanje selekcijom  
void sortiraj3(int& a, int& b, int& c) {  
    if (a > b)  
        swap(a, b);  
    if (a > c)
```

```

    swap(a, c);
    if (b > c)
        swap(b, c);
}

int main() {
    // cene tri para patika
    int p1, p2, p3;
    cin >> p1 >> p2 >> p3;
    sortiraj3(p1, p2, p3);
    cout << p1 << endl << p2 << endl << p3 << endl;
    return 0;
}

```

Сортирање можемо извршити и уметањем.

```

// uredjivanje umetanjem
void sortiraj3(int& a, int& b, int& c) {
    if (b < a)
        swap(a, b);
    if (c < b) {
        swap(b, c);
        if (b < a)
            swap(a, b);
    }
}

```

Најједноставнији начин да се три броја сортирају је да се они упамте у низ и да се примени библиотечка функција за сортирање елемената низа.

```

// cene tri para patika
int p[3];
cin >> p[0] >> p[1] >> p[2];
sort(p, p+3);
cout << p[0] << endl << p[1] << endl << p[2] << endl;

```

Задатак: Најлошији контролни

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види текстови задатка.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Сортирање

Задатак можемо решити тако што сортирамо све оцене неоппадајуће и израчунамо просек свих елемената сортираног низа оцена осим првог (то је оцена на најлошије урађеном контролном). Начини сортирања мале серије елемената описани су у задатку [Уреди три броја](#).

Најједноставније је да се бројеви сместе у низ и да се употреби библиотечка функција. У језику С++ за сортирање се користи функција `sort` декларисана у заглављу `<algorithm>`.

```

int ocena[5];
cin >> ocena[0] >> ocena[1] >> ocena[2] >> ocena[3] >> ocena[4];
sort(begin(ocena), end(ocena));
double prosek = (ocena[1] + ocena[2] + ocena[3] + ocena[4]) / 4.0;
cout << fixed << showpoint << setprecision(2)
    << prosek << endl;

```

Задатак: Подела бројевне праве на интервале

Бројевна права је трима различитим тачкама a , b и c подељена на четири дела. Написати програм који одређује у ком делу, првом, другом, трећем или четвртном, се налази унета (четврта) тачка x , различита од тачака

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

a , b и c .

Улаз: На стандардном улазу налазе се 4 различита реална броја, сваки у посебној линији, који редом представљају тачке којима је реалне праве подељена (тачке a , b и c) и тачку за коју одређујемо у ком делу се налази (тачку x).

Изназ: У једној линији стандардног излаза исписује број 1, 2, 3 или 4 у зависности ком делу тачка x припада.

Пример

Улаз	Изназ
5	2
-6	
3	
1	

Решење

Редослед интервала на које је бројевна права подељена тачкама a , b , c зависи од њиховог распореда. Постоји 6 могућих распореда деоних тачака ($a < b < c$, $a < c < b$, $b < a < c$, $b < c < a$, $c < a < b$, $c < b < a$). За сваку од тих могућности имамо 4 интервала на које је бројевна права подељена. Овакавим приступом решењу задатка имамо да разматрамо 24 могућа интервала, па врло лако можемо неки случај пропустити, а чак и да га не пропустимо, решење је веома ружно.

Једноставније и лепше је да изаберемо један од 6 могућих распореда тачака a , b , c и уредимо их тако да тај изабрани распоред важи. Размотримо случај кад уредимо тачке тако да је $a < b < c$ (поступци уређивања тј. сортирања три броја описани су у задатку **Уреди три броја**). После таквог уређења деоних тачака бројевна права је подељена редом на интервале $(-\infty, a)$, (a, b) , (b, c) и $(c, +\infty)$. Дакле, тачка x припада овим интервалима ако у првом случају важи $x < a$, у другом $a < x < b$, у трећем $b < x < c$ и у четвртном $x > c$. Ово се може остварити конструкцијом `else-if` како је објашњено у задатку **Агрегатно стање**.

```
// učitavamo tacke podele i cetvrtu tacku
double a, b, c, x;
cin >> a >> b >> c >> x;
// sortiramo tacke podele neopadajuci
sortiraj3(a, b, c);
// odredjujemo i ispisujemo interval kome tacka pripada
int deo;
if (x < a)
    deo = 1;
else if (x < b)
    deo = 2;
else if (x < c)
    deo = 3;
else
    deo = 4;
cout << deo << endl;
```

Задатак: Најмањи троцифрени број

Од цифара четвороцифреног природног броја треба формирати најмањи троцифрен број.

Улаз: У првој линији стандардног улаза налази се четвороцифрен природан број n ($1000 \leq n < 10000$).

Изназ: Најмањи троцифрен број добијен од цифара броја n .

Пример

Улаз	Изназ
5302	203

Решење

Из датог четвороцифреног броја n издвојимо цифру јединица c_0 , десетица c_1 , стотина c_2 и хиљада c_3 . Цифре издвајамо на уобичајени начин, на пример, одређивањем остатка при дељењу са 10 количника $\frac{n}{10^i}$ (види задатак **Збир цифара четвороцифреног броја**). Број n је четвороцифрен, па при издавајању цифре хиљада није неопходно одређивати остатак при дељењу са 10.

Издвојене цифре уредимо од најмање до највеће тако да је $c_0 \leq c_1 \leq c_2 \leq c_3$ (уређивање можемо извршити модификацијом неког од поступака приказаних у задатку **Уреди три броја**). Од три најмање цифре формирамо троцифрен број $c_0 \cdot 100 + c_1 \cdot 10 + c_2$ тако што за цифру стотина узмемо цифру c_0 , за цифру десетица цифру c_1 , а за цифру јединица c_2 . При томе морамо водити рачуна да би број био троцифрен цифра стотина мора бити различита од 0. Приликом формирања троцифреног броја за цифру стотина узимамо најмању цифру која је различита од 0, а за цифру десетица и цифру јединица две најмање од преосталих цифара.

```
// Funkcija za uredjivanje 4 cela broja a, b, c, d u neopadajucem
// poretku a <= b <= c <= d
void sortiraj4(int& a, int& b, int& c, int& d) {
    if (a > b) swap(a, b);
    if (a > c) swap(a, c);
    if (a > d) swap(a, d);
    if (b > c) swap(b, c);
    if (b > d) swap(b, d);
    if (c > d) swap(c, d);
}

// Funkcija za kreiranje trocifrenog broja abc
int trocifreniBroj(int a, int b, int c) {
    return 100 * a + 10 * b + c;
}

int main() {
    int broj;
    // citanje broja
    cin >> broj;

    // odredjivanje cifra broja
    int c0 = broj % 10;
    int c1 = broj / 10 % 10;
    int c2 = broj / 100 % 10;
    int c3 = broj / 1000;

    // uredjivanje cifara tako da je c0 <= c1 <= c2 <= c3
    sortiraj4(c0, c1, c2, c3);

    // formiranje i ispis najmanjeg trocifrenog broja
    int x;
    if (c0 != 0)
        x = trocifreniBroj(c0, c1, c2);
    else if (c1 != 0)
        x = trocifreniBroj(c1, c0, c2);
    else if (c2 != 0)
        x = trocifreniBroj(c2, c0, c1);
    else
        x = trocifreniBroj(c3, c0, c1);
    cout << x << endl;
    return 0;
}
```

Задатак: Једнакокраки од 4 дужи

Дата су четири позитивна реална броја који представљају дужине 4 дужи. Напиши програм који испитује да ли се од неке 3 од њих може направити једнакокраки троугао.

Улаз: Четири позитивна реална броја (сваки у засебном реду).

Излаз: Једна линија текста у којој пише да ако је могуће направити троугао тј. не ако није.

Пример 1

Улаз	Израз
1.1	ne
2.2	
3.3	
4.4	

Пример 2

Улаз	Израз
3.2	da
2.3	
3.2	
4.5	

Пример 3

Улаз	Израз
3.2	ne
2.3	
1.1	
1.1	

Решење

Да би три позитивна броја могла да представљају дужине страница једнакокраког троугла потребно и довољно је да су (бар) две од њих једнаке и да ти бројеви задовољавају неједнакост троугла (да је збир сваке две странице већи од треће) описану у задатку **Постоји ли троугао датих дужина страница**.

Да би тај троугао био једнакокраки, бар две од те три дужи морају бити једнаке дужине. Задатак, дакле, можемо решити тако што дефинишемо функцију која проверава да ли три броја могу чинити дужине страница једнакокраког троугла, а затим можемо применити ту функцију на испробавање свих могућих тројки дужи (на тројке (d_1, d_2, d_3) , (d_1, d_2, d_4) , (d_1, d_3, d_4) и (d_2, d_3, d_4)). Ипак, овакво решење би било неефикасно када би се уопштило на већи број датих дужи (јер би број тројки које потенцијално требало испитати био веома велики).

```
// provera da li se od tri duzi duzina a, b i c moze napraviti trougao
```

```
bool mozeTrougao(double a, double b, double c) {
    return a + b > c && a + c > b && b + c > a;
}
```

```
// provera da li se od tri duzi duzina a, b i c moze napraviti
// jednakokraki trougao
```

```
bool mozeJednakokrakiTrougao(double a, double b, double c) {
    return mozeTrougao(a, b, c) && (a == b || a == c || b == c);
}
```

```
int main() {
    // učitavamo duzine duzi
    double duz1, duz2, duz3, duz4;
    cin >> duz1 >> duz2 >> duz3 >> duz4;
    // proveravamo sve trojke
    if (mozeJednakokrakiTrougao(duz1, duz2, duz3) ||
        mozeJednakokrakiTrougao(duz1, duz2, duz4) ||
        mozeJednakokrakiTrougao(duz1, duz3, duz4) ||
        mozeJednakokrakiTrougao(duz2, duz3, duz4))
        cout << "da" << endl;
    else
        cout << "ne" << endl;
    return 0;
}
```

Решење засновано на сортирању

Задатак се ефикасније решава ако четири броја уредимо по величини, јер ће се тада два иста броја броја наћи један до другог (довољно је упоредити само суседне вредности у уређеном редоследу). Ако су сви бројеви различити, тада сигурно није могуће направити једнакокраки троугао. Зато претпоставимо да постоје два једнака броја (рецимо a и b , тако да је $a = b$).

Ако то нису два најмања броја тј. ако постоји број мањи од њих (рецимо c , тако да је $c < a$) тада они сигурно задовољавају све потребне неједнакости троугла. Заиста, пошто је $b > 0$ важи да је $a + b > a > c$, пошто је $c > 0$ важи да је $a + c > a = b$ и $b + c > b = a$.

Са друге стране, ако су a и b најмањи међу датим бројевима, тада је додатно потребно проверити једну неједнакост троугла. Нека су c и d преостала два броја и нека је c мањи од њих тј. нека важи да је $c > a$ и $c < d$. Пошто је $c > 0$ важи да је $a + c > a = b$, да је $b + c > b = a$, међутим, услов да је $a + b > c$ не мора да важи и треба га додатно проверити. Ако он важи онда се од дужи a , b и c може направити једнакокраки троугао. Ако не важи, тј. ако је $a + b \leq c$ онда се од њих не може оформити троугао, али се не може оформити ни од a , b и d јер је $a + b \leq c < d$.

Поступак уређивања тј. сортирања бројева описали смо у задатку **Уреди три броја**. Ако вредности чувамо у низу од четири елемента, тада сортирање можемо урадити коришћењем библиотечке функције.

```
// ucitavamo duzine duzi
double duzi[4];
cin >> duzi[0] >> duzi[1] >> duzi[2] >> duzi[3];
// sortiramo ih
sort(duzi, duzi+4);
// proveravamo da li je moguće napraviti jednakokraki trougao
if ((duzi[0] == duzi[1] && duzi[0] + duzi[1] > duzi[2]) ||
    duzi[1] == duzi[2] ||
    duzi[2] == duzi[3])
    cout << "da" << endl;
else
    cout << "ne" << endl;
```

Задатак: Прозор и кутија

Петар и Лука треба да изнесу кутију облика квадрата датих димензија a, b, c , кроз прозор облика правоугаоника димензија $p \times q$ тако да су одговарајуће ивице кутије буду паралелне ивицама прозора. Написати програм којим се проверава да ли је то могуће.

Улаз: Са стандардног улаза уносе се 5 природних бројева који представљају редом димензије кутије a, b, c и прозора p, q изражене у центиметрима.

Издаз: У једној линији стандардног излаза приказати реч да ако је могуће кутију изнети кроз прозор, а реч не ако то није могуће.

Пример 1		Пример 2	
Улаз	Издаз	Улаз	Издаз
75	da	30	da
30		40	
50		50	
70		30	
60		40	

Решење

Испитивање свих могућности

Проверу да ли кутија може да прође кроз прозор можемо урадити испитивањем свих могућих окретања кутије. Укупно имамо 6 могућих окретања и за свако треба проверити да ли кутија може да прође кроз прозор. Проверу када кутију окренемо тако да су ивице кутије a и b редом паралелне са ивицама прозора p и q вршимо упоређивањем дужина ивица, ако је $p > a$ и $q > b$ онда кутија може да прође кроз прозор. Тако проверимо за сва окретања и ако може бар једним окретањем да прође кутија кроз прозор одговор је да, у супротном је не. Дакле, могуће је испитати један сложени логички израз у коме су услови који одговарају сваком од 6 могућих окретања повезани оператором дисјункције (везником *или*), док се свака два унутрашња поређења повезују оператором конјункције (везником *и*).

```
int a, b, c, p, q;
cin >> a >> b >> c >> p >> q;
if ((p >= a && q >= b) || (p >= a && q >= c) ||
    (p >= b && q >= a) || (p >= b && q >= c) ||
    (p >= c && q >= a) || (p >= c && q >= b))
    cout << "da" << endl;
else
    cout << "ne" << endl;
```

Сортирање дужина ивица

Задатак можемо решити и на следећи начин. Ако уредимо дужине ивица кутије тако да је $a \leq b \leq c$ и дужине ивица прозора тако да је $p \leq q$ (поступке сортирања малих скупова бројева описали смо у задатку **Уреди три броја**). Кутија може да прође кроз прозор ако и само ако је $p > a$ и $q > b$. Заиста, ако овај услов није испуњен, онда је $p \leq a$ или $q \leq b$, а због уређености дужина ивица је онда $p \leq a \leq b$ или $p \leq q \leq b$, па не може бити

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

испуњено $p > b$ и $q > a$. Слично, кутија не може да прође кроз прозор ни на неки од преосталих начина, јер је трећа ивица c већа или једнака од ивица a и b , па ако не могу да прођу ивице a или b онда сигурно не може ни ивица c .

У случају да имамо проблем већих димензија ово решење је значајно ефикасније од поређења ивица у разним редоследима.

Сортирање можемо постићи коришћењем низа и библиотечке функције.

```
int a[3], p[2];
cin >> a[0] >> a[1] >> a[2] >> p[0] >> p[1];
sort(begin(a), end(a));
sort(begin(p), end(p));
if (a[0] <= p[0] && a[1] <= p[1])
    cout << "da" << endl;
else
    cout << "ne" << endl;
```

Сортирање можемо постићи и имплементацијом сортирања три променљиве селекцијом или уметањем.

Задатак: Сумо рвачи

Четири сумо рвача познатих тежина чекају испред лифта познате носивости. Колики је најмањи број возњи потребан да би се они превезли?

Улаз: Са стандардног улаза учитава се 5 целих бројева, сваки у посебном реду:

- a, b, c, d ($100 \leq a, b, c, d, \leq 200$) - тежине, у килограмима, 4 сумо рвача,
- L ($200 \leq L \leq 700$) - носивост лифта у килограмима.

Изназ: На стандардни излаз исписати један цео број - број возњи потребних да се рвачи превезу.

Пример

Улаз	Изназ
200	3
103	
160	
154	
280	

Решење

По претпоставци задатака сигурно је да лифт увек може да превезе сваког рвача посебно (зато што они имају до 200 килограма, а лифт може да носи бар 200 килограма).

Груба сила

Решење се може извести грубом силом, тј. анализирањем свих могућности. Једини начин да се сви превезу одједном је да је њихова укупна тежина мања или једнака носивости лифта тј. да важи да је $a + b + c + d \leq L$.

Ако то није могуће, постоји седам могућности да се превезу у две возње:

- могуће је да се у једном лифту возе рвач a и рвач b , а у другом рвач c и рвач d (тада мора да важи да је $a + b \leq L$ и да је $c + d \leq L$),
- могуће је да се возе a и c , а затим b и d (тада мора да важи да је $a + c \leq L$ и да је $b + d \leq L$),
- могуће је да се возе a и d , а затим b и c (тада мора да важи да је $a + d \leq L$ и да је $b + c \leq L$),
- могуће је да се возе рвачи a, b и c , у једном лифту, а затим рвач d у наредном (тада мора да важи да је $a + b + c \leq L$ и да је $d \leq L$, што сигурно важи по претпоставци задатка),
- могуће је да се возе рвачи a, b и d , у једном лифту, а затим рвач c у наредном (тада мора да важи да је $a + b + d \leq L$ и да је $c \leq L$, што сигурно важи по претпоставци задатка),
- могуће је да се возе рвачи a, c и d , у једном лифту, а затим рвач b у наредном (тада мора да важи да је $a + c + d \leq L$ и да је $b \leq L$, што сигурно важи по претпоставци задатка),
- могуће је да се возе рвачи b, c и d , у једном лифту, а затим рвач a у наредном (тада мора да важи да је $b + c + d \leq L$ и да је $a \leq L$, што сигурно важи по претпоставци задатка).

Ако ништа од тога није могуће, постоји шест могућности да се превезу у три возње:

- могуће је да се возе рвачи a и b , затим сам рвач c и на крају сам рвач d (тада мора да важи да је $a + b \leq L$, $c \leq L$ и $d \leq L$, при чему последња два услова сигурно важе по претпоставци задатка).
- могуће је да се возе рвачи a и c , затим сам рвач b и на крају сам рвач d (тада мора да важи да је $a + c \leq L$, $b \leq L$ и $d \leq L$, при чему последња два услова сигурно важе по претпоставци задатка).
- могуће је да се возе рвачи a и d , затим сам рвач b и на крају сам рвач c (тада мора да важи да је $a + d \leq L$, $b \leq L$ и $c \leq L$, при чему последња два услова сигурно важе по претпоставци задатка).
- могуће је да се возе рвачи b и c , затим сам рвач a и на крају сам рвач d (тада мора да важи да је $b + c \leq L$, $a \leq L$ и $d \leq L$, при чему последња два услова сигурно важе по претпоставци задатка).
- могуће је да се возе рвачи b и d , затим сам рвач a и на крају сам рвач c (тада мора да важи да је $b + d \leq L$, $a \leq L$ и $c \leq L$, при чему последња два услова сигурно важе по претпоставци задатка).
- могуће је да се возе рвачи c и d , затим сам рвач a и на крају сам рвач b (тада мора да важи да је $c + d \leq L$, $a \leq L$ и $b \leq L$, при чему последња два услова сигурно важе по претпоставци задатка).

На крају, ако ништа од овог није могуће, по претпоставци задатка је сигурно могуће да се превезу у четири вожње, тако да се свако вози сам (јер сигурно важи да је $a \leq L$, $b \leq L$, $c \leq L$ и $d \leq L$).

```
// tezine sumo rvaca
int a, b, c, d;
cin >> a >> b >> c >> d;
// nosivost lifta
int L;
cin >> L;

// izracunavamo broj voznji
int brojVoznji;
if (a+b+c+d <= L)
    brojVoznji = 1;
else if ((a+b <= L && c+d <= L) ||
         (a+c <= L && b+d <= L) ||
         (a+d <= L && b+c <= L) ||
         (a+b+c <= L /* && d <= L*/ ) ||
         (a+b+d <= L /* && c <= L*/ ) ||
         (a+c+d <= L /* && b <= L*/ ) ||
         (b+c+d <= L /* && a <= L*/ ))
    brojVoznji = 2;
else if (a+b <= L /* && c <= L && d <= L */ ||
         a+c <= L /* && b <= L && d <= L */ ||
         a+d <= L /* && b <= L && c <= L */ ||
         b+c <= L /* && a <= L && d <= L */ ||
         b+d <= L /* && a <= L && c <= L */ ||
         c+d <= L /* && a <= L && b <= L */)
    brojVoznji = 3;
else
    brojVoznji = 4;

// ispisujemo rezultat
cout << brojVoznji << endl;
```

Сортирање

Еlegantно решење може се добити ако се претпостави да су тежине рвача сортиране, тако да је $a \geq b \geq c \geq d$.

Поново, једна вожња је могућа само ако важи $a + b + c + d \leq L$.

Ако то није могуће онда су две вожње могуће ако и само ако важи да је $a + d \leq L$ и $b + c \leq L$ или ако важи да је $b + c + d \leq L$ и то су услови које је довољно проверити. Заиста, ако нека три рвача стају у један лифт, онда сигурно стају и три најлакша (тј. ако важи да је $a + b + c \leq L$ или је $a + b + d \leq L$ или је $a + c + d \leq L$ тада је сигурно и $b + c + d \leq L$ па је довољно проверити само тај услов). Ако су могуће неке две вожње са по два сумо рвача, онда је сигурно могуће да се возе најлакши и најтежи и средња два. Ако би важило да је $a + b \leq L$, тада би сигурно важило и да је $a + d \leq L$ (пошто из $b \geq d$ следи да је $a + d \leq a + b \leq L$) и да је $b + c \leq L$ (пошто из $a \geq c$ следи да је $b + c \leq b + a = a + b \leq L$). Слично, ако би важило да је $a + c \leq L$,

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

тада би важило и да је $a + d \leq L$ (јер из $c \geq d$ следи да је $a + d \leq a + c \leq L$) и да је $b + c \leq L$ (јер из $a \geq b$ следи да је $b + c \leq a + c \leq L$).

Ако две возње нису могуће, три возње су могуће ако и само ако важи да је $c + d \leq L$ и то је једини услов који треба проверити. Наиме, ако могу нека два рвача да стану у лифт, тада сигурно могу да стану два најлакша (на пример, ако би важило да је $b + c \leq L$, тада би из $b \geq d$ следило да је $c + d = d + c \leq b + c \leq L$).

Остаје још питање како технички реализовати сортирање тежина (разни начини сортирања неколико бројева описани су у задатку [Уреди три броја](#)). Сортирање можемо урадити разменама вредности променљивих.

```
// tezine 4 sumo rvaca
int a, b, c, d;
cin >> a >> b >> c >> d;
// nosivost lifta
int L;
cin >> L;

// sortiramo tezine nerastuce
sortiraj4(a, b, c, d);

// izracunavamo broj voznji
int brojVoznji;
if (a+b+c+d <= L)
    brojVoznji = 1;
else if ((a+d <= L && b+c <= L) || b+c+d <= L)
    brojVoznji = 2;
else if (c+d <= L)
    brojVoznji = 3;
else
    brojVoznji = 4;

// ispisujemo rezultat
cout << brojVoznji << endl;
```

Сортирање се може извршити и тако што се бројеви сместе у низ и да се позове библиотечка функција сортирања.

Задатак: Одсутна

Ана, Бранка, Весна и Гоца су четири сестре. Ана је виша од Бранке исто колико Бранка од Весне, као и Весна од Гоце. Ана и још две од сестара су биле на систематском прегледу, а четврта није. Одредити висину одсутне сестре ако су познате висине остале три.

Улаз: У сваком од три реда стандардног улаза по један цео број x ($85 \leq x \leq 200$), висине трију сестара у центиметрима, међу којима је и Анина. Подаци су дати у произвољном редоследу, али тако да увек постоји јединствено целобројно решење.

Излаз: Један цео број, висина четврте сестре у сантиметрима.

Пример

Улаз	Излаз
176	166
146	
156	

Задатак: Најјефтинији за динар

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. [Види тексты задатка.](#)

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Први и други максимум

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. [Види тексты задатка.](#)

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Најбољи такмичари

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шексти задатака.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Оријентација троугла

У равни има пуно тачака и свака од њих има свој редни број (од 1 до n). Троугао је одређен са три тачке, које су одређене својим редним бројевима. Међутим, проблем је то што су приликом записивања троугла тачке задате у произвољном редоследу. На пример, троугао $A_3A_5A_9$ може уместо тачака у редоследу A_3, A_5 и A_9 , бити задат тако што су записане тачке A_5, A_9 и на крају A_3 . Напиши програм који ће тачке унетог троугла исписати у основном редоследу (од најмањег броја, до највећег) и уз то одредити да ли распоред тачака који је унет одређује троугао који има исту или супротну оријентацију од овог добијеног уређивањем тачака. На пример, троуглови $A_3A_5A_9$ и $A_9A_3A_5$ имају међусобно исту оријентацију, док троугао $A_5A_3A_9$ има супротну оријентацију у односу на њих.

Улаз: Са стандардног улаза се уносе три различита природна броја, раздвојена размаком, који представљају редне бројеве тачака.

Излаз: На стандардни излаз исписати три броја раздвојена размаком који се добијају уређивањем бројева тачака од најмањег до највећег, а у наредном реду исписати *isto* ако полазни распоред одређује исту тј. *suprotno* ако одређује супротну оријентацију од оног добијеног уређивањем.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
32 14 55	14 32 55	11 5 8	5 8 11
	suprotno		isto

3.1.6 Позициони запис - вредност и цифре

Раније смо се већ упознали са основним дефиницијама позиционог записа и видели смо да су две основне операције израчунавање вредности броја на основу познатих цифара и израчунавање цифара датог броја.

Цифре у запису броја x у основи b се могу добити тако што се примењује низ корака таквих да се у сваком кораку одређује и брише цифра јединица броја. Обратимо пажњу на то да се у овом итеративном поступку број коме се одређују цифре стално мења. Цифру c_0 у запису броја $x = c_n \cdot b^n + \dots + c_1 \cdot b + c_0$ можемо одредити као $x \bmod b$, а важи да је $x \operatorname{div} b = c_n \cdot b^{n-1} + \dots + c_2 \cdot b + c_1$. Ако се сада број x замени вредношћу $x \operatorname{div} b$, цифру c_1 добијамо као остатак при дељењу са b , док је количник при дељењу са b једнак $c_n b^{n-2} + \dots + c_3 \cdot b + c_2$. Када се поступак настави на потпуно исти начин, можемо издвојити цифру c_2 , затим c_3 и тако даље. Цифре, дакле, добијамо поступком наредног облика.

```
c0 = x % b; x = x / b;
c1 = x % b; x = x / b;
...
```

Вредност броја се најједноставније одређује Хорнеровом схемом, кренувши од цифре највеће тежине, па уназад до цифара јединица. Подсетимо се, по Хорнеровој схеми вредност броја једнака је $(\dots((c_n \cdot b + c_{n-1}) \cdot b + c_{n-2}) \cdot b + \dots + c_1) \cdot b + c_0$. У итеративном поступку вредност иницијализујемо на нулу, а затим је у сваком кораку множимо основом b и додајемо наредну цифру. Вредност, дакле, одређујемо итеративним поступком наредног облика.

```
x = 0;
x = x * b + c[n];
x = x * b + c[n-1];
...
x = x * b + c[0];
```

Приметимо и да је број могао бити иницијализован на вредност c_n након чега би се до резултата стизало у једном кораку мање.

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

Индукцијом се лако може доказати да је након k корака вредност променљиве x једнака броју $(c_n \dots c_{n-k+1})_b$ који се добија од првих k цифара тј. $c_n \cdot b^{k-1} + c_{n-1} \cdot b^{k-2} + \dots + c_{n-k+1}$.

На пример, за вредности цифара 1, 2 и 3, променљива x имаће редом вредности 0, 1, 12, 123.

Итеративни поступак у ком вредност одређујемо на основу вредности цифара кренувши од цифре најмање тежине захтева да се поред вредности броја одржава и тежина наредне цифре, тј. текући степен основе (њега иницијализујемо на 1 и множимо га основом у сваком кораку).

```
x = 0; tezina = 1;
x = x + tezina * c[0]; tezina = tezina * b;
x = x + tezina * c[1]; tezina = tezina * b;
...
```

Индукцијом се лако може доказати да ће након k корака променљива x имати вредност једнаку броју $(c_{k-1} \dots c_0)_b$ који се добије од последњих k цифара, док ће променљива $tezina$ имати вредност 10^k .

На пример, за вредности цифара 1, 2 и 3, променљива x имаће редом вредности 0, 3, 23, 123.

Приметимо да Хорнерова шема не захтева израчунавање степена основе, што је за дуже бројеве чини ефикаснијом од класичног поступка.

Задатак: Збир цифара четвороцифреног броја

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Цифре можемо одредити и итеративним поступком. На пример, ако кренемо од броја 1234, прву цифру 4 одређујемо као остатак при дељењу броја са 10, а број затим мењамо његовим целобројним количником при дељењу са 10, тј. бројем 123. Наредну цифру 3 одређујемо као остатак при дељењу тог броја са 10, а број затим мењамо његовим целобројним количником при дељењу са 10, тј. бројем 12. Наредну цифру 2 одређујемо као остатак при дељењу тог броја са 10, а број затим мењамо његовим целобројним количником при дељењу са 10, тј. бројем 1. То је уједно и последња цифра броја (а она је такође једнака остатку при дељењу тог броја са 10).

```
// polazni broj i njegovo ucitavanje
int broj;
cin >> broj;

// izdvanjanje cifara
int cifraJedinica = broj % 10; broj = broj/10;
int cifraDesetica = broj % 10; broj = broj/10;
int cifraStotina = broj % 10; broj = broj/10;
int cifraHiljada = broj;

// izracunavanje zbira
int zbirCifara = cifraJedinica + cifraDesetica +
                cifraStotina + cifraHiljada;

// prikaz rezultata
cout << zbirCifara << endl;
```

Задатак: Октални бројеви

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Задатак се може решити итеративним поступцима. Одређивање вредности броја на основу цифара се може засновати на Хорнеровој шеми (тада број израчунавамо као $((c_3 \cdot 8 + c_2) \cdot 8 + c_1) \cdot 8 + c_0$). Одређивање цифара се

може извршити тако што се понавља низ корака у коме се наредна цифра израчунава као целобројни остатак при дељењу са 8) и уклања из броја, заменом броја његовим целобројним количником при дељењу са 8. Поново приликом имплементације можемо дефинисати и користити функцију која на основу броја одређује цифре (она враћа више вредности преко излазних параметара) и функцију која на основу цифара одређује вредност броја.

```
// odredjivanje vrednosti oktalnog zapisa (c3c2c1c0)8
int izOktalnog(int c3, int c2, int c1, int c0) {
    // Hornerova shema
    int n = 0;
    n = n * 8 + c3;
    n = n * 8 + c2;
    n = n * 8 + c1;
    n = n * 8 + c0;
    return n;
}

// prevodjenje broja n u oktalni zapis (c3c2c1c0)8
void uOktalni(int n, int& c3, int& c2, int& c1, int& c0) {
    // Iterativno određivanje cifara
    c0 = n % 8; n /= 8;
    c1 = n % 8; n /= 8;
    c2 = n % 8; n /= 8;
    c3 = n;
}

int main() {
    // izracunavanje vrednosti oktalno zapisanog broja
    int c3, c2, c1, c0;
    cin >> c3 >> c2 >> c1 >> c0;
    cout << izOktalnog(c3, c2, c1, c0) << endl;

    // izracunavanje cifara broja koji treba zapisati oktalno
    int n;
    cin >> n;
    uOktalni(n, c3, c2, c1, c0);
    cout << c3 << c2 << c1 << c0 << endl;
    return 0;
}
```

Задатак: UNIX време

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шекст задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Конверзија у милсекунде може да тече Хорнеровом схемом тако што се резултујућа променљива иницијализује на број дана, а затим се редом множи за 24 и увећава за број сати (тима се добије укупан број сати), затим се множи са 60 и увећава за број минута (тима се добија укупан број минута), затим се множи са 60 и увећава за број секунди (тима се добија укупан број секунди) и на крају се множи са 1000 и увећава за број милсекунди (и тако се добија тражени број милсекунди).

Конверзија од милсекунди тече тако што се итеративно одређује остатак и целобројни количник при дељењу редом са 1000, 60, 60 и 24, слично техници коју смо приказали у задатку **Поноћ**. Заиста, ово следи директно из услова да је $0 \leq \text{mili} < 1000$, $0 \leq \text{sek} < 60$, $0 \leq \text{min} < 60$, $0 \leq \text{sat} < 24$ и дефиниције количника и остатка при дељењу коју смо навели у задатку **Разломак у мешовит број**.

```
// prevodi broj dana, sati, minuta, sekundi i milisekundi u broj
// milisekundi
int uMilisekunde(int dan, int sat, int min, int sek, int mili) {
```

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

```
// Hornerova shema
int rez = dan;
rez = rez*24 + sat;
rez = rez*60 + min;
rez = rez*60 + sek;
rez = rez*1000 + mili;
return rez;
}

// prevodi broj milisekundi u broj dana, sati, minuta, sekundi
// i milisekundi
void odMilisekundi(int v, int& dan, int& sat, int& min,
                  int& sek, int& mili) {
    mili = v % 1000; v /= 1000;
    sek = v % 60; v /= 60;
    min = v % 60; v /= 60;
    sat = v % 24; v /= 24;
    dan = v;
}
```

Задатак: Збир два троцифрена

Ђаци увежбавају сабирање троцифрених бројева. Учитељица диктира задатак редом цифру по цифру. Напиши програм који на основу учитаних цифара израчунава и исписује тражени збир.

Улаз: У шест линија стандардног улаза задато је шест цифара.

Излаз: На стандардни излаз исписати један цео број - тражени збир.

Пример

Улаз	Излаз
1	579
2	
3	
4	
5	
6	

Решење

Кључни део решења може представљати алгоритам који на основу три цифре гради троцифрени број. Формирање броја на основу његових цифара ради се или на основу класичне дефиниције позиционог записа броја или Хорнеровим поступком. Када се од цифара формирају бројеви (представљени целобројним променљивама), можемо их лако сабрати коришћењем оператора +. Сличну технику смо видели у задацима у којима смо време задато у сатима, минутима и секундима преводили у секунде, пре извођења аритметичких операција.

Класичан поступак

Ако цифре стотина, десетица и јединица означимо са c_2 , c_1 и c_0 класичним поступком број добијамо изразом $c_2 \cdot 100 + c_1 \cdot 10 + c_0$. Да бисмо ово имплементирали потребно је да прво учитамо све три цифре (складиштимо их у три променљиве) и да онда израчунамо вредност броја.

Хорнеров поступак

Хорнеров поступак (каже се и Хорнерова шема) даје алтернативу чија је основна предност да, ако се цифре читају с лева надесно, тада се број може градити итеративно и није потребно истовремено памтити све цифре. Текућа вредност броја се иницијализује нулом, а затим се у сваком кораку учитава следећа цифра и она се дописује на текући број десна, тако што се број помножи са 10 и сабере са том цифром.

```
int ucitajTrocifreniBroj() {
    int broj = 0, cifra;
    cin >> cifra; broj = 10*broj + cifra;
    cin >> cifra; broj = 10*broj + cifra;
    cin >> cifra; broj = 10*broj + cifra;
}
```



```

    return broj;
}

int main() {
    int broj1 = ucitajTrocifreniBroj();
    int broj2 = ucitajTrocifreniBroj();
    cout << broj1 + broj2 << endl;
    return 0;
}

```

Види групачија решења овој задатка.

Задатак: Цифре десна

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види текст задатка.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Приказаћемо решење које неће прво учитавати све цифре, већ ће обрађивати цифру по цифру, редом којим пристижу. Број (памтићемо га у променљивој број) иницијализујемо на нулу, и у сваком кораку учитавамо нову цифру и дописујемо је слеве стране тако што тренутну вредност броја саберемо са вредношћу цифре помножену одговарајућим тежинским фактором. У првом кораку тај фактор је 1, у наредном 10, затим 100, 1000, 10000 и на крају 100000.

Употребићемо променљиву тежина у којој се памти тренутна вредност тежинског фактора (текући степен основе). Променљиву број ћемо ажурирати тако што ћемо је увећати за производ тренутно учитане цифре cifra и текуће вредности променљиве тежина. Променљиву тежина иницијализујемо на вредност 1 и на крају сваког корака ажурирамо множењем са 10.

```

// trenutna cifra, njena teжина i trenutna vrednost broja
int cifra, teжина, broj;

// nakon k koraka broj je sastavljen od poslednjih k cifara trazenog
// broja, a teжина ima vrednost 10^k
broj = 0; teжина = 1;

// ucitavamo novu cifru i dodajemo je na broj sa njegove leve strane
cin >> cifra;
broj += teжина*cifra;
teжина *= 10;

// ucitavamo novu cifru i dodajemo je na broj sa njegove leve strane
cin >> cifra;
broj += teжина*cifra;
teжина *= 10;

// ...
// ispisujemo rezultat
cout << broj << endl;

```

У елементарној имплементацији коју приказујемо копирамо исти код 6 пута. Ствар је, наравно, могуће побољшати употребом петље или бар издвајањем корака у засебну функцију.

Функција ће учитавати цифру и имати два параметра број и тежина који ће јој бити улазно-излазни (нови број и нова тежина израчунавају се на основу вредности учитане цифре, али и на основу старих вредности променљивих број и тежина). У језику С++ улазно-излазни параметри се преносе преко референце, тако што се иза назива типа параметра функције наведе симбол &.

```

void dodajCifru(int& teжина, int& broj) {
    // trenutna cifra
    int cifra;

```

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

```
// ucitavamo novu cifru i dodajemo je na broj sa njegove leve strane
cin >> cifra;
broj += tezina*cifra;
tezina *= 10;
}

int main() {
// trenutna tezina cifre i trenutna vrednost broja
int tezina, broj;

// nakon k koraka broj je sastavljen od poslednjih k cifara trazenog
// broja, a tezina ima vrednost 10^k
tezina = 1; broj = 0;
dodajCifru(tezina, broj);
dodajCifru(tezina, broj);
// ...
cout << broj << endl;
return 0;
}
```

Задатак: Избаци цифру

Напиши програм који одређује број добијен када се из датог троцифреног броја избацују сва појављивања његове цифре јединица.

Улаз: Са стандардног улаза се уноси један троцифрени број.

Излаз: На стандардни излаз исписати тражени број.

Пример 1		Пример 2		Пример 3		Пример 4	
Улаз	Излаз	Улаз	Излаз	Улаз	Излаз	Улаз	Излаз
131	3	133	1	123	12	333	0

Решење

У решењу овог задатка комбинујемо поступак одређивања серије цифара броја, филтрирања те серије и изградње новог броја од резултујућих цифара (оних различитих од цифара јединице полазног броја).

Један могући начин је да прво одредимо све цифре и да их упамтимо у посебним променљивима. Цифре броја можемо одредити на стандардни начин, комбиновањем целобројног дељења степеневима броја 10 и одређивања остатка при дељењу са 10, како смо показали у задатку **Збир цифара четвороцифреног броја**. Након тога, резултат можемо најлакше изградити Хорнеровим поступком, тако што га иницијализујемо на нулу, затим пролазимо кроз цифре оригиналног броја сдесна на лево и сваки пут када наиђемо на цифру која је различита од цифре јединице полазног броја додамо је на резултат на његову десну страну (тако што резултат помножимо са 10 и додамо му ту цифру), како смо показали у задатку **Јарди, стопе и инчи**.

```
// polazni broj
int n;
cin >> n;
// odredjujemo cifre polaznog broja
int c0 = (n / 1) % 10;
int c1 = (n / 10) % 10;
int c2 = (n / 100) % 10;
// Hornerovim postupkom gradimo rezultat, zadržavajući samo
// cifre različite od cifre jedinice
int rezultat = 0;
if (c2 != c0)
    rezultat = 10*rezultat + c2;
if (c1 != c0)
    rezultat = 10*rezultat + c1;
// naredni uslov sigurno neće biti ispunjen, pa se može preskociti
// if (c0 != c0)
```

```
// rezultat = 10*rezultat + c0;

// ispisujemo konacan rezultat
cout << rezultat << endl;
```

Уместо Хорнеровог поступка могућа је анализа различитих могућих случајева за вредности цифара стотина и десетица полазног броја, међутим, овакво решење је веома лоше (јер је случајева много, чак и за троцифрени број).

Ипак, поступак који је најпожељнији јер се, како ћемо касније видети, може лако уопштити и на бројеве са већим бројем цифара избегава потребу да се прво одреде све цифре, већ обрађује једну по једну цифру полазног броја, с десна на лево (јер је то редослед у коме можемо лакше одређивати цифре броја), истовремено је анализира и ако је потребно одмах дописује на текући резултат, овај пут са његове леве стране. Дакле, овде је потребно искористити алгоритам изградње броја с десна на лево, који је описан у задатку **Цифре с десна**. Променљиву `rezultat` иницијализујемо на вредност 0, `tezina` на вредност 1. У сваком кораку одређујемо последњу цифру полазног броја израчунавањем остатка при дељењу са 10 и одмах је уклањамо из полазног броја замењујући полазни број његовим целобројним количником са 10. Ако је текућа цифра (последња цифра текуће вредности полазног броја) различита од цифре која се избацује, тада је додајемо на резултат, претходно помножену вредношћу променљиве `tezina`, а променљиву `tezina` ажурирамо множећи је са 10. Поменути корак понављамо три пута (пошто знамо да је полазни број троцифрен).

```
// polazni broj
int n;
cin >> n;
// cifra jedinica polaznog broja
int c0 = n % 10;

// vrednost rezultata gradimo sdesna na levo
int rezultat = 0, tezina = 1, cifra;

// uklanjamo poslednju cifru tekuceg broja i ako je ona razlicita od
// cifre koja se izbacuje, dodajemo je na levu stranu rezultata
cifra = n % 10; n /= 10;
// naredni uslov sigurno nece biti ispunjen, pa se moze preskociti
// if (cifra != c0) {
//     rezultat += cifra*tezina;
//     tezina *= 10;
// }

// uklanjamo poslednju cifru tekuceg broja i ako je ona razlicita od
// cifre koja se izbacuje, dodajemo je na levu stranu rezultata
cifra = n % 10; n /= 10;
if (cifra != c0) {
    rezultat += cifra*tezina;
    tezina *= 10;
}

// uklanjamo poslednju cifru tekuceg broja i ako je ona razlicita od
// cifre koja se izbacuje, dodajemo je na levu stranu rezultata
cifra = n % 10; n /= 10;
if (cifra != c0) {
    rezultat += cifra*tezina;
    tezina *= 10;
}

// ispisujemo konacan rezultat
cout << rezultat << endl;
```

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

Задатак: Бројање оваца

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текснi задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Обрни цифре

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текснi задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Поноћ

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текснi задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Види групачија решења овог задатак.

Задатак: Тајмер

Тајмер се у тренутку $h : m : s$ (h од 0 до 23, m и s од 0 до 59) наштима да звони након периода који се исказује са $hh:mm:ss$ ($0 \leq hh, mm, ss \leq 1000$). Може се задати $100 : 100 : 100$ што је исто као да је задато $101 : 41 : 40$. Одредити када ће тајмер да звони у формату $h : m : s + dan$ (dan узима вредности почев од 0).

Улаз: Прво стартно време - у сваком реду по један податак: сат, минут и секунд. Затим трајање периода - у сваком реду по један податак: број сати, минута и секунди.

Излаз: Сат, минут, секунд и дан у формату $h:m:s+dan$.

Пример 1

Улаз	Излаз
23	0:0:0+1
59	
59	
0	
0	
1	

Пример 2

Улаз	Излаз
1	1:1:1+2
1	
1	
48	
0	
0	

3.1.7 Позициони запис - сабирање и одузимање

У наредних неколико задатака приказаћемо итеративне алгоритме за сабирање и одузимање бројева чије су цифре у неком позиционом запису познате, при чему је број цифара мали (бројеви су углавном двоцифрени или троцифрени). Исти ови алгоритми се примењују и на вишецифрене бројеве, међутим, тада се у имплементацији користе петље.

Задатак: Збир два троцифрена

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текснi задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Друга група решења користи алгоритам сабирања позиционо записаних бројева, без формирања вредности појединачних сабирака (то је алгоритам који сабира бројеве потписане један испод другог, који се учи у нижим разредима основне школе).

Директно сабирање (без израчунавања вредности бројева)

Још један од начина да израчунамо збир је да уопште не израчунавамо вредности бројева, већ да сабирамо цифру по цифру, као што се то ради када се два броја сабирају потписивањем. Убичајени начин да то урадимо је да саберемо цифре јединица, затим цифре десетица и на крају цифре стотина. При том, збир две цифре може да буде и већи од 10 и тада је потребно извршити пренос на следећу позицију. Приметимо да гранање није неопходно, јер се цифра резултата може израчунати као остатак збира цифара при дељењу са 10, док се пренос може израчунати као целобројни количник тог збира при дељењу са 10. На крају, потребно је

проверити да ли је остао пренос након сабирања цифара стотина, и ако јесте, тај пренос урачунати као цифру хиљада.

За пренос и збир цифара на текућој позицији могуће је употребити посебне променљиве. Да би се добио униформан код, понекад се пренос иницијализује нулом и урачунава се и у првом кораку.

```
// ucitavamo sve cifre
int a2, a1, a0;
int b2, b1, b0;
cin >> a2 >> a1 >> a0;
cin >> b2 >> b1 >> b0;

// sabiramo cifre odmah vrseci prenos
int prenos;
int zbirCifara;

zbirCifara = a0 + b0;
int z0 = zbirCifara % 10;
prenos = zbirCifara / 10;

zbirCifara = a1 + b1 + prenos;
int z1 = zbirCifara % 10;
prenos = zbirCifara / 10;

zbirCifara = a2 + b2 + prenos;
int z2 = zbirCifara % 10;
prenos = zbirCifara / 10;

int z3 = prenos;

// ispisujemo rezultat bez vodecih nula
if (z3 > 0) {
    cout << z3 << z2 << z1 << z0 << endl;
} else if (z2 > 0) {
    cout << z2 << z1 << z0 << endl;
} else if (z1 > 0) {
    cout << z1 << z0 << endl;
} else {
    cout << z0 << endl;
}
```

Друга могућа имплементација не користи посебне променљиве за текући збир и пренос, већ се и пренос и збир памте у истим променљивама које чувају цифре. Свака наредна променљива се иницијализује вредношћу преноса, да би се затим увећала за збир одговарајућих цифара.

```
// sabiranje sa prenosom, bez posebne promenljive
int z0 = a0 + b0;
int z1 = z0 / 10;
z0 %= 10;
z1 += a1 + b1;
int z2 = z1 / 10;
z1 %= 10;
z2 += a2 + b2;
int z3 = z2 / 10;
z2 %= 10;
```

У другој варијанти имплементације алгоритма сабирања у првој фази би се израчунали сви зборови, занемарујући пренос (тада је потпуно небитно којим редом се цифре сабирају, док год се увек сабирају цифре на истим позицијама). У другој фази се врше поправке тако што се кренувши позиције од цифре јединица гледа да ли је цифра збира на тој позицији достигла или превазишла вредност основе (у овом случају 10) и тада се врши пренос тако што се та цифра замени вредношћу свог остатка при дељењу са основом, док се њој

3.1. ОСНОВНИ АЛГОРИТМИ НАД МАЛИМ СЕРИЈАМА ЕЛЕМЕНАТА

претходна цифра увећа за вредност количника цифре збира са основом (и тако изврши пренос). Ни у овој варијанти нису неопходне помоћне променљиве.

```
// sabiramo cifre vrseci prenos u posebnoj fazi
int z0 = a0 + b0;
int z1 = a1 + b1;
int z2 = a2 + b2;
z1 += z0 / 10;
z0 %= 10;
z2 += z1 / 10;
z1 %= 10;
int z3 = z2 / 10;
z2 %= 10;
```

Види групација решења овој задајци.

Задатак: Време завршетка филма

Познати су сат, минут и секунд почетка филма и дужина трајања филма у секундама. Написати програм који одређује сат, минут и секунд завршетка филма.

Улаз: Са стандардног улаза учитавају се 4 природна броја (сваки у засебном реду). Прво сат, минут и секунд почетка филма, а затим дужина трајања филма у секундама.

Излаз: На стандардни излаз се исписује један ред у коме су три броја (сат, минут и секунд) раздвојена двотачкама који представљају време завршетка филма.

Пример

Улаз	Излаз
23	0:23:57
15	
22	
4115	

Решење

Време завршетка филма добијамо кад на време почетка филма додамо дужину трајања филма у секундама. Дакле, овај задатак подразумева сабирање два временска записа.

Директно сабирање позиционих записа

Један начин је да се сабирање ради директно, увећавањем броја секунди када филм почиње за дужину трајања филма. Добијени број секунди ће вероватно прећи 60, па је потребно пренети вишак минута и оставити број секунди такав да буде мањи од 60. Број минута који се преноси се може израчунати као целобројни количник увећаног броја секунди са 60, док је преостали број секунди остатак при дељењу увећаног броја секунди са 60. Након преноса минута, број минута ће вероватно прећи 60, па је потребно пренети вишак сати и оставити број минута таква да буде мањи од 60. Број сати који се преноси се може израчунати као целобројни количник увећаног броја минута са 60, док је преостали број минута остатак при дељењу увећаног броја минута са 60. Слично, и број сати може прећи 24 и тада је потребно пренети вишак сати (тима бисмо добили увећан број дана, међутим њега занемарујемо).

У претходном алгоритму примењујемо поступак сабирања два позициона записа. Ово је један од класичних итеративних поступака. Алгоритам сабирања бројева датих својим цифрама у неком позиционом запису (тада са основом 10) детаљно је описан у задатку **Збир два троцифрена**. Пошто број сати мора бити у интервалу од 0 до 23, а број минута од 0 до 59, приметимо да се овде заправо ради са записима у мешовитој бројевној основи (прва основа је 24, а друга 60).

```
// dodajemo duzinu u sekundama na vreme pocetka
// vrseci prenos istovremeno sa sabiranjem
int sZavrsetka = (sPocetka + duzina) % 60;
int prenos = (sPocetka + duzina) / 60;
int mZavrsetka = (mPocetka + prenos) % 60;
prenos = (mPocetka + prenos) / 60;
int hZavrsetka = (hPocetka + prenos) % 24;
```

```
// prenos broja dana se zanemaruje
// prenos = (hPocetka + prenos) / 24;

// ispisujemo rezultat
cout << hZavrsetka << ":" << mZavrsetka << ":" << sZavrsetka << endl;
```

Види групачија решења овој задатку.

Задатак: Тајмер

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текс задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Поноћ

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текс задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Трајање вожње

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текс задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

3.2 Основни алгоритми над серијама елемената

У претходном поглављу описали смо основне итеративне алгоритме у варијанти у којој су примењени на мале серије тако да су имплементирани без коришћења петљи. У овом поглављу ћемо исте алгоритме применити да дуже серије елемената (укључујући и серије чији број елемената није унапред познат). Предуслов за имплементацију таквих алгоритама је коришћење **петљи** (каже се и **циклуса**).

3.2.1 Елементи програмског језика

У језику C++ постоје три врсте петљи: `while`, `for` и `do-while`.

3.2.1.1 Петља `while`

Општи облик петље `while` је:

```
while (uslov)
    telo
```

У петљи `while` испитује се вредност логичког израза `uslov` и уколико је он тачан, извршавају се наредбе задате унутар тела петље. Ако се тело петље састоји од више наредби оне морају бити наведене унутар витичастих заграда. Услов петље `while` се први пут испитује пре извршавања тела петље. Када се тело изврши услов се поново испитује и поступак се понавља све док се први пут не деси да услов није испуњен. Свако извршавање тела петље називаћемо једном *итерацијом* петље.

3.2.1.2 Петља `for`

Општи облик петље `for` је:

```
for (inicijalizacija; uslov; korak)
    telo
```

Петља `for` се најчешће користи тако што се променљивој (која се назива *бројачка променљива*) редом додељују вредности од најмање до највеће и за сваку од тих вредности се извршавају наредбе у оквиру тела петље (ако их је више, морају се навести у витичастим заградама). У делу *inicijalizacija* се поставља почетна вредност бројачке променљиве (најчешће се на том месту и декларише бројачка променљива), у делу `uslov` се задаје услов петље који се проверава у свакој итерацији и први пут када није испуњен петља се прекида, док се у делу `korak` мења вредност бројачке променљиве. На пример, уколико желимо да испишемо све бројеве из интервала $[a, b]$ то можемо да урадимо наредном петљом:

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

```
for (int i = a; i <= b; i++)
    cout << i << endl;
```

Свака петља `for` може се једноставно изразити помоћу петље `while`. Иницијализацију је потребно извршити непосредно пре петље `while`, услов петље остаје исти, док се корак петље додаје као последња наредба у телу петље `while`. Дакле, претходно исписивање бројева из интервала $[a, b]$ смо могли да реализујемо и на следећи начин:

```
int i = a;
while (i <= b) {
    cout << i << endl;
    i++;
}
```

3.2.1.3 Петља `do-while`

Поред петље `while`, постоји и петља `do-while`, која јој наликује у свему, осим што се услов петље испитује на крају тела петље. Дакле, у овој петљи се тело извршава барем једном, без обзира на то да ли је услов испуњен или не (јер се он испитује на крају тела петље). На пример, наредни блок кода:

```
int i = a;
do {
    cout << i << endl;
    i++;
} while (i <= b);
```

би радио исправно ако је $a \leq b$, међутим у ситуацији када је $a > b$ решење засновано на `for` и `while` петљи не би исписивало ниједан број (што бисмо и очекивали), док би овај блок кода исписивао број a .

3.2.1.4 Прекиди петље (`break` и `continue`)

Некада је у програму потребно написати петљу која се извршава бесконачно пута. Уобичајен начин да се то постигне је коришћењем наредног фрагмента кода:

```
while (true){
    ...
}
```

Услов ове `while` петље је увек испуњен те се она извршава бесконачно пута. Из овакве петље може се изаћи коришћењем наредби `break` и `continue`. Наредба `break` у телу петље прекида извршавање петље. Наредба `continue` прекида извршавање тела петље (тј. извршавање текуће итерације).

3.2.2 Итерација кроз правилне серије бројева

Петље нам омогућавају да итерацију извршимо кроз правилне серије бројева (на пример, узастопне целе бројеве из неког интервала, аритметичке и геометријске серије бројева, серије подједнако размакнутих реалних бројева из неког реалног интервала и слично). Илуструјмо ово кроз неколико једноставних задатака. Технике које се у њима илуструју интензивно ће се користити у каснијим сложенијим задацима.

Задатак: Бројеви од a до b

Ако су дати цели бројеви a и b , написати програм који исписује редом све целе бројеве у задатом интервалу $[a, b]$.

Улаз: У првој линији стандардног улаза налази се цео број a , а у другој је цео број b ($-1000 \leq a \leq 1000$, $-1000 \leq b \leq 1000$).

Израз: На стандардном излазу исписују се редом сви цели бројеви из интервала, у свакој линији по један број.

Пример

Улаз	Изназ
3	3
6	4
	5
	6

Решење

Потребно је исписати све целе бројеве од a до b и да би се то могло урадити потребно је употребити петљу.

Најприроднији начин је да се употреби петља `for`. Користићемо петљу у којој ћемо бројачкој променљивој i додељивати редом вредности од најмање (то је вредност a) до највеће (то је вредност b) и у сваком пролазу кроз петљу исписивати њену вредност. У иницијализацији петље вредност променљиве i поставићемо на a , у услову ћемо проверавати да ли важи $i \leq b$, док ћемо у кораку петље вредност променљиве i увећавати за један.

```
int a, b;
cin >> a >> b;
for (int i = a; i <= b; i++)
    cout << i << endl;
```

Свака петља `for` може се једноставно изразити и помоћу петље `while`. Иницијализацију $i = a$ потребно је извршити непосредно пре петље `while`, услов петље је $i \leq b$, док се корак $i++$, $i += 1$ или $i = i + 1$ додаје као последња наредба у телу петље.

Задатак је могуће решити и исписивањем тренутне вредности променљиве a у петљи `while`, при чему се при сваком проласку кроз петљу вредност променљиве a увећа за 1, све док a не достигне вредност већу од b . Мана таквог решења је да након исписа бројева, оригинални интервал $[a, b]$ није више доступан.

```
int a, b;
cin >> a >> b;
int i = a;
while (i <= b) {
    cout << i << endl;
    i++;
}
```

Види другачија решења овој задајци.

Задатак: Бројање у игри жмурке

У игри жмурке деца обично броје по пет (5, 10, 15, 20, ...). Напиши програм који исписује баш те бројеве.

Улаз: Са стандардног улаза уноси се број x ($100 \leq x \leq 1000$) дељив са 5.

Изназ: На стандардни излаз исписати бројеве дељиве са 5, почевши од 5 и завршивши са x . Сваки број исписати у посебном реду.

Пример

Улаз	Изназ
30	5
	10
	15
	20
	25
	30

Решење**Набрајање бројева са кораком 5**

Приметимо сличност са задатком **Бројеви од a до b** . Једина разлика у односу њега је корак за који се увећава бројачка променљива. Бројачку променљиву потребно је иницијализовати на 5, итерацију вршити док је њена вредност мања или једнака од x и у сваком кораку је увећавати за 5 (помоћу $i += 5$ или $i = i + 5$). Наравно, ово је могуће реализовати било уз помоћу петље `for` било петље `while`.

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

```
int x;
cin >> x;
for (int i = 5; i <= x; i += 5)
    cout << i << endl;
```

Филтрирање

Још један могући приступ (додуше, мање ефикасан од претходног), је да се у петљи набрајају сви бројеви између 5 и x (слично као у задатку **Бројеви од a до b**), да се за сваки проверава да ли је дељив са 5 (слично као у задатку **Збир година браће и сестре**) и исписује се само ако јесте дељив. Кажемо да међу свим бројевима од 5 до горње границе *филтрирамо* оне дељиве са 5.

```
int x;
cin >> x;
for (int i = 5; i <= x; i++)
    if (i % 5 == 0)
        cout << i << endl;
```

Види групаџија решења овој задајџи.

Задатак: Троцифрени парни бројеви

За дате целе бројеве a и b , написати програм који исписује редом све парне троцифрене бројеве који припадају датом интервалу $[a, b]$

Улаз: Са стандардног улаза учитавају се бројеви a и b ($0 \leq a \leq 1500$ и $a \leq b \leq 1500$).

Излаз: На стандардном излазу исписују се редом (од најмањег до највећег) сви парни троцифрени бројеви, у свакој линији по један број.

Пример

Улаз	Излаз
85	100
109	102
	104
	106
	108

Решење

Овај задатак је веома сличан задатку **Бројање у игри жмурке**, једино што се као корак користи 2 уместо 5 и што доња и горња граница нису задате експлицитно.

Набрајање бројева са кораком 2

Потребно је исписати све троцифрене парне бројеве редом, почевши од најмањег троцифреног парног броја из целобројног интервала $[a, b]$ до највећег троцифреног парног броја из тог интервала. Троцифрени бројеви припадају целобројном интервалу $[100, 999]$. Дакле, тражени бројеви припадају пресеку та два интервала тј. интервалу $[\max(a, 100), \min(b, 999)]$ (одређивање пресека два интервала описано је у задатку **Интервали**). Пошто се траже само парни бројеви из интервала добијеног пресеком, набрајање треба почети од његове леве границе, ако је она парна, односно од њеног следбеника ако је она непарна, а завршити на десној граници ако је она парна тј. на њеном претходнику ако је она непарна. На тај начин добијамо интервал $[od, do]$ чије су границе парни бројеви и чије све парне бројеве треба исписати (провера да ли је број паран се своди на израчунавање остатака при дељењу са 2, слично како је описано у задатку **Збир година браће и сестре**).

Само набрајање бројева можемо извршити слично као у задатку **Бројање у игри жмурке**, било помоћу петље `for`, било помоћу петље `while`. У оба случаја бројачку променљиву ћемо иницијализовати на почетну вредност od , услов петље ће бити то да је вредност бројачке променљиве мања или једнака завршној вредности do , док ће корак петље бити увећање бројачке променљиве за 2. Нагласимо и да променљива не може да носи име `do` јер је то резервисана реч у већини програмских језика.

```
// ucitavamo interval [a, b]
int a, b;
cin >> a >> b;
```

```
// odredjujemo presek sa intervalom [100, 999]
int Od = max(a, 100), Do = min(b, 999);

// osiguravamo da levi i desni kraj budu parni brojevi
if (Od % 2 != 0)
    Od++;
if (Do % 2 != 0)
    Do--;

// vrsimo iteraciju i ispisujemo sve trazene brojeve
for (int i = Od; i <= Do; i += 2)
    cout << i << endl;
```

Филтрирање

Други начин (мање ефикасан од претходног) је заснован на филтрирању. Могуће је да се итерација (набрајање) врши редом по свим бројевима из целобројног интервала $[100, 999]$, да се за сваки број проверава да ли припада интервалу $[a, b]$ (да ли је између a и b) и да ли је паран, и да се број испишује само ако су оба услова испуњена.

```
// ucitavamo interval [a, b]
int a, b;
cin >> a >> b;

// za svaki trocifreni broj
for (int i = 100; i <= 999; i++)
    // proveravamo da li je paran broj iz intervala [a, b]
    if (a <= i && i <= b && i % 2 == 0)
        cout << i << endl;
```

Задатак: Одбројавање уназад

Написати програм који одбројава уназад од датог броја до нуле.

Улаз: Са стандардног улаза уноси се природан број a мањи од 100 од којег почиње одбројавање.

Излаз: На стандардном излазу испишују се редом (од највећег до најмањег) сви бројеви од a до нуле. Сваки број приказати у посебној линији.

Пример

Улаз	Излаз
3	3
	2
	1
	0

Решење

Потребно је исписати све бројеве од a до 0. Задатак можемо решити помоћу петље `for` или помоћу петље `while`. Ако користимо петљу `for`, тада ћемо променљивој i (бројачка променљива петље `for`), додељивати редом вредности од највеће (тј. од вредности a) до најмање (тј. до вредности 0) и у сваком пролазу кроз петљу исписивати њену вредност. Умањивање бројачке променљиве за један се може постићи изразом `i--`, `i -= 1` или `i = i - 1`.

```
int a;
cin >> a;
for (int i = a; i >= 0; i--)
    cout << i << endl;
```

Решење са петљом `while` се може имплементирати и без помоћне бројачке променљиве (тако што се умањује вредност a).

```
int a;
cin >> a;
```

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

```
while (a >= 0) {
    cout << a << endl;
    a--;
}
```

Задатак: Најаве емисије у правилним временским интервалима

Познат је почетак и крај једног филма (времена у сатима и минутима). У правилним временским интервалима, прво на почетку филма и затим након сваких m минута у углу екрана се приказује најаве следеће емисије. Напиши програм који исписује времена у којима се приказује та најаве.

Улаз: На стандардном улазу налази се време почетка филма дато у облику два цела броја, сваког у засебној линији који представљају број сати и број минута, затим, у следећој линији време завршетка филма дато у истом облику и на крају један цео број који представља интервал у минутима у којем се приказује најаве.

Изаз: На стандардном излазу приказати, времена емитовања најаве у облику $h:m$, свако у засебном реду.

Пример

Улаз	Изаз
12	12:0
0	12:15
13	12:30
23	12:45
15	13:0
	13:15

Решење

Рад са временом се олакшава ако се уведу помоћне функције за превођење датог времена у број минута протеклих од поноћи и обратно (слично као што је то рађено, на пример, у задатку [Поноћ](#)). Тако ћемо времена почетка и завршетка филма изразити у минутима. Времена приказивања најаве ћемо израчунати и исписати у оквиру петље у којој се користи променљива која представља време текуће најаве (опет у минутима). Полазећи од времена почетка филма у сваком кораку петље приказаћемо текуће време најаве и увећати га за интервал у минутима, док не престигне време завршетка филма, слично као у задатку [Бројање у игри жмурке](#). Наравно, ово је могуће имплементирати било петљом `for`, било петљом `while`.

```
// време pocetka filma
int hPocetak, mPocetak;
cin >> hPocetak >> mPocetak;
int pocetak = UMinute(hPocetak, mPocetak);

// време zavrsetka filma
int hKraj, mKraj;
cin >> hKraj >> mKraj;
int kraj = UMinute(hKraj, mKraj);

// interval u kojem se emituju najave
int interval;
cin >> interval;

// izracunavamo vremena u kojima se emituje najava i ispisujemo ih
for (int najava = pocetak; najava <= kraj; najava += interval) {
    int hNajava, mNajava;
    OdMinuta(najava, hNajava, mNajava);
    cout << hNajava << ":" << mNajava << endl;
}
```

Задатак: Подела интервала на једнаке делове

Написати програм којим се исписују вредности n равномерно размакнутих реалних бројева из интервала $[a, b]$, тако да је прва вредност a , а последња b .

Улаз: Прва линија стандардног улаза садржи природан број n ($1 < n \leq 20$), друга линија садржи реалан број a , а трећа линија реалан број b , при чему је $a < b$.

Издаз: На стандардном издазу приказати редом тражене бројеве, заокружене на пет децимала, сваки у посебној линији.

Пример

Улаз	Издаз
5	-1.00000
-1	-0.50000
1	0.00000
	0.50000
	1.00000

Решење

Потребно је прво одредити равномерно растојање dx , између n бројева интервала $[a, b]$ тако да је први број a , а последњи b . Таквих n бројева деле интервал $[a, b]$ на $n - 1$ једнаких делова, па је растојање dx између свака два броја $\frac{b-a}{n-1}$. Након тога је могуће применити исту технику као у задацима **Бројање у игри жмурке** или **Најаве емисије у правилним временским интервалима** и делује да га је могуће решити петљом облика

```
for (double x = a; x <= b; x += dx)
    ...
```

Међутим, због непрецизности до којих може доћи при раду са реалним бројевима, а о чему ће више речи бити у наставку збирке могуће је да се вредност растојања dx не може сасвим прецизно репрезентовати и да променљива $dx = (b - a) / (n - 1)$ заправо садржи вредност која је мало већа од стварне вредности растојања dx . Зато је могуће да се деси да вредност променљиве x у последњој итерацији тек за мало премаши вредност променљиве b и да се због тога прескочи исписивање десног краја интервала тј. тачке b , што је грешка.

Због тога је ипак пожељно итерацију контролисати бројачем помоћу којег се броје тачке које су исписане, тј. помоћу петље облика

```
for (int i = 0; i < n; i++)
    ...
```

Једна могућност је да се у реалној променљивој x чува вредност текуће тачке, да се пре почетка петље она иницијализује на вредност a , а да се у сваком кораку петље она испишује и затим увећава за вредност dx .

```
int n;
cin >> n;
double a, b;
cin >> a >> b;
double dx = (b - a) / (n - 1);
double x = a;
for (int i = 0; i < n; i++) {
    cout << setprecision(5) << showpoint << fixed
        << x << endl;
    x += dx;
}
```

Може се приметити да су тражени бројеви $a, a + dx, a + 2 \cdot dx, \dots, a + (n - 1) \cdot dx$, па се они могу приказивати као бројеви $a + i \cdot dx$ за све целе бројеве i од 0 до $n - 1$ (кажемо да *ипресликавамо* бројеве од 0 до $n - 1$ функцијом $f(i) = a + i \cdot dx$, што је слично задацима **Прерачунавање миља у километре** и **Табелирање функције**).

```
int n;
cin >> n;
double a, b;
cin >> a >> b;
double dx = (b - a) / (n - 1);
for (int i = 0; i < n; i++)
    cout << setprecision(5) << showpoint << fixed
        << a + i * dx << endl;
```

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

Задатак: Геометријска серија

Написати програм који за дате природне бројеве a , b исписује бројеве из интервала $[a, b]$, од којих је први број који се исписује једнак a , а сваки следећи је три пута већи од претходног. На пример, за $[a, b] = [5, 50]$ треба исписати 5, 15, 45.

Улаз: Са стандардног улаза се учитавају природни бројеви a ($1 \leq a \leq 50$) и b ($a \leq b \leq 10000$) сваки у посебном реду.

Излаз: На стандардном излазу исписују се сви тражени бројеви, редом (од најмањег до највећег). Сваки број исписати у посебној линији.

Пример

Улаз	Излаз
5	5
50	15
	45

Решење

Потребно је исписати све природне бројеве из интервала $[a, b]$, од којих је први број који се исписује једнак a , а сваки следећи три пута већи од претходног.

Задатак ћемо решити тако што ћемо у променљивој коју ћемо мењати кроз петљу одржавати текућу вредност ове серије бројева. Променљиву ћемо иницијализовати на вредност a , а затим у петљи проверавати да ли јој је вредност мања или једнака b и ако јесте исписиваћемо је и множићемо јој вредност са три. На оваквом месту је прилично природно употребити петљу `while` (нарочито под утицајем језика попут Паскала или Пајтона, где је петља `for` резервисана за серије у којима се бројачка променљива у сваком кораку увећава за 1 или евентуално за неки константан корак).

```
int a, b;
cin >> a >> b;
int i = a;
while (i <= b) {
    cout << i << endl;
    i *= 3;
}
```

Задатак можемо решити и рекурзивно (мада је то овде неефикасније и неприродније решење).

```
#include <iostream>

using namespace std;

void geometrijska_serija(int a, int b, int q) {
    if (a <= b) {
        cout << a << endl;
        geometrijska_serija(q*a, b, q);
    }
}

int main() {
    int a, b;
    cin >> a >> b;
    geometrijska_serija(a, b, 3);
    return 0;
}
```

3.2.3 Учитавање серија бројева

У великом броју задатака серије података које се обрађују учитавају се са стандардног улаза. У неколико наредних једноставних програма сусрешћемо се са најчешћим улазним форматима и техникама њиховог учитавања. Те технике ће се интензивно користити у каснијим сложенијим задацима.

Задатак: Збир n бројева

Написати програм којим се одређује збир n датих целих бројева.

Улаз: У првој линији стандардног улаза налази се природан број n ($1 \leq n \leq 1000$). У свакој од наредних n линија налази се по један цео број x_i .

Излаз: У првој линији стандардног излаза приказати збир унетих n целих бројева x_1, \dots, x_n .

Пример

Улаз	Излаз
4	13
10	
-3	
2	
4	

Решење

Пошто је познат укупан број n бројева које треба учитати, централно место у програму биће петља чије тело (у којем се учитава и обрађује следећи број) треба да се изврши тачно n пута. Пошто програмски језици обично немају наредбу облика `ponovi n puta`, извршавање наредби n пута најчешће се остварује петљом следећег облика:

```
for (int i = 0; i < n; i++)
    ...
```

За одређивање збира учитаних бројева користимо променљиву `zbir` у којој редом акумулирамо збир тренутно прочитаних бројева са стандардног улаза (идеју таквог поступног израчунавања збира видели смо, на пример, у задатку **Квалификациони праг**). Вредност променљиве `zbir` иницијализујемо на 0. Након тога, кроз петљу пролазимо n пута, а у сваком проласку учитавамо број са стандардног улаза и сваки прочитани број додајемо на збир. По завршетку петље, вредност збира приказујемо на стандардном излазу. На пример, ако се учитавају редом бројеви 1, 2, 3 и 4, променљива `zbir` ће редом имати вредности 0, 1, 3, 6 и 10. Нагласимо и да није било потребе истовремено памтити све бројеве да би се одредио њихов збир.

Збир серије бројева представља једну од основних њених статистика. У каснијим задацима показаћемо како можемо израчунавати и друге важне статистике (број елемената, производ, просек, минимум, максимум итд.).

```
int n; // broj brojeva
cin >> n;
int zbir = 0; // zbir brojeva
for (int i = 0; i < n; i++) { // n puta ponavljamo
    int broj; // ucitavamo broj
    cin >> broj;
    zbir += broj; // uvecavamo zbir za ucitani broj
}
cout << zbir << endl; // ispisujemo konacan zbir
```

Види груписања решења овог задатка.

Задатак: Читање до нуле

Уносе се цели бројеви док се не унесе нула. Написати програм којим се приказује колико је унето бројева, не рачунајући нулу.

Улаз: Свака линија стандардног улаза, изузев последње, садржи цео број различит од нуле. Последња линија садржи нулу.

Излаз: На стандардном излазу у првој линији приказати колко је учитано бројева, не рачунајући нулу.

Пример 1

Улаз	Излаз
5	3
-675	
123	
0	

Пример 2

Улаз	Излаз
0	0

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

Решење

Задатак ћемо решити помоћу разних врста петљи. У свим случајевима услов за излазак из петље је када је учитани број једнак 0.

У решењу помоћу петље `while` први број учитавамо пре петље, а у телу петље прво увећавамо бројач за 1 а затим учитавамо следећи број. На овај начин 0 као последњи учитани број неће довести до промене бројача.

```
int broj = 0;           // broj ucitanih brojeva razlicitih od nule
int x;                 // ucitavamo prvi broj
cin >> x;
while (x != 0) {       // dok nije ucitana nula
    broj++;            // uvecavamo brojac (nije ucitana nula)
    cin >> x;          // citamo naredni broj
}
cout << broj << endl; // ispisujemo rezultat
```

Ако користимо петљу `do-while` сва учитавања ћемо вршити у телу петље и при томе увећавати бројач за 1. На овај начин ћемо избројати све бројеве укључујући и 0, па се након изласка из петље вредност бројача умањује за 1.

```
int broj = 0;           // broj ucitanih brojeva razlicitih od nule
int x;                 // tekuci broj
do {                   // ponavljamo
    cin >> x;           // citamo naredni broj
    broj++;            // uvecavamo brojac
} while (x != 0);      // sve dok nije ucitana nula
broj--;                // izbrojana je i nula, pa smanjujemo brojac
cout << broj << endl; // prijavljujemo rezultat
```

Још једна могућност је да се испитивање услова прекида петље врши унутар њеног тела, непосредно након учитавања броја. То се може остварити тиме што се креира бесконачна петља облика `while (true)`, чији је услов стално испуњен, а унутар чијег тела се налази наредба `if (x == 0) break;`.

```
int broj = 0;           // broj ucitanih brojeva razlicitih od nule
int x;                 // tekuci broj
while (true) {         // ponavljamo sledece
    cin >> x;           // ucitavamo broj
    if (x == 0)         // ako je ucitana nula, prekidamo petlju
        break;
    broj++;             // uvecavamo brojac (nije ucitana nula)
}
cout << broj << endl; // ispisujemo konacan rezultat
```

Задатак: Читање до краја улаза

Са улаза се уносе цели бројеви све док се не дође до краја улаза. Написати програм којим се приказује колико је унето бројева.

Улаз: Свака линија стандардног улаза садржи по један цео број. **НАПОМЕНА:** приликом интерактивног тестирања програма, крај стандардног улаза се означава комбинацијом тастера `ctrl + z` ако се користи оперативни систем Windows тј. `ctrl + d` ако се користи оперативни систем Linux.

Излаз: У првој линији стандардног излаза приказати колико је бројева унето.

Пример

Улаз	Излаз
20	4
145	
-23	
-12	

Решење

Слично као у задатку **Читање до нуле** и овом задатку врши се пребројавање серије елемената. Бројач се иницијализује на нулу и увећава се за сваки учитани број. Кључно питање је како организовати петљу у којој се читавају бројеви све док се не достигне крај улаза.

У језику С++ оператор `>>` враћа вредност која се, када је потребно, конвертује у логичку вредност (вредност типа `bool`) `true` ако је претходно читање успело, тј. `false` ако није. Зато је најједноставнији начин да читају сви бројеви са улаза да се употреби петља облика

```
while (cin >> x)
    ...
```

Тада се програм може написати на следећи начин.

```
int broj = 0;           // broj ucitanih brojeva
int x;                 // tekuci broj
while (cin >> x)       // citamo brojeve dok ih ima
    broj++;            // uvecavamo brojac
cout << broj << endl; // prijavljujemo konacan rezultat
```

Решење можемо засновати и на обради изузетака. Учитавање вршимо у бесконачној петљи (петљи облика `У` телу петље покушавамо читавање броја, али у наредби која је ограђена блоком `Ако` читавање успе, увећавамо бројач. Када више није могуће прочитати број, наступиће изузетак и контрола ће се пребацити на део у који ћемо поставити наредбу `break` за прекид петље.

Задатак: Читање до -1 или до n-тог броја

Пера се током часа играо и сецкао квадрате од папира. Учитељица је то приметила и дала му је задатак да израчуна укупну површину свих квадрата које је исекао. Напиши програм који Пери помаже да то уради. Пера, на самом почетку уноси број n за који је сигуран да је већи или једнак броју квадратића. Након тога, уноси највише n бројева који представљају дужине страница квадратића, при чему, ако примети да квадратића има мање од броја n , он уноси -1 , чиме означава да је потребно прекинути унос.

Улаз: Са стандардног улаза се читава број n ($0 \leq n \leq 15$), а затим m ($1 \leq m \leq n$) целих бројева између 1 и 10, при чему, ако је $m < n$, онда се након њих уноси -1 .

Излаз: На стандардни излаз исписати један цео број који представља укупну површину свих квадрата.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
3	14	5	14
1		1	
2		2	
3		3	
		-1	

Решење

Задатак захтева израчунавање збира квадрата учитаних елемената. То се ради веома слично израчунавању збира елемената које смо приказали у задатку **Збир n бројева**, једино што се на збир уместо сваког учитаног броја x додаје његов квадрат. Основна тежина овог задатка не лежи у израчунавању збира квадрата, већ у организацији уноса података тако да се бројеви читавају све док се не прочита број -1 или док се не прочита n бројева.

Један начин да се то уради је да се организује петља у којој се читава n бројева (на уобичајени начин, као у задатку **Збир n бројева**), да се у телу те петље читава број, да се затим проверава да ли је учитан број -1 и ако јесте да се прекине петља (наредбом `break`), а ако није, да се збир увећа за квадрат учитаног броја. Напоменимо да се увећавање не мора вршити у грани `else`, већ може и након наредбе `break` - ако је услов да је учитан број -1 испуњен, извршиће се наредба `break`, петља ће се прекинути и неће се ни стићи до наредбе иза наредбе `break`.

```
int n;                               // broj brojeva
cin >> n;
int zbir = 0;                         // zbir kvadrata brojeva
for (int i = 0; i < n; i++) {         // ponavljamo n puta
    int x;                             // citamo novi broj
```

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

```
cin >> x;
if (x == -1)           // ako je ucitan broj -1
    break;             // prekidamo petlju
zbir += x * x;         // azuriramo zbir kvadrata
}
cout << zbir << endl; // ispisujemo konacan zbir
```

Један начин је да услов изласка из петље обухвати оба услова (да је број x различит од -1 и да је тренутни број учитаних елемената мањи од n). Пошто је због случаја $n = 0$ услов потребно проверавати пре читања и једног броја, морамо користити петљу са провером услова на почетку (петљу `for` или петљу `while`). Да би услов да је x различит од -1 био испуњен при првом уласку у петљу, променљиву x пре петље морамо иницијализовати на вредност различиту од -1 (иницијализоваћемо је, на пример, на вредност 0). Након учитавања броја, пре додавања његовог квадрата поново морамо проверити да ли је учитани број различит од -1 .

```
int n;                // broj brojeva
cin >> n;
int zbir = 0;         // zbir kvadrata ucitanih brojeva
int i = 0;            // broj ucitanih brojeva
int x = 0;            // tekuci broj
while (i < n && x != -1) { // dok je ucitano manje od n brojeva i
    // dok nije ucitana vrednost -1
    cin >> x;          // ucitavamo naredni broj
    i++;              // uvecavamo broj ucitanih brojeva
    if (x != -1)      // ako nije ucitan broj -1
        zbir += x * x; // uvecavamo zbir kvadrata
}
cout << zbir << endl; // ispisujemo konacan zbir kvadrata
```

Модификација овог решења уместо наредбе `break` може да користи логичку променљиву `kraj` којом се одређује да ли је уčitана вредност -1 . Њену вредност иницијализујемо на `false`, услов петље је да важи `!kraj` као и да је уčitано мање од n елемената, док се у телу петље, након учитавања броја проверава да ли је он -1 и ако јесте, вредност `kraj` се поставља на `true`, а у супротном се збир увећава за квадрат уčitаног броја.

```
int n;                // broj brojeva
cin >> n;
int zbir = 0;         // zbir kvadrata ucitanih brojeva
bool kraj = false;   // da li se stiglo do kraja
for (int i = 0; !kraj && i < n; i++) { // dok se ne stigne do kraja
    // a najvise n puta
    // ucitavamo broj
    int x;
    cin >> x;
    if (x == -1)      // ako je ucitan broj -1
        kraj = true; // stiglo se do kraja
    else               // u suprotnom
        zbir += x * x; // uvecavamo zbir kvadrata
}
cout << zbir << endl; // ispisujemo konacan rezultat
```

3.2.4 Пресликавање серије бројева

Пресликавање серије подразумева примену одређене трансформације (математичке функције) на сваки њен елемент.

Задатак: Прерачунавање миља у километре

Миља је енглеска историјска мера за дужину која износи 1609.344 m. Напиши програм који исписује таблицу прерачунавања миља у километре.

Улаз: Са стандардног улаза се уносе цели бројеви a ($1 \leq a \leq 10$), b ($10 \leq b \leq 100$) и k ($1 \leq k \leq 10$).

Излаз: На стандардни излаз исписати табелу конверзије миља у километре за сваки број миља из интервала $[a, b]$, са кораком k . Број километара заокружити на 6 децимала, а табелу приказати у формату идентичном као у примеру.

Пример

Улаз	Излаз
10	10 mi = 16.093440 km
20	12 mi = 19.312128 km
2	14 mi = 22.530816 km
	16 mi = 25.749504 km
	18 mi = 28.968192 km
	20 mi = 32.186880 km

Решење

Овај задатак представља класичан пример табелирања дате функције, тј. примене функције на сваки елемент неке серије бројева (кажемо да се врши *пресликавање серије*).

У петљи је потребно проћи кроз све бројеве миља из интервала $[a, b]$, са кораком k . То је најлакше урадити помоћу петље `for`, слично као у задатку **Бројање у игри жмурке** и **Троцифрени парни бројеви**. Број километара у сваком кораку петље добијамо тако што број миља помножимо са 1.609344.

```
const double kilometaraUMilji = 1.609344; // broj kilometara u milji
// ucitavamo interval i korak
int a, b, korak;
cin >> a >> b >> korak;
// pretvaramo sve zeljene brojeve milja u kilometre
for (int brojMilja = a; brojMilja <= b; brojMilja += korak) {
    double brojKilometara = brojMilja * kilometaraUMilji;
    cout << fixed << showpoint << setprecision(6)
         << brojMilja << " mi" << " = "
         << brojKilometara << " km" << endl;
}
```

Види групачија решења овој задатку.

Задатак: Табелирање функције

Аутомобил се креће равномерно убрзано са почетном брзином v_0 (израженом у $\frac{\text{m}}{\text{s}}$) и убрзањем a (израженим у $\frac{\text{m}}{\text{s}^2}$). Укупно време до постизања максималне брзине је T секунди. На сваких Δt секунди од почетка потребно је израчунати пређени пут аутомобила. Напомена: за равномерно убрзано кретање пређени пут након протеклог времена t изражава се са $s = v_0 \cdot t + \frac{a \cdot t^2}{2}$.

Улаз: Са стандардног улаза учитавају се 4 реална броја (сваки је у посебном реду):

- v_0 ($0 \leq v_0 \leq 5$) - почетна брзина
- a ($1 \leq a \leq 3$) - убрзање
- T ($5 \leq T \leq 10$) - укупно време
- Δt ($0.1 \leq \Delta t \leq 2.5$) - интервал

Излаз: На стандардни излаз исписати серију бројева који представљају пређени пут у задатим тренуцима.

Пример

Улаз	Излаз
1	0.000000
1	0.625000
2	1.500000
0.5	2.625000
	4.000000

Решење

Потребно је у петљи обићи све тачке t интервала $[0, T]$ (што се може урадити на исти начин како је то урађено у задатку **Најаве емисије у правилним временским интервалима**). За свако време t израчунавамо пређени пут по формули $s = v_0 \cdot t + \frac{a \cdot t^2}{2}$ (видели смо је у задатку **Бициклиста**) и исписујемо га.

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

Приметимо да се у овом задатку заправо врши табелирање једне (у овом случају квадратне) функције тј. пресликавање серије бројева, слично као у задатку **Прерачунавање миља у километре** у којем се вршило табелирање линеарне функције и све технике решавања се могу пренети из тог задатка.

```
// pocetna brzina i ubrzanje
double v0, a;
cin >> v0 >> a;
// vreme koje traje kretanje i interval u kome se prikazuje predjeni put
double T, dt;
cin >> T >> dt;
// krenuvsi od vremena 0 pa sve do T, na svakih dt izracunavamo
// predjeni put i ispisujemo ga
for (double t = 0; t <= T; t += dt) {
    double s = v0*t + a*t*t / 2.0;
    cout << fixed << showpoint << setprecision(5)
         << s << endl;
}
```

3.2.5 Елементарне статистике (број елемената, збир, производ, просек, средине, ...)

У неколико претходних задатака смо рачунали одређене једноставне статистике дужих серија бројева (на пример, у задатку **Збир n бројева** смо рачунали збир унетих бројева, у задатку **Читање до нуле** смо рачунали број учитаних елемената). У задацима који следе ћемо показати како се израчунавају и остале елементарне статистике дужих серија бројева: број њихових елемената, збир, производ, просек тј. аритметичку средину, друге облике средина (геометријску, хармонијску) итд.

Задатак: Факторијел

Дате су цифре $1, 2, \dots, n$. Напиши програм који израчунава колико се различитих n -тоцифрених бројева састављених од свих тих цифара може направити (на пример, од цифара 1, 2, 3 могу се направити бројеви 123, 132, 213, 231, 312 и 321).

Напомена: Број пермутација скупа од n елемената једнак је факторијелу броја n тј. броју $n! = 1 \cdot 2 \cdot \dots \cdot n$. Размисли зашто је баш тако.

Улаз: Прва линија стандарног улаза садржи природан број n ($1 \leq n \leq 9$).

Излаз: У првој линији стандарног излаза приказати број различитих бројева који се могу направити од цифара $1, 2, \dots, n$.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
5	120	9	362880

Решење

За одређивање факторијела природног броја n користимо променљиву `faktorijel` у којој редом акумулирамо производ бројева од 1 до n .

На почетку вредност променљиве `faktorijel` поставимо на 1. Након тога, коришћењем петље у којој ћемо бројачкој променљивој i , додељивати редом вредности од 1 до n , у сваком пролазу кроз петљу променљиву `faktorijel` множити са i .

Приметимо велику сличност са алгоритмом одређивања збира n бројева (види задатак **Збир n бројева**). Збир се иницијализује на вредност 0, што је неутрални елемент при сабирању, слично као што се производ иницијализује на вредност 1, што је неутрални елемент при множењу. Такође, у сваком кораку збир се увећава, док се производ множи текућим елементом серије.

```
int n;
cin >> n;
int faktorijel = 1;
for (int i = 1; i <= n; i++)
```

```
faktorijel = faktorijel * i;
cout << faktorijel << endl;
```

Види групачија решења овој задатку.

Задатак: Степен

Напиши програм који израчунава степен x^n . Покушај да програм напишеш без употребе библиотечких функција и оператора за степеновање.

Улаз: Са стандардног улаза се уноси реалан број x ($0.8 \leq x \leq 1.2$) и цео број n ($0 \leq n \leq 20$).

Излаз: На стандардни излаз испиши вредност x^n заокружену на пет децимала.

Пример

Улаз	Излаз
1.1	1.61051
5	

Решење

Множење

Резултат можемо добити ако израчунамо производ $\underbrace{x \cdot \dots \cdot x}_n$. Производ серије итеративно можемо израчунати слично као у задатку **Факторијел** - вредност производа иницијализујемо на јединицу, а затим га у сваком кораку петље множимо са наредним елементом серије (у овом случају вредношћу x).

```
double x;
int n;
cin >> x >> n;
double stepen = 1.0;
for (int i = 0; i < n; i++)
    stepen *= x;
cout << fixed << showpoint << setprecision(5)
     << stepen << endl;
```

Библиотечка функција степеновања

Већина језика већ пружа готову функцију или операцију степеновања. У језику C++ степеновање реалних бројева се врши функцијом `pow` декларисаној у заглављу `<cmath>`.

```
double x;
int n;
cin >> x >> n;
cout << fixed << showpoint << setprecision(5)
     << pow(x, n) << endl;
```

Види групачија решења овој задатку.

Задатак: Просек свих бројева до краја улаза

Са стандардног улаза се учитава број поена такмичара на такмичењу из програмирања. Напиши програм који израчунава просечан број поена свих такмичара.

Улаз: Сваки ред стандардног улаза садржи један цео број између 0 и 100. **НАПОМЕНА:** приликом интерактивног тестирања програма, крај стандардног улаза се означава комбинацијом тастера `ctrl + z` ако се користи оперативни систем Windows тј. `ctrl + d` ако се користи оперативни систем Linux.

Излаз: На стандардни излаз исписати просек заокружен на 5 децимала.

Пример

Улаз	Излаз
1	2.50000
2	
3	
4	

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

Решење

Као што смо већ видели, на пример, у задатку **Најлошији контролни**, рачунање просека серије бројева своди се на засебно израчунавање збира и броја елемената серије и њихово дељење (ако су и збир и број цели бројеви, пре дељења је потребно конвертовати их у реални тип).

У задатку **Збир n бројева** видели смо да се збир серије бројева израчунава тако што се променљива у којој се акумулира збир иницијализује на нулу, а затим се у петљи пролази кроз елементе серије и збир се увећава за текући елемент.

У задатку **Читање до нуле** видели смо да се број елемената серије одређује тако што се бројачка променљива у којој се тај број акумулира иницијализује на нулу, а затим се у петљи пролази кроз елементе серије и бројач се у сваком кораку увећава за један.

у задатку **Читање до краја улаза** показано је како се може организовати петља која учитава све бројеве са стандардног улаза.

```
int zbir = 0, broj = 0; // zbir i broj svih ucitanih brojeva
int x; // tekuci broj
while (cin >> x) { // citamo sve brojeve do kraja ulaza
    zbir += x; // uvecavamo zbir
    broj++; // uvecavamo broj
}
// izracunavamo i ispisujemo prosek
double prosek = (double)zbir/(double)broj;
cout << fixed << showpoint << setprecision(5)
    << prosek << endl;
```

Види групачија решења овој задатку.

Задатак: Средине

Аутомобил путује мењајући брзину током путовања. Познато је да се један део пута кретао равномерно брзином $v_1 \frac{\text{km}}{\text{h}}$, затим се један део пута кретао равномерно брзином $v_2 \frac{\text{km}}{\text{h}}$, и тако даље, све до последњег дела пута где се кретао равномерно брзином од $v_n \frac{\text{km}}{\text{h}}$. Написати програм који одређује просечну брзину аутомобила на том путу и то:

- ако се претпостави да је сваки део пута трајао исто време,
- ако се претпостави да је на сваком делу пута аутомобил прешао исто растојање.

Улаз: Са стандардног улаза уноси се n ($2 \leq n \leq 10$) позитивних реалних бројева: v_1, v_2, \dots, v_n (за свако v_i важи $30 \leq v_i \leq 120$), након чега следи крај улаза.

Издаз: У првој линији стандардног излаза исписати реалан број заокружен на 2 децимале који представља просечну брзину под претпоставком да је сваки део пута трајао исто време, а у другом реду реалан број заокружен на 2 децимале који представља просечну брзину под претпоставком да је на сваком делу пута аутомобил прешао исто растојање.

Пример

Улаз	Издаз
60	50.00
40	48.00

Решење

Просечна брзина представља однос укупног пређеног пута и укупног времена трајања пута (види задатак **Путовање**).

Претпоставимо да је сваки део пута трајао исто време t (у сатима). Тада је на првом делу пута аутомобил прешао $v_1 \cdot t$ километара, на другом $v_2 \cdot t$ километара, и тако даље, док је на последњем делу прешао $v_n \cdot t$ километара. Дакле, укупан пређени пут је $(v_1 + v_2 + \dots + v_n) \cdot t$ километара. Укупно време је $n \cdot t$ сати. Зато је просечна брзина једнака $\frac{(v_1 + v_2 + \dots + v_n) \cdot t}{n \cdot t} = \frac{v_1 + v_2 + \dots + v_n}{n}$.

Претпоставимо сада да је на сваком делу пута аутомобил прешао исто растојање s (у километрима). Тада се на првом делу пута кретао $\frac{s}{v_1}$ сати, на другом $\frac{s}{v_2}$ сати и тако даље, све до последњег дела где се кретао $\frac{s}{v_n}$

сати. Дакле, укупни пређени пут је $n \cdot s$, док је укупно време једнако $\frac{s}{v_1} + \frac{s}{v_2} + \dots + \frac{s}{v_n}$. Дакле, просечна брзина једнака је $\frac{s \cdot n}{\frac{s}{v_1} + \frac{s}{v_2} + \dots + \frac{s}{v_n}} = \frac{n}{\frac{1}{v_1} + \frac{1}{v_2} + \dots + \frac{1}{v_n}}$.

Приметимо да се у првом случају просечна брзина израчунава као *аритметичка средина* појединачних брзина, док се у другом случају израчунава као *хармонијска средина* појединачних брзина.

Израчунавање аритметичке средине (просечне вредности) свих бројева учитаних са стандардног улаза приказали смо у задатку **Просек свих бројева до краја улаза**. У овом случају радимо са реалним бројевима, па је имплементација још једноставнија јер не морамо да водимо рачуна о конверзији типова. Што се тиче израчунавања хармонијске средине, она се може израчунати тако што се израчуна збир реципрочних вредности свих брзина, а затим се n подели тим бројем. Збир реципрочних вредности се рачуна тако што се на променљиву коју иницијализујемо на нулу у петљи додају једна по једна реципрочна вредност.

```
int broj = 0; // broj delova puta
double zbir = 0.0; // zbir svih brzina
double zbirReciprocnih = 0.0; // zbir reciprocnih vrednosti svih brzina
double x;
while (cin >> x) { // ucitavamo sve brojeve do kraja ulaza
    broj++; // azuriramo sve statistike
    zbir += x;
    zbirReciprocnih += 1.0 / x;
}
// izracunavamo i ispisujemo aritmeticku i harmonijsku sredinu
double aritmetickaSredina = zbir / broj;
double harmonijskaSredina = broj / zbirReciprocnih;
cout << fixed << showpoint << setprecision(2)
    << aritmetickaSredina << endl
    << harmonijskaSredina << endl;
```

Види групачија решења овој задатку.

Задатак: Просечан раст цена

Потрошачка корпа је коштала 60000 динара. Након годину дана цене су порасле 21 проценат и потрошачка корпа је коштала 72600 динара. Након друге године цене су порасле још 44 процента и потрошачка корпа је коштала 104544 динара. Поставља се питање колико је просечно годишње порасла цена потрошачке корпе. Дефинишимо да ће просечан проценат пораста цене бити онај који, када се примени након сваке године, на крају дати исту цену као и у случају полазних пораста цена. У нашем примеру, ако би цена порасла 32 процента након годину дана потрошачка корпа би коштала 79200 динара, и ако би након друге године порасла још 32 процента, на крају би коштала поново тачно 104544 динара, тако да кажемо да је просечан пораст цене 32 процента. Напиши програм који за дате појединачне проценте пораста цене након сваке године израчунава просечан проценат пораста цене током целог периода.

Улаз: Са стандардног улаза се уноси број n (важи $2 \leq n \leq 10$) а затим n реалних бројева p_1, p_2, \dots, p_n који представљају проценте раста цене на крају сваке године (за свако p_i важи да је $5 \leq p_i \leq 50$).

Излаз: На стандардни излаз приказати један реалан број заокружен на две децимале који представља просечан проценат пораста цене.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
2	32.00	5	22.68
21			47.0
44			13.0
			13.5
			26.5
			16.5

Решење

Претпоставимо да је основна цена производа C . Тада је цена након првог месеца једнака $C \cdot (1 + \frac{p_1}{100})$, након два месеца једнака $C \cdot (1 + \frac{p_1}{100}) \cdot (1 + \frac{p_2}{100})$ и тако даље, да би на крају n месеци била једнака $C \cdot (1 + \frac{p_1}{100})(1 + \frac{p_2}{100}) \cdot \dots \cdot (1 + \frac{p_n}{100})$. Ако би пораст цене у сваком месецу било p тада би цена након n месеци

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

била једнака $C \cdot (1 + \frac{p}{100})^n$. Пошто желимо да овако израчунате коначне цене буду једнаке, тј. да важи $C \cdot (1 + \frac{p_1}{100}) \cdot \dots \cdot (1 + \frac{p_n}{100}) = C \cdot (1 + \frac{p}{100})^n$, p се може израчунати као

$$p = 100 \cdot \left(\sqrt[n]{(1 + \frac{p_1}{100}) \cdot \dots \cdot (1 + \frac{p_n}{100})} - 1 \right).$$

n -ти корен из производа n бројева назива се *геометријска средина*.

Производ серије можемо израчунати на исти начин као у задатку **Факторијел** - алгоритмом у коме резултат иницијализујемо на 1 и množимо га редом једним по једним кофицијентом повећања $1 + \frac{p}{100}$.

Корен можемо израчунати свођењем на степеновање $\sqrt[n]{x} = x^{\frac{1}{n}}$ и применом функције `pow` (као у задатку **Степен и корен**).

```
int n; // broj godina
cin >> n;
// učitavamo procenat rasta za svaku godinu i
// izračunavamo proizvod koeficijenata rasta
double proizvod = 1.0;
for (int i = 0; i < n; i++) {
    double x; // procenat rasta u tekucoj godini
    cin >> x;
    proizvod *= (1.0 + x / 100.0);
}
// izračunavamo i ispisujemo prosečno poskupljenje
double prosečnoPoskupljenje = 100.0 * (pow(proizvod, 1.0/n) - 1.0);
cout << fixed << showpoint << setprecision(2)
     << prosečnoPoskupljenje << endl;
```

Види групација решења овој задатку.

Задатак: Производња малина

Власник имања, Маринко, одлучио је да гаји малине. Направио је план за n година. Прве године планира да произведе t тона малина, а сваке следеће да повећа производњу за $p\%$. Написати програм којим се одређује колико тона малина Маринко планира да произведе n -те године узгајања малина.

Улаз: Прва линија стандардног улаза садржи природан број n ($n \leq 10$) број планираних година. Друга и трећа линија стандардног улаза садрже по један реалан број, који редом представљају колико тона малина Маринко планира да произведе прве године t ($0 < t < 5$) и за колико процената Маринко планира да повећа производњу сваке године p ($10 \leq p \leq 50$).

Излаз: На стандардном излазу приказати, на две децимале, колико тона малина Маринко планира да произведе n -те године узгајања малина.

Пример

Улаз	Излаз
5	4.88
2	
25	

Задатак: Сума низа бројева

Професор математике је поставио следећи задатак:

Одредити суму n бројева, ако је први број дати број a , а сваки следећи број добија се тако што претходни број помножимо са датим бројем q .

Ученик који први реши задатак добија петицу. Помозите ученицима, напишите програм којим се за дати природан број n , и дате реалне бројеве a и q одређује тражена сума.

Улаз:

- Прва линија стандардног улаза садржи природан број n ($n \leq 50$).

- Друга линија стандардног улаза садржи реалан број a ($0 < a \leq 10$).
- Трећа линија стандардног улаза садржи реалан број q ($0 < q < 1$).

Излаз: На стандардном излазу приказати, на пет децимала, тражену суму бројева.

Пример

Улаз	Излаз
4	37.50000
20.00	
0.50	

Задатак: Једнакост растојања

Становници једне дугачке улице желе да одреде положај на којем ће бити направљена антена за мобилну телефонију. Пошто желе да локацију одреде на најправеднији могући начин, договорили су се да антену саграде на месту на ком ће збир растојања свих оних који се налазе лево од антене до ње, бити једнак збиру растојања свих оних који се налазе десно од антене до ње. Ако су познате координате свих кућа у улици (можемо замислити да су то координате тачака на једној правој), напиши програм који одређује положај антене.

Улаз: Са стандардног улаза у првој линији се уноси природан број n ($1 \leq n \leq 100$) који представља број станара, а у наредних n линија реални бројеви (од -1000 до 1000) који представљају координате станара (x координате тачака на оси).

Излаз: На стандардни излаз исписати један реалан број који представља тражени положај антене (допуштена је толеранција грешке 10^{-5}).

Пример

Улаз	Излаз
5	-0.80800
-7.34	
15.6	
3.67	
-22.17	
6.2	

Задатак: Тежиште

Тежиште T троугла ABC је пресек његових тежишних дужи. Та тачка има и друга занимљива својства. На пример, када бисмо изрезали троугао од папира, могли бисмо да га балансирамо на врху оловке, само ако бисмо оловку поставили у тежиште троугла. Такође, тежиште је тачка таква да је збир квадрата растојања између ње и темена троугла (тј. израз $|AT|^2 + |BT|^2 + |CT|^2$) најмањи могућ. Уједно, то је једина тачка таква да је збир вектора $\vec{AT} + \vec{BT} + \vec{CT} = \vec{0}$. Користећи последње поменуто својство, појам тежишта се може уопштити на произвољан коначан скуп тачака (тада се понекад назива и барицентар или центроид). Напиши програм који за n тачака равни задатих својим координатама одређује њихово тежиште.

Улаз: Са стандардног улаза читава се број $3 \leq n \leq 100$ и затим $2n$ парова реалних бројева (x_i, y_i) , при чему је сваки број у посебном реду, који представљају координате n тачака равни.

Излаз: На стандардни излаз исписати координате тежишта тог скупа тачака, прво координату x , затим координату y , сваку у посебном реду, заокружену на пет децимала.

Пример

Улаз	Излаз
3	0.33333
0	0.33333
0	
0	
1	
1	
0	

Решење

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

Овај задатак представља уопштење задатка **Једнакост растојања** у коме је заправо разматрано тежиште скупа тачака на правој и одређено је као просечна вредност координата тачака. Исто ће се догодити и код тачака у равни. Заиста, обележимо тежиште са T , а тачке са A_1, \dots, A_n , а са $\vec{OT} = (T_x, T_y)$, $\vec{OA}_1 = (A_{1x}, A_{1y})$, \dots , $\vec{OA}_n = (A_{nx}, A_{ny})$ векторе њихових координата. Кренимо од услова да је $A_1\vec{T} + \dots + A_n\vec{T} = \vec{0}$. У последњој једнакости додамо на обе стране $\vec{OA}_1 + \vec{OA}_2 + \dots + \vec{OA}_n$ и добијемо $n \cdot \vec{OT} = \vec{OA}_1 + \vec{OA}_2 + \dots + \vec{OA}_n$, односно $\vec{OT} = \frac{\vec{OA}_1 + \vec{OA}_2 + \dots + \vec{OA}_n}{n}$.

Одатле следи:

$$T_x = \frac{A_{1x} + A_{2x} + \dots + A_{nx}}{n}$$

$$T_y = \frac{A_{1y} + A_{2y} + \dots + A_{ny}}{n}$$

Задатак се решава једноставним израчунавањем аритметичких средина координата x и координата y датих тачака, на потпуно исти начин као што је то урађено у задатку **Једнакост растојања**.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
    // број тачака
    int n;
    cin >> n;
    // збир координата x и координата y
    double zbirX = 0.0, zbirY = 0.0;
    for (int i = 0; i < n; i++) {
        // координате текуће тачке
        double x, y;
        cin >> x >> y;
        // укупна збировања
        zbirX += x; zbirY += y;
    }
    // израчунавања просека
    double Tx = zbirX / n, Ty = zbirY / n;
    // исписујемо резултат
    cout << fixed << showpoint << setprecision(5)
         << Tx << endl << Ty << endl;
    return 0;
}
```

3.2.6 Филтрирање серије бројева

Филтрирање подразумева издвајање свих елемената серије који задовољавају неки дати услов. Над филтрираном серијом обично се спроводи даља обрада (пресликавање, израчунавање различитих статистика и слично).

Задатак: Бројеви дељиви са 3

Напиши програм који међу унетим бројевима одређује и исписује оне који су дељиви са 3.

Улаз: Са стандардног улаза се најпре учитава природан број n ($1 \leq n \leq 1000$), а потом и n природних бројева из интервала $[1, 10^8]$, сваки у посебном реду.

Изаз: На стандардни излаз исписати све учитане бројеве који су дељиви са 3 (у истом редоследу у ком су учитани). Сваки број исписати у посебној линији.

Пример

Улаз	Изназ
5	12
100	18
11	102
12	
18	
102	

Решење

Потребно је исписати само оне учитане бројеве који су дељиви са 3. Учитавање n бројева постижемо коришћењем петље (слично као што смо радили, на пример, у задатку **Збир n бројева**).

У овом задатку је природно да се користи петља `for`, мада је могуће користити и `while`. У телу петље читавамо текући број и коришћењем наредбе гранања проверавамо да ли је дељив са 3. Проверу дељивости вршимо упоређивањем остатка при дељењу са 3 и нуле (како смо видели у задатку **Збир година браће и сестре**). Ако број јесте дељив са 3, исписујемо га.

Овај задатак је класичан пример тзв. алгорита *филтрирања серије* тј. издвајања елемената који задовољавају неки дати услов. У општем случају у петљи пролазимо кроз елементе серије, наредбом гранања проверавамо да ли текући елемент задовољава задати услов, и ако задовољава у телу наредбе гранања га обрађујемо на одговарајући начин.

```
// ponavljamo n puta
int n;
cin >> n;
for (int i = 1; i <= n; i++) {
    // ucitavamo broj
    int broj;
    cin >> broj;
    // ako je deljiv sa 3, ispisujemo ga
    if (broj % 3 == 0)
        cout << broj << endl;
}
```

Види групачија решења овог задатка.

Задатак: Бројање гласова за омиљеног глумца

Ученици су бирали свог омиљеног глумца. Напиши програм који Ђоки помаже да сазна колико је гласова добио његов омиљени глумац.

Улаз: Сваки глумац је означен природним бројем од 1 до 10. Прва линија стандардног улаза садржи редни број x Ђокиног омиљеног глумца. Друга линија стандардног улаза садржи природан број n ($1 \leq n \leq 100$) - број ученика у школи. Наредних n линија садрже гласове сваког ученика (бројеве од 1 до 10, сваки у посебном реду).

Изназ: На стандардни излаз исписати један природан број - број гласова које је добио глумац x .

Пример

Улаз	Изназ
2	2
4	
1	
2	
3	
2	

Решење**Број елемената филтриране серије**

Потребно је избројати колико елемената серије задовољава неки дати услов (у овом случају услов је да су једнаки броју x). Уобичајени начин да се то уради је да се уведе бројач елемената који иницијализујемо на

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

нулу (слично као у задатку **Читање до нуле**), да у петљи учитавамо један по један елемент серије и да ако елемент задовољава тражени услов увећавамо бројач за један.

Издвајање елемената који задовољавају дати услов (филтрирање) употребили смо и у задатку .

```
int x; // sifra omiljenog glumca
cin >> x;
int n; // ukupan broj glasova
cin >> n;
int brojGlasova = 0; // broj glasova za omiljenog glumca
for (int i = 0; i < n; i++) { // analiziramo sve glasove
    int y;
    cin >> y;
    if (x == y) // ako je tekuci glas za omiljenog glumca
        brojGlasova++; // uvecavamo broj glasova
}
cout << brojGlasova << endl; // ispisujemo broj glasova
```

Види групација решења овој задатку.

Задатак: Просек одличних

Дате су просечне оцене n ученика једног одељења. Написати програм којим се одређује просек просечних оцена свих одличних ученика тог одељења.

Улаз: Прва линија стандарног улаза садржи природан број n ($1 \leq n \leq 100$) који представља број ученика. У наредних n линија налази се по један реалан број из интервала $[2, 5]$. Ти бројеви представљају просеке ученика.

Изназ: У првој линији стандарног излаза приказати просек просечних оцена одличних ученика одељења заокружен на две децимале. Ако одличних ученика нема приказати -.

Пример 1		Пример 2	
Улаз	Изназ	Улаз	Изназ
4	4.62	4	-
3.5		3.5	
4.75		3.75	
3		2.80	
4.5		4.35	

Решење

Овај задатак је веома сличан задатку **Квалификациони праг**. Кључна разлика је у томе што број ученика није унапред фиксиран, па је у решењу потребно употребити петљу у којој се читавају и обрађују просеци појединачних ученика. Такође, постоји сличност са задатком **јер је** и у овом случају потребно међу свим учитаним елементима серије потребно анализирати само оне који задовољавају дати услов.

Просек просечних оцена одличних ученика одређујемо тако што збир просечних оцена одличних ученика поделимо бројем одличних ученика. Рачунање просека (аритметичке средине) серије произвољне дужине видели смо, на пример, у задатку **Просек свих бројева до краја улаза**, једина разлика је у томе што се у овом задатку не рачуна просек свих учитаних просечних оцена ученика, већ само оних који су одлични, тј. пре израчунавања просека серија се филтрира на основу услова да је просечна оцена већа или једнака 4.5.

Користимо променљиве у којима редом чувамо збир просека и број одличних ученика. На почетку обе променљиве постављамо на 0, јер у почетку још нисмо читали ни један просек ученика. Затим у петљи редом читамо просеке ученика и ако је ученик одличан (ако је текући просек већи од 4,5) додајемо његову просечну оцену збиру просека одличних док број одличних ученика увећавамо за 1. Кад смо прочитали просечне оцене свих ученика одељења, након петље, проверавамо да ли је број одличних ученика једнак нули и ако јесте приказујемо -, а иначе одређујемо просечну вредност просека одличних ученика дељењем вредности збира и броја одличних.

```
int n; // ukupan broj ucenika
cin >> n;
// ucitavamo proseke svih ucenika i odredjujemo broj odlicnih ucenika
```

```

// i zbir njihovih proseka
double zbirProsekaOdlicnih = 0.0;
int brojOdlicnih = 0;
for (int i = 0; i < n; i++) {
    double tekuciProsek;
    cin >> tekuciProsek;
    if (tekuciProsek >= 4.5) {
        zbirProsekaOdlicnih += tekuciProsek;
        brojOdlicnih++;
    }
}
// odredjujemo i ispisujemo prosek proseka odlicnih učenika
if (brojOdlicnih != 0) {
    double prosekOdlicnih = zbirProsekaOdlicnih / brojOdlicnih;
    cout << fixed << setprecision(2) << showpoint
        << prosekOdlicnih << endl;
} else
    cout << "-." << endl;

```

Види груписања решења овог задатка.

Задатак: Категорије џудиста

На једном турниру џудисти се такмиче у три категорије: до 50 килограма, од 51 до 75 килограма и од 76 килограма навише. Напиши програм који читава број џудиста једног клуба пријављеног на тај турнир, а затим тежину сваког од њих и за сваку категорију редом исписује колико ће се џудиста тог клуба борити у тој категорији.

Улаз: Са стандардног улаза се читава број џудиста n ($1 \leq n \leq 100$), а затим у n наредних редова тежине џудиста (цели бројеви између 30 и 120).

Излаз: На стандардни излаз исписати три броја, сваки у посебном реду: број џудиста који имају мање или једнако од 50 килограма, број џудиста који имају између 51 и 75 килограма и број џудиста који имају више или једнако од 76 килограма.

Пример

Улаз	Излаз
5	2
48	2
51	1
73	
82	
50	

Решење

Основу задатка чини разврставање џудиста по категоријама (што можемо урадити гранањем на основу припадности интервалима, конструкцијом `else-if` као у задатку [Агрегатно стање](#)). Увешћемо три бројачке променљиве, сваку иницијализовати на нулу и повећавати за 1 сваки пут када наиђемо на џудисту у одговарајућој категорији. Дакле, разврставамо серију бројева на три подсерије (ово донекле одговара алгоритму филтрирања серије описаном у задатку) и израчунавамо број елемената сваке од три подсерије (као у задатку [Читање до нуле](#)).

```

// broj dzudista u raznim kategorijama
int broj_do_50 = 0;
int broj_od_51_do_75 = 0;
int broj_od_76 = 0;
// ukupan broj dzudista
int n;
cin >> n;
// učitavamo i analiziramo težine svih dzudista
for (int i = 0; i < n; i++) {

```

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

```
int tezina;
cin >> tezina;
if (tezina <= 50)
    broj_do_50++;
else if (tezina <= 75)
    broj_od_51_do_75++;
else
    broj_od_76++;
}
// ispisujemo rezultate za svaku kategoriju
cout << broj_do_50 << endl;
cout << broj_od_51_do_75 << endl;
cout << broj_od_76 << endl;
```

Задатак: Статистике33

Чувени кошаркаш Дабло Трипловић има толико добре статистике, да се већ помало сумња да су његови успеси добро пребројани.

Написати програм који одређује колико пута је Трипловић постигао такозвани трипл–дабл.

Улаз: На стандардном улазу се у првом реду налази цео број N , ($1 \leq N \leq 100$), број утакмица које је Трипловић одиграо у сезони. У сваком од следећих N редова по 3 цела броја раздвојена по једним размаком: број поена, скокова и асистенција редом (ови бројеви нису већи од 1000).

Излаз: На стандардни излаз исписати један број, број утакмица на којима је Трипловић постигао трипл–дабл.

Напомена

За потребе овог задатка трипл–дабл дефинишемо као најмање двоцифрен учинак (10 или више) у све три категорије које се прате. Другим речима: сва три броја већа од 9 у једном реду улаза.

Пример

Улаз	Излаз
3	2
20 9 7	
127 12 11	
11 14 12	

Задатак: Секција

У школи је велико интересовање за програмерску секцију, на којој ће се правити рачунарске игре. Наставник је поставио услов да на секцију могу да иду само они ученици који имају 5 из информатике и бар 4 из математике.

Написати програм који одређује колико ученика може да се пријави за секцију.

Улаз: у првом реду број N , број ученика заинтересованих за секцију, природан број не већи од 200. У сваком од следећих N редова низ оцена једног од заинтересованих ученика из свих 11 предмета редом, раздвојени по једним размаком. Оцена из математике је шеста, а из информатике девета у низу од 11 цифара.

Излаз: На стандардни излаз исписати један цео број, број ученика који испуњавају услов.

Пример 1

Улаз	Излаз
4	2
5 5 5 5 5 3 5 5 4 5 5	
5 5 5 5 5 4 5 5 4 5 5	
5 5 5 5 5 4 5 5 5 5 5	
2 2 2 2 2 5 2 2 5 2 2	

Пример 2

Улаз	Излаз
2	0
5 5 5 5 5 5 5 5 4 5 5	
5 5 5 5 5 3 5 5 5 5 5	

Задатак: Купци

Менаџер једног великог ланца продавница жели да зна структуру прихода. Он је зато све куповине разврстао у мале, средње и велике на следећи начин: ако је износ рачуна за неку куповину X , та куповина се сматра за малу ако је $X \leq M$, за средњу ако је $M < X \leq V$, а за велику ако је $V < X$. Менаџер жели да за посматрани период зна колики је укупан приход продавница од малих, средњих и великих куповина редом.

Улаз: Са стандардног улаза се у првом реду учитавају бројеви M и V ($1 \leq M < V \leq 100000$). У другом реду је цео број n , укупан број куповина у свим продавницама у посматраном периоду ($1 \leq n \leq 50$). У трећем и последњем реду су исноси рачуна за n куповина, цели позитивни бројеви до 200000 раздвојени по једним размаком.

Изназ: Три цела броја, сваки у посебном реду. Ови бројеви су укупни приходи од малих, средњих и великих куповина.

Пример

Улаз	Изназ
100 1000	188
5	512
88 1010 512 100 2500	3510

3.2.7 Минимум и максимум

У задацима који следе биће приказано израчунавање највећег и најмањег елемента серије (минимума и максимума), њихове позиције и још неких варијација ових алгоритама (на пример, проналажење другог елемента по величини). Нагласак ће бити стављен и на примену ових статистика у решавању конкретних проблема.

Задатак: Најнижа температура

Дате су температуре у неколико градова. Написати програм којим се одређује, колика је температура у најхладнијем граду.

Улаз: Прва линија стандардног улаза садржи природан број n ($3 \leq n \leq 50$) који представља број градова, а у свакој од наредних n линија налази се цео број t ($-20 \leq t \leq 40$) који представља температуру у одговарајућем граду.

Изназ: На стандардном излазу, у једној линији, приказати температуру у најхладнијем граду.

Пример

Улаз	Изназ
5	-13
10	-12
22	-13
15	

Решење

Одређивање екстремне вредности (минимума тј. максимума) разматрали смо у случају малих серија са фиксираним бројем елемената (на пример, у задатку **Најлошији контролни** одређивали смо најмањи од пет датих бројева).

У решењу овог задатка применићемо уобичајени алгоритам за одређивање минимума који смо тада описали, при чему ћемо, пошто број елемената серије није унапред фиксиран, елементе учитавати и обрађивати у петљи. Током извршавања алгоритма одржаваћемо вредност текућег минимума тј. најмању од свих до тада учитаних температура. Ту вредност можемо иницијализовати на први прочитани елемент серије тј. на температуру у првом граду. Затим, у сваком проласку кроз петљу читамо температуру у следећем граду и проверавамо да ли је њена вредност мања од текућег минимума. Ако јесте, ажурирамо вредност текућег минимума и постављамо га на температуру учитану у том кораку. На крају, када се учитају и обраде сви елементи серије, вредност текућег минимума биће уједно и вредност минимума целе серије тј. најнижа температура у свим градовима.

```
int n;           // број градова
cin >> n;
```

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

```
int t;          // temperatura u tekucem gradu
cin >> t;
int min_t = t; // najniza od svih do sada ucitanih temperatura
for (int i = 1; i < n; i++) { // ponavljamo za sve preostale gradove
    cin >> t;                // ucitavamo temperaturu
    if (t < min_t)          // ako je ona niza od prethodno najnize,
        min_t = t;         // azuriramo najnizu temperaturu
}
cout << min_t << endl;     // ispisujemo konacni rezultat
```

Уместо гранања могуће је у сваком кораку петље користити и функцију за одређивање минимума два броја. У језику C++ се може употребити функција `min` декларисана у заглављу `<algorithm>`.

```
int n;          // broj gradova
cin >> n;
int t;          // temperatura u tekucem gradu
cin >> t;
int min_t = t; // najniza od svih do sada ucitanih temperatura
// ponavljamo za sve preostale gradove
for (int i = 1; i < n; i++) {
    // ucitavamo temperaturu u narednom gradu
    cin >> t;
    // azuriramo vrednost najnize temperature
    min_t = min(min_t, t);
}
// ispisujemo konacni rezultat
cout << min_t << endl;
```

Иницијализација

У претходно описаном решењу температура у првом граду учитава се пре почетка петље, а у петљи се учитавају вредности температура у преосталих $n - 1$ градова. Издвајање обраде прве температуре испред петље можемо избећи тако што ћемо пре уласка у петљу текући минимум иницијализовати на вредност $+\infty$ која је сигурно већа од прве учитане вредности. Пошто ту вредност није могуће представити у програму, уместо ње можемо употребити максималан цео број који се може представити типом `int` (ова вредност има улогу $+\infty$). У језику C++ се та вредност може добити помоћу `numeric_limits<int>::max()` из заглавља `<limits>` или вредност `INT_MAX` из заглавља `<climits>`.

Већ код обраде температуре у првом граду текући ће минимум бити постављен на вредност температуре у том граду (јер је свака вредност мања или једнака од почетне).

```
int n;          // broj gradova
cin >> n;
// najnizu temperaturu inicijalizujemo na +beskonacno
int min_t = numeric_limits<int>::max();
for (int i = 0; i < n; i++) { // ponavljamo za sve gradove
    int t;                // ucitavamo temperaturu
    cin >> t;
    if (t < min_t)        // ako je ona niza od prethodno najnize,
        min_t = t;       // azuriramo najnizu temperaturu
}
cout << min_t << endl;     // ispisujemo konacni rezultat
```

Види групација решења овој задатку.

Задатак: Најтоплији дан

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види текстови задатка.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Пошто је седам прилично велики број, алгоритам за одређивање позиције максимума има пуно понављања (практично копирања и лепљења истих наредби, уз мале измене). Боље решење се може добити уз употребу петљи.

```
int t, rbr;           // temperatura i redni broj tekuceg dana
int maxT, rbrMax;    // temperatura i redni broj najtoplijeg dana

cin >> t; rbr = 1;    // ponedeljak
maxT = t; rbrMax = 1; // pretpostavljamo da je prvi dan najtopliji

for (int i = 2; i <= 7; i++) {
    cin >> t; rbr++;
    if (t > maxT) { // ako je dan topliji od prethodno najtoplijeg
        maxT = t; // azuriramo vrednost najtoplijeg dana
        rbrMax = i; // azuriramo redni broj najtoplijeg dana
    }
}

cout << rbrMax << endl; // ispisujemo redni broj najtoplijeg dana
```

Види групачија решења овој задатку.

Задатак: Просек скокова

Марко је велики обожаваатељ скокова у воду и не пропушта ниједно такмичење. Скок се оцењује тако што сваки од n судија донесе своју оцену, а затим се најслабија и најбоља оцена одбаце и одреди се просек преосталих оцена. Написати програм којим се одређује Марков резултат на основу судијских оцена.

Улаз: У n ($2 < n \leq 10$) линија стандардног улаза налази се по један цео број из интервала $[0, 10]$, ти бројеви представљају оцене које је Марко добио. Након тога долази крај улаза (он се може унети помоћу `ctrl + z` у системима Windows или `ctrl + d` у системима Linux).

Излаз: У првој линији стандардног излаза приказати, на две децимале, Марков резултат.

Пример

Улаз	Излаз
8	8.00
7	
9	
9	
5	

Решење

Минимум и максимум

Слично задатку [Најлошији контролни](#), резултат можемо добити тако што одредимо збир свих оцена, број унетих оцена, одредимо најмању и највећу оцену, па од добијеног збира одуземо најмању и највећи оцену и одредимо просек. Разлика је у томе што у овом задатку број оцена није фиксиран и за рачунање збира, минимума и максимума користимо петље (збир рачунамо исто као у задатку [Збир \$n\$ бројева](#), број као у задатку [Читање до краја улаза](#), а минимум и максимум аналогно задатку [Најнижа температура](#)).

```
// ucitavamo prvu ocenu
int ocena;
cin >> ocena;
// zbir ocena, najmanja i najveca ocena se inicijalizuju na prvu ocenu,
// a broj ocena na 1
int zbir, broj, min, max;
zbir = min = max = ocena;
broj = 1;
// obradjujemo ostale ocene
while (cin >> ocena) {
    // azuriramo broj, zbir, minimum i maksimum
```

```

број++;
збир += оцена;
if (оцена < min)
    min = оцена;
else if (оцена > max)
    max = оцена;
}
// збир уманјујемо за збир најмање и највеће оцене
збир -= min+max;
// израчунавамо просек преостале број-2 оцене
double просек = (double)збир / (double)(број-2);
// да би се добио реалан резултат, потребно је извршити конверзију
// типова исписујемо резултат
cout << fixed << setprecision(2) << showpoint << просек << endl;

```

Види групачија решења овој задатку.

Задатак: Најближи датом целом броју

Ученици једног одељења су организовали награду игру. Победник игре је онај ученик који што боље процени колико метара је дугачка њихова учионица. Ако су две процене такве да за исти износ потцењују тј. прецењују дужину учионице, победник је онај ко је дао већу процену.

Улаз: У првој линији стандардног улаза налази се цео број x ($10 \leq x \leq 20$), стварна дужина учионице, у другој линији природан број n ($3 \leq n \leq 30$) који представља број ученика, док следећих n линија садрже различите реалне бројеве x_i ($5 \leq x_i \leq 30$) који представљају процене ученика.

Изназ: У првој линији стандардног излаза приказати, највећи број међу учитаним реалним бројевима који је најближи броју x (заокружен на две децимале).

Пример

Улаз	Изназ
13.0	13.60
4	
12.4	
8.4	
13.6	
7.5	

Решење

Растојање између два броја једнако је њиховој апсолутној разлици. Две процене ученика се пореде тако што се прво упореде њихово растојање до тачне вредности x , а ако су оне једнаке, онда се пореде по величини. Приметимо да се овде ради о лексикографском поретку који смо увели и анализирали у задатку [Пунолетство](#). Број a представља бољу процену од броја b ако и само ако важи $|a - x| < |b - x|$ или ако је $|a - x| = |b - x|$ и $a > b$.

Дакле, потребно је пронаћи максималну вредност међу бројевима који се пореде не само на основу величине, већ на основу описане релације лексикографског поретка. То је могуће урадити модификацијом алгоритма за одређивања максимума тј. минимума серије бројева који смо приказали у задатку [Најнижа температура](#). Једина разлика је то што ћемо приликом поређења вредности тренутно читаног броја са тренутном вредношћу минимума, уместо обичне примене релације $<$ применити описани лексикографски поредак. Поредак можемо реализовати у засебној функцији.

```

// proverа da li je x1 bolja procena broja x nego broj x2
bool boljaProcena(double x1, double x2, double x) {
    if (abs(x1 - x) < abs(x2 - x))
        return true;
    if (abs(x1 - x) == abs(x2 - x) && x1 > x2)
        return true;
    return false;
}

```

```

int main() {
    double x; // dati broj
    cin >> x;
    int n;    // broj elemenata serije
    cin >> n;
    double broj; // učitavamo prvi element serije
    cin >> broj;
    double najbliziBroj = broj; // proglašavamo ga za najblizi broju x
    for (int i = 1; i < n; i++) { // za ostale elemente serije
        cin >> broj; // učitavamo naredni element
        // ako je on bolja procena od trenutno najblizeg
        if (boljaProcena(broj, najbliziBroj, x))
            najbliziBroj = broj; // azuriramo trenutno najblizi broj
    }
    // prijavljujemo rezultat
    cout << fixed << setprecision(2) << showpoint << najbliziBroj << endl;
    return 0;
}

```

Поређење можемо реализовати и једним сложеним логичким изразом којим се врши лексикографско поређење.

```

// ...
// učitavamo naredni element
cin >> broj;
double rastojanje = abs(x - broj);
// ako je on blizi broju x od trenutno najblizeg ili je na istom
// rastojanju od x kao trenutno najblizi, ali je veci od njega
if (rastojanje < najmanjeRastojanje ||
    (rastojanje == najmanjeRastojanje && broj > najbliziBroj)) {
    najbliziBroj = broj; // azuriramo trenutno najblizi broj
    najmanjeRastojanje = rastojanje; // azuriramo najmanje rastojanje
}

```

Задатак: Број максималних

На такмичењу из математике учествује n ученика. За сваког такмичара познат је број поена које је освојио на такмичењу. Написати програм којим се одређује број такмичара који су постигли најбољи резултат на такмичењу.

Улаз: Прва линија стандарног улаза садржи природан број n ($10 \leq n \leq 250$) који представља број ученика. У наредних n линија налази се по један цео број из интервала $[0, 100]$, ти бројеви представљају поене које су такмичари освојили.

Излаз: У првој линији стандарног излаза приказати укупан број такмичара који су постигли најбољи резултат на такмичењу.

Пример

Улаз	Излаз
5	2
57	
90	
68	
90	
73	

Решење

У овом задатку комбинује се одређивање максимумума серије елемената (што радимо на уобичајени начин, слично као у задатку [Најнижа температура](#)) и број појављивања неког елемента у серији (што радимо и у задатку [Бројање гласова за омиљеног глумца](#)).

Имплементација алгоритама за бројање појављивања максимума

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

Кључни увид који нам омогућава да задатак решимо само једним пролазом кроз елементе серије и да их не морамо памтити истовремено у меморији је то да се максимум у низу јавља онолико пута колико се јавља и у делу низа од првог свог појављивања па до краја. Дакле, када наиђемо на елемент који је максимум до тада обрађеног дела низа, он постаје нови кандидат за глобални максимум и од те тачке надаље бројимо његова појављивања (ако је строго већи од претходног кандидата за максимум, сигурни смо да се до ове тачке раније није јављао).

Да бисмо одредили број такмичара са најбољим резултатом морамо одредити и вредност најбољег резултата. Током рада програма ћемо у једној променљивој чувати максимум свих до сада учитаних поена такмичара, док ће друга променљива чувати број појављивања те највеће вредности међу до сада учитаним поенима такмичара. Максимум ћемо иницијализовати на вредност поена првог учитаног такмичара, док ћемо број његових појављивања иницијализовати на 1. Затим ћемо учитавати и анализирати поене осталих такмичара. Приликом анализе броја поена ученика могућа су три случаја:

- број поена је строго већи од текућег максимума, и у том случају се поставља нови текући максимум и његов број појављивања се поставља на 1;
- број поена је једнак текућем максимуму, и у том случају се његов број појављивања повећава за 1;
- број поена је мањи од текућег максимума, у ком случају се не предузима никаква акција.

```
// broj takmicara
int n;
cin >> n;
// poeni tekuceg takmicara
int poeni;
// ucitavamo poene prvog takmicara
cin >> poeni;
// najveci broj poena od poena takmicara koji su do sada ucitani
int maks = poeni;
// broj pojavljivanja vrednosti maks medju poenima koji su do sada
// ucitani
int brojPojavljivanjaMaksimuma = 1;
// za sve preostale takmicare
for (int i = 1; i < n; i++) {
    // ucitavamo poene narednog takmicara
    cin >> poeni;
    // ako je oni bolji od dosada najboljeg
    if (poeni > maks) {
        // azuriramo vrednost maksimuma
        maks = poeni;
        // ovo je prvo i za sada jedino pojavljivanje te vrednosti
        brojPojavljivanjaMaksimuma = 1;
    }
    // ako trenutni takmicar ima isti broj poena kao trenutni najbolji
    else if (poeni == maks)
        // uvecavamo broj pojavljivanja najveceg broja poena medju
        // trenutno ucitanim takmicarima
        brojPojavljivanjaMaksimuma++;
}
// prijavljujemo rezultat
cout << brojPojavljivanjaMaksimuma << endl;
```

Види групација решења овој задатку.

Задатак: Други на ранг листи

На основу резултата такмичењу на којем је учествовало N ученика формирана је ранг листа. Ранг листа се формира у нерастућем поретку по резултатима, од најбољег до најлошијег резултата. Написати програм којим се одређује број поена такмичара који је други на ранг листи.

Улаз: Прва линија стандардног улаза садржи природан број N ($1 \leq N \leq 50000$) који представља број

такмичара. У свакој од наредних N линија налази се цео број из интервала $[0, 50000]$, ти бројеви представљају резултате такмичара, који нису сортирани по поенима, већ по бројевима рачунара за којима су такмичари седели.

Излаз: У једној линији стандардног излаза приказати број поена другог такмичара на ранг листи.

Пример

Улаз	Излаз
5	95
80	
95	
75	
50	
95	

Решење

Задатак можемо решити тако што пратимо текућу највећу вредност и другу вредност на ранг листи међу учитаним бројевима (слично као у задатку *Друга вредност по величини*). Читамо један по један број и упоређујемо га са текућим вредностима прва два елемента.

- Ако је учитани број строго већи од вредности текућег максимума, друга вредност на ранг листи добија вредност текућег максимума, а текући максимум постаје број који смо читали.
- Иначе, ако је учитани број (строго) већи од текуће друге вредности на ранг листи, коригујемо другу вредност на ранг листи.

Остаје питање како иницијално поставити ове две вредности (наизглед није јасно које су вредности првог и другог елемента по реду у празном или једночланом скупу). Најприроднији начин је да се читају два елемента, да се упореде и да се на основу њиховог редоследа изврши иницијализација. Међутим, слично као и у случају одређивања једног максимума, можемо употребити иницијализацију на неку специјалну вредност која има улогу вредности $-\infty$. Пошто су учитани бројеви су из интервала $[0, 100]$, можемо иницијализовати обе вредности на -1 . Заиста, након читавања првог броја, он ће сигурно бити већи од -1 (вредности првог максимума), па ће вредност првог елемента на ранг-листи бити постављена на њега, а вредност другог елемента на ранг-лист на -1 . Након читавања другог елемента он ће бити или већи од првог елемента на ранг-листи и тада се вредност првог елемента на ранг-листи поставити на тај, други по реду учитани број, а вредност другог елемента на ранг-листи постаће једнака вредности првог читаног броја, или ће бити мањи од првог елемента на ранг-листи, али ће бити строго већи од -1 , па ће вредности првог и другог елемента на ранг листи бити редом први и други учитани број. Могуће је употребити и најмањи број који се може репрезентовати типом `int`. У језику C++ он се може добити функцијом `numeric_limits<int>::min()` декларисаном у заглављу `<limits>`.

```
// broj elemenata serije
int n;
cin >> n;

// prvi i drugi maksimum inicijalizujemo na -1
int prviMax, drugiMax;
prviMax = drugiMax = -1;

for (int i = 0; i < n; i++) {
    // učitavamo novi element serije
    int x;
    cin >> x;
    // ako je potrebno azuriramo vrednost maksimuma
    if (x > prviMax) {
        drugiMax = prviMax;
        prviMax = x;
    } else if (x > drugiMax) {
        drugiMax = x;
    }
}
```

```
// ispisujemo vrednost drugog maksimuma
cout << drugiMax << endl;
```

(Парцијално) сортирање

Овај задатак тражи да се одреди k -ти по величини у серији елемената (некада се овај проблем назива проблем ранговских статистика). Један начин да се то уради је да се елементи сместе у низ и сортирају (као у задатку **Сортирање бројева**) и да се онда испише k -ти по реду у сортираном низу. Сортирање целог низа је често вишак посла (да би се одредио други елемент по величини у сортираном низу, јасно је да није, на пример, потребно прецизно одредити међусобни редослед последња два елемента) и оно доводи до мало неефикаснијег решења. За решавање проблема ранговских статистика смишљени су алгоритми тзв. парцијалног сортирања низа (нпр. алгоритам QuickSelect о којем ће више речи бити у другом делу ове збирке). Поменимо само да у библиотеци језика C++ постоји функција `nth_element` која применом парцијалног сортирања проналази k -ти елемент по величини (она ће детаљније бити описана у другом делу збирке).

Задатак се може решити и дефинисањем рекурзивне функције која одређује највећа два елемента у низу. Ако је низ празан, резултат треба поставити на $(-\infty, -\infty)$, јер је $-\infty$ неутрални елемент за операцију максимума. Пошто су у задатку сви бројеви позитивни, уместо вредности $-\infty$ коју обично не подржавају програмски језици, можемо употребити $(-1, -1)$. Ако низ није празан, рекурзивно проналазимо први и други максимум првих $n - 1$ елемената низа и затим учитавамо последњи елемент. Ако је тај последњи елемент већи од првог максимума префикса, тада је он први максимум целог низа, а први максимум префикса је други максимум целог низа. У супротном, ако је последњи елемент низа већи од другог максимума префикса, тада је први максимум префикса уједно и први максимум целог низа, док је други максимум целог низа последњи елемент x . У супротном, први и други максимум целог низа се поклапају са првим и другим максимумом префикса.

```
#include <iostream>

using namespace std;

// ucitava n elemenata i odredjuje i vraca najveca dva
void dvaNajveca(int n, int& prviMaks, int& drugiMaks) {
    if (n == 0) {
        prviMaks = drugiMaks = -1;
        return;
    }
    dvaNajveca(n-1, prviMaks, drugiMaks);
    int x;
    cin >> x;
    if (x > prviMaks) {
        drugiMaks = prviMaks;
        prviMaks = x;
    } else if (x > drugiMaks)
        drugiMaks = x;
}

int main() {
    ios_base::sync_with_stdio(false);

    // broj elemenata serije
    int n;
    cin >> n;

    int prviMaks, drugiMaks;
    dvaNajveca(n, prviMaks, drugiMaks);

    // ispisujemo vrednost drugog maksimuma
    cout << drugiMaks << endl;

    return 0;
}
```

}

Задатак можемо решити и репном рекурзијом (коју неки компилатори аутоматски оптимизују и претварају у итерацију). Дефинисаћемо рекурзивну функцију која прима два највећа елемента у делу низа који је обрађен пре позива те функције и број n преосталих елемената које треба обрадити. Ако је $n = 0$ цео низ је обрађен и функција може да врати дотадашњих други елемент по величини. У супротном, читава се наредни елемент x и врши се рекурзивни позив за преосталих $n - 1$ елемената низа, при чему се по потреби ажурирају два до тада највећа елемента. Ако је читани елемент x већи од дотадашњег првог максимума, онда је он нови први максимум, а дотадашњи први максимум је нови други максимум. У супротном, ако је читани елемент x већи од дотадашњег другог максимума, тада се први максимум не мења, а елемент x постаје нови други максимум. У супротном се не мењају ни први ни други максимум.

```
#include <iostream>

using namespace std;

int drugi(int n, int prviMaks, int drugiMaks) {
    if (n == 0)
        return drugiMaks;
    int x;
    cin >> x;
    if (x > prviMaks)
        return drugi(n-1, x, prviMaks);
    else if (x > drugiMaks)
        return drugi(n-1, prviMaks, x);
    else
        return drugi(n-1, prviMaks, drugiMaks);
}

int drugi(int n) {
    return drugi(n, -1, -1);
}

int main() {
    ios_base::sync_with_stdio(false);

    // broj elemenata serije
    int n;
    cin >> n;

    // ispisujemo vrednost drugog maksimuma
    cout << drugi(n) << endl;

    return 0;
}
```

Задатак: Друга вредност по величини

У студентском удружењу се организује наградна игра. Договор је да награде добијају два члана удружења која имају највеће бројеве индекса. Проблем је у томе што су на постојећем списку неки студенти уписани и више пута. Написати програм који на основу серије бројева индекса са тог списка одређује два награђена студента.

Улаз: Прва линија стандардног улаза садржи природан број N ($10 \leq N \leq 50000$) који представља број студената на списку. Свака од наредних N линија садржи по један природан број из интервала $[1, 50000]$, који представља број индекса студената из удружења. Може се претпоставити да ће на списку постојати бар два различита индекса.

Издаз: На стандардном излазу приказати индексе два награђена студента. У првом реду највећу, а у другом реду следећу по величини вредност међу бројевима индекса.

Пример

Улаз	Израз
5	71
22	35
35	
71	
71	
22	

Решење

Сортирање

Одређивање другог по величини у серији од n целих бројева, можемо решити уређивањем бројева од највећег до најмањег. После уређења бројева највећи број је број x_1 , а други по величини је следећи број из уређене серије који је различит од x_1 . Ако такав број не постоји сви бројеви су једнаки. За разлику од задатка **Други на ранг листи** у овом случају до решења није могуће доћи једноставном применом парцијалног сортирања, јер не знамо на ком месту се у сортираном редоследу налази елемент који тражимо. Подсетимо се, у језику C++ парцијално сортирање можемо извршити функцијом `partial_sort`, а n -ти елемент се може наћи функцијом `nth_element`.

Алгоритам одређивања два максимума

Задатак можемо решити модификацијом класичног поступка одређивања максималног елемента серије (који је описан у задатку **Најнижа температура**), слично као у задатку **Други на ранг листи**. Претпоставимо опет да знамо прву и другу вредност по величини (назовимо их првим и другим максимумом) у до сада обрађеном делу низа и размотримо како их ажурирати када се прочита нови број.

- Ако је учитани број строго већи од текуће вредности првог максимума, други максимум треба поставити на текућу вредност првог максимума, а први максимум треба поставити на учитани број.
- Ако је учитани број мањи од текуће вредности првог максимума потребно га је упоредити са другим максимумом и ако је (строго) већи од текуће вредности другог максимума, потребно је други максимум поставити на учитани број.
- Ако је учитани број једнак текућој вредности првог максимума, не радимо ништа.

Поново је проблематично питање иницијализације првог и другог максимума.

Почетну вредност оба максимума можемо поставити на неку вештачку вредност која има улогу вредности $-\infty$ (то је обично најмања вредност целобројног типа, а у нашем задатку, пошто је допуштени интервал вредности $[1, 1000]$, може се употребити 0 или -1). Уз овакву иницијализацију, све елементе можемо обрадити на начин који смо на почетку описали. После првог учитаног броја догодиће се да је он сигурно већи од првог максимума и тада ће вредност другог максимума постати стара вредност првог максимума (то је број који представља $-\infty$), док ће први максимум бити постављен на учитани број. Наредна промена ће се десити када се прочита први елемент у серији који је различит од првог. Ако је строго већи од првог максимума, први максимум ће бити постављен на вредност тог броја, а други максимум на вредност првог учитаног броја, што су управо коректне иницијалне вредности за део серије који је до сада читан. Ако је учитани елемент строго мањи од првог максимума, тада ће он остати непромењен, али ће други максимум бити постављен на тај учитани број и опет ће те две вредности бити коректно иницијализоване у односу на део серије који је до сада читан.

```
// broj elemenata serije
int n;
cin >> n;

// maksimume inicijalizujemo na broj koji simulira vrednost
// -beskonacno
int prviMax = numeric_limits<int>::min();
int drugiMax = numeric_limits<int>::min();
for (int i = 0; i < n; i++) {
    // učitavamo naredni element serije
    int x;
    cin >> x;
    // ako je potrebno azuriramo vrednosti maksimuma
```



```

if (x > prviMax) {
    drugiMax = prviMax;
    prviMax = x;
} else if (x < prviMax) {
    if (x > drugiMax)
        drugiMax = x;
} // else if (x == prviMax) ne radimo nista
}

// ispisujemo vrednosti prvog i drugog maksimuma
cout << prviMax << endl << drugiMax << endl;

```

Задатак: Победник у три дисциплине

Такмичари су радили тестове из програмирања, математике и физике. За сваки предмет добили су одређени број поена (цео број од 0 до 50). Такмичари се рангирају по укупном броју поена из сва три предмета. Ако два такмичара имају исти број поена, победник је онај који има више поена из програмирања. Ако је и број поена из програмирања једнак, онда се посматра број поена из математике. На крају, ако је и број поена из математике једнак, посматра се број поена из физике. Потребно је написати програм који одређује победника такмичења.

Улаз: Учитава се прво број такмичара, а затим и подаци за сваког такмичара. За сваког такмичара учитава се број поена из програмирања, а затим број поена из математике, и на крају број поена из физике, сви у истом реду.

Излаз: Потребно је исписати редни број победника и број његових поена из сваког предмета. Бројање такмичара почиње од 1. Ако су два или више такмичара остварила потпуно идентичан успех, победник је онај који има мањи редни број (јер је остварио више поена на квалификационом такмичењу).

Пример

Улаз	Излаз
4	3: 20 40 30
20 30 40	
10 20 30	
20 40 30	
10 50 20	

Решење

У овом задатку је потребно одредити максимум серије елемената и његову позицију у серији, при чему су ти елементи сложеног типа (поени из три предмета) и релација поретка над њима је дефинисана на начин дефинисан условима задатка. Када мало боље размотримо дефиницију поретка између такмичара приметимо да је у питању лексикографски поредак, сличан ономе коришћеном у задацима [Пунолетство](#) и [Најбољи у две дисциплине](#).

Приметимо да за одређивање победника није неопходно истовремено памтити податке о свим такмичарима, већ је могуће истовремено учитавати податке и одређивати максимум применом уобичајеног алгорита за одређивања максимума тј. минимума (описаног у задатку [Најнижа температура](#)).

Једно од основних питања у имплементацији решења је питање како представити податке о такмичару.

Посебне променљиве

Најелементарније решење је оно у којем се подаци представљају помоћу три посебне променљиве (по једна за поене из свака три предмета). У том случају дефинишемо функцију која прима 6 параметара (податке о два такмичара) и која проверава да ли је један такмичар бољи од другог (тј. да ли је други лошији од првог). Функција два такмичара пореди лексикографски, проверавајући редом један по један критеријум (прво укупан број поена, а онда, ако је укупан број поена једна, број поена из програмирања, затим математике и на крају из физике). Сличне функције приказане су и у задацима [Пунолетство](#) и [Најбољи у две дисциплине](#). Када је дефинисана релација (тј. буловска функција) поретка, победник се одређује алгоритмом за одређивање максимума и његове позиције, који је приказан у задацима [Најтоплији дан](#) и [Редни број максимума](#). Памтићемо број поена и редни број победника, прогласићемо првог такмичара за победника,

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

а затим редом обрађивати остале такмичаре, и ако је дотадашњи кандидат за победника лошији од текућег такмичара, ажурираћемо податке о победнику.

```
// proverava da li je takmicar A losiji od takmicara B
bool losiji(int progA, int matA, int fizA,
            int progB, int matB, int fizB) {
    // Prvo se poredi ukupan broj poena
    int ukupnoA = progA + matA + fizA,
        ukupnoB = progB + matB + fizB;
    if (ukupnoA < ukupnoB)
        return true;
    if (ukupnoA > ukupnoB)
        return false;
    // ukupan broj poena je jednak, pa se porede poeni iz programiranja
    if (progA < progB)
        return true;
    if (progA > progB)
        return false;
    // i ukupan broj poena iz programiranja je jednak,
    // pa se porede poeni iz matematike
    if (matA < matB)
        return true;
    if (matA > matB)
        return false;
    // i ukupan broj poena iz matematika je jednak,
    // pa se porede poeni iz fizike
    return fizA < fizB;
}

void ucitajTakmicara(int& prog, int& mat, int& fiz) {
    cin >> prog >> mat >> fiz;
}

void prikaziTakmicara(int prog, int mat, int fiz) {
    cout << prog << " " << mat << " " << fiz;
}

int main() {
    // ucitavamo broj takmicara
    int brojTakmicara;
    cin >> brojTakmicara;

    int prog, mat, fiz; // poeni tekuceg takmicar

    int progPobednik, matPobednik, fizPobednik; // poeni pobednika
    int brojPobednika; // redni broj pobednika

    // ucitavamo podatke i istovremeno odredjujemo poziciju najveceg
    // elementa, kada se poredjenje vrsi u odnosu na relaciju "losiji"

    // ucitavamo poene prvog takmicara
    ucitajTakmicara(prog, mat, fiz);
    // pretpostavljamo da je prvi takmicar pobednik
    progPobednik = prog; matPobednik = mat; fizPobednik = fiz;
    brojPobednika = 1;
    for (int i = 2; i <= brojTakmicara; i++) {
        // ucitavamo poene tekuceg takmicara
        ucitajTakmicara(prog, mat, fiz);
        // ako je dosadasnji pobednik losiji od tekuceg takmicara
```

```

    if (losiji(progPobednik, matPobednik, fizPobednik,
              prog, mat, fiz)) {
        // za pobednika proglasavamo tekuceg takmicara
        progPobednik = prog; matPobednik = mat; fizPobednik = fiz;
        brojPobednika = i;
    }
}
// prijavljujemo rezultat
cout << brojPobednika << ": ";
prikaziTakmicara(progPobednik, matPobednik, fizPobednik);
cout << endl;
return 0;
}

```

Види групачија решења овој задатку.

Задатак: Најмањи круг

За дати низ тачака у равни одредити полупречник најмањег круга, са центром у координатном почетку, који садржи све дате тачке.

Улаз: Са стандардног улаза у првој линији се уноси природан број n ($1 \leq n \leq 100$) који представља број тачака у равни, а у наредних n линија у свакој линији по два реалана броја (од -1000 до 1000), који представљају координате x и y тачке у равни.

Излаз: На стандардни излаз исписати један реалан број (допуштена је толеранција грешке 10^{-5}) - полупречник најмањег круга са центром у координатном почетку, који садржи све учитане тачке.

Пример

Улаз	Излаз
4	6.80661
2.6 3.4	
1.2 6.7	
3.33 5.55	
2.57 3.44	

Задатак: Редни број максимума

На аукцији неке слике учествује n купаца. Сваки купац је понудио извесни износ новца и сви купци су понудили различите износе. Написати програм којим се одређује редни број купца који је понудио највећи износ.

Улаз: Прва линија стандардног улаза садржи природан број n ($1 \leq n \leq 1000$) који представља број купаца. У наредних n линија налази се по један позитиван реалан број, ти бројеви представљају износе новца коју су понудио купци редом.

Излаз: У првој линији стандардног излаза приказати редни број купца који је понудио највећи износ новца.

Пример

Улаз	Излаз
5	4
123.23	
234.45	
100.23	
345.00	
128.80	

3.2.8 Линеарна претрага

Претрага подразумева испитивање да ли постоји елемент серије који задовољава неки дати услов. Такође, можемо проверавати да ли сви елементи задовољавају услов, одређивати позицију првог или последњег елемента који задовољава услов и слично. Поред самог алгоритма линеарне претраге приказаћемо и његову примену у решавању неколико конкретних задатака.

Задатак: Негативан број

Написати програм којим се проверава да ли међу учитаним бројевима постоји негативан број.

Улаз: Прва линија стандарног улаза садржи природан број n ($1 \leq n \leq 100$) који представља број бројева. Свака од наредних n линија садржи по један реалан број.

Излаз: На стандарном излазу у једној линији исписти да ако међу учитаним бројевима постоји негативан број, иначе исписати не.

Пример 1

Улаз	Излаз
5	da
10.89	
12.349	
-5.9	
2.3	
-2.45	

Пример 2

Улаз	Излаз
4	ne
100.89	
12.349	
0	
2.3	

Решење

Коришћење логичке променљиве

Потребно је испитати да ли међу учитаним бројевима постоји неки који је негативан. Логичком променљивом можемо пратити да ли смо у читали неки негативан број. Пошто на почетку нисмо још у читали ни један број, нисмо видели ни један негативан, и на почетку вредност логичке променљиве поставимо на `false`. Затим у петљи редом у читавамо бројеве и за сваки број проверавамо да ли је негативан. Ако је тренутно у читани број негативан вредност логичке променљиве поставимо на `true` (јер сада знамо да међу до сада у читаним бројевима постоји један који је негативан). После читања свих бројева, на основу вредности логичке променљиве исписујемо одговарајућу поруку.

```
postoji = false;
for (int i = 0; i < n; i++)
{
    // ucitaj x
    if (x < 0)
        postoji = true;
}
```

Једна честа грешка

Нагласимо да је честа грешка да се унутар петље провери да ли је тренутно у читани број негативан и да се ако јесте вредност логичке променљиве постави на `true`, а да се у супротном она постави на `false`.

```
postoji = false;
for (int i = 0; i < n; i++)
{
    // ucitaj x
    if (x < 0)
        postoji = true;
    else
        postoji = false;
}
```

или (са истим ефектом)

```
postoji = false;
for (int i = 0; i < n; i++)
{
    // ucitaj x
    postoji = (x < 0);
}
```

Наилазак на негативан број одређује дефинитивно да у серији постоје негативни бројеви, док наилазак на ненегативни број не указује на то да су у серији сви бројеви ненегативни (зато грану `else` не треба писати). Исправно решење може бити следеће.

```

bool postojiNegativan = false; // da li medju ucitanim postoji negativan
int n; // obradjujemo n elemenata
cin >> n;
for (int i = 0; i < n; i++) {
    double x; // ucitavamo element
    cin >> x;
    if (x < 0) // ako je on negativan
        postojiNegativan = true; // onda medju ucitanim postoji negativan
}
cout << (postojiNegativan ? "da" : "ne") << endl; // ispisujemo rezultat

```

Рани прекид

Приметимо да читање бројева можемо прекинути када прочитамо први негативан број тј. када вредност логичке променљиве постане true, јер тада знамо да међу бројевима постоји негативан број. Према томе читамо бројеве и проверавамо да ли је учитани број негативан, док не прочитамо све бројеве и док је вредност логичке променљиве false. Нагласимо да овакав ранији прекид петље не би био могућ да је у неком случају било потребно учитати све бројеве. Рани прекид петље можемо остварити на различите начине.

Један је да се услов петље ојача условом да је вредност логичке променљиве једнака false (на пример, да се тражи да је `i < n && !postoji`).

```

bool postojiNegativan = false; // da li medju ucitanim postoji negativan
int n; // obradjujemo n elemenata
cin >> n;
// dok nisu obradjeni svi elementi i dok nije pronadjen negativan
for (int i = 0; i < n && !postojiNegativan; i++) {
    double x; // ucitavamo element
    cin >> x;
    if (x < 0) // ako je on negativan
        postojiNegativan = true; // onda medju ucitanim postoji negativan
}
cout << (postojiNegativan ? "da" : "ne") << endl; // ispisujemo rezultat

```

Један начин да се оствари рани прекид приликом постављања логичке променљиве на true је да се употреби наредба break.

```

bool postojiNegativan = false; // da li medju ucitanim postoji negativan
int n; // obradjujemo n elemenata
cin >> n;
for (int i = 0; i < n; i++) {
    double x; // citamo element
    cin >> x;
    if (x < 0) { // ako je on negativan
        postojiNegativan = true; // onda medju ucitanim postoji negativan
        break; // mozemo prekinuti ispitivanje ostalih
    }
}
cout << (postojiNegativan ? "da" : "ne") << endl; // ispisujemo rezultat

```

Ако би се читавање и провера вршили у засебној функцији, тада би се приликом наилажења на негативан елемент могла употребити и наредба return true (тада не би било потребе за посебном логичком променљивом), док би се након петље употребила наредба return false.

```

// provera da li medju n ucitanih elemenata postoji negativan
bool postojiNegativan() {
    int n; // ukupan broj elemenata
    cin >> n;
    for (int i = 0; i < n; i++) { // obradjujemo n elemenata
        double x; // ucitavamo element
        cin >> x;
        if(x < 0) // ako je on negativan

```

```

    return true;           // medju ucitanim postoji negativan
}
return false;           // medju ucitanim ne postoji negativan
}

int main() {
    if (postojiNegativan())
        cout << "da" << endl;
    else
        cout << "ne" << endl;
    return 0;
}

```

Задатак можемо решити и без коришћења логичке променљиве. У петљи читамо број по број, и ако је читани број негативан прекинемо извршавање петље коришћењем наредбе `break`. После завршетка петље на основу вредности бројачке променљиве можемо закључити да ли је међу читаним бројевима постојао негативан број. Ако је бројачка променљива имала почетну вредност 0 и ако је после завршетка петље њена вредност мања од n онда је петља завршена пре читања свих бројева, према томе постоји негативан број међу читаним бројевима. Слично ако је почетна вредност бројачке променљиве 1 и ако је после завршетка петље њена вредност мања или једнака n , постојао је негативан број међу читаним бројевима. Важно је напоменути да бројачка променљива мора бити декларисана пре петље (не сме бити локална променљива петље), да бисмо је могли користити после петље.

```

int n;           // ukupan broj elemenata
cin >> n;
int i;           // broj ucitanih elemenata
for (i = 0; i < n; i++) {
    double x;    // ucitavamo element
    cin >> x;
    if (x < 0)   // ako je negativan
        break;  // prekidamo petlju
}
// negativan postoji ako je petlja prekinuta pre obrade svih elemenata
cout << (i < n ? "da" : "ne") << endl;

```

Задатак можемо решити и дефинисањем рекурзивне функције која читава n бројева и проверава да ли међу њима постоји неки негативан. Ако је $n = 0$, тада нема бројева, па самим тим нема ни негативних. У супротном читавамо први број. Негативних бројева у низу има ако и само ако је тај први број негативан или ако има негативних бројева међу наредних $n - 1$ елемената низа (што можемо установити рекурзивним позивом).

Види групаџија решења овој задајџка.

Задатак: Дељив бројевима од 1 до n

Написати програм којим се проверава да ли је дати природан број k дељив свим природним бројевима од 1 до n .

Улаз: Прва линија стандардног улаза садржи природан број k ($0 < k \leq 10^9$), а друга линија садржи природан број n ($1 \leq n \leq 10$).

Издаз: У првој линији стандардног излаза приказати реч `да` ако је број k дељив свим природним бројевима од 1 до n , иначе приказати реч `не`.

Пример 1		Пример 2	
Улаз	Издаз	Улаз	Издаз
60	да	90	не
6		5	

Решење

За сваки број од 1 до n треба проверити да ли дели дати број k .

Задатак се решава линеарном претрагом, на веома сличан начин као у задатку **Негативан број**. Разлика је то што је тамо проверавамо да ли у серији постоји неки елемент који задовољава дати услов, док се у овом

задатку тражи да се провери да ли сви елементи у серији задовољавају дати услов. То је, међутим, веома слично, јер се испитивање да ли сви елементи задовољавају неки услов тривијално своди на испитивање да ли постоји неки елемент који не задовољава тај услов.

У петљи обилазимо серију делилаца тј. имплементирамо бројачку петљу у којој бројачка променљива узима вредности од 1 до n . Можемо увести логичку променљиву `deljiv` која има вредност `true` ако је број k дељив свим бројевима који су у тој петљи обрађени, а иначе има вредност `false`. На почетку, пре уласка у петљу, вредност променљиве `deljiv` иницијализујемо на `true` (јер нисмо обрадили још ни један број, па је k сигурно дељив свим бројевима из тог празног скупа). Редом за сваки број од 2 до n проверавамо да ли је делилац броја k (број 1 нема потребе проверавати јер је k сигурно дељив њиме) и ако није, онда постављамо вредност променљиве `deljiv` на `false` (јер смо пронашли број између 1 и n којим број k није дељив). У том тренутку можемо и прекинути петљу (јер већ знамо да број k није дељив свим бројевима од 1 до n , па нема потребе анализирати остале делиоце). Прекид је могуће остварити наредбом `break`.

```
int n, k;
cin >> k >> n;
bool deljiv = true; // da li je k deljiv sa svim brojevima od 1 do n
for (int d = 2; d <= n; d++) // proveravamo sve delioce od 2 do n
    if (k % d != 0) { // ako k nije deljiv sa d
        deljiv = false; // onda nije deljiv sa svim brojevima
        break;
    }
cout << (deljiv ? "da" : "ne") << endl; // ispisujemo rezultat
```

Ако не желимо да користимо `break`, петља се може прекинути тако што се у услову петље тражи да да променљива `deljiv` чува тачну вредност.

Још једноставније решење добијамо ако напишемо функцију којом се проверава дељивост броја. У функцији коришћењем петље проверавамо да ли је број k дељив редом бројевима од 1 до n , ако није дељив неким од тих бројева прекидамо извршавање функције и враћамо `false`. На крају после завршетка петље, враћамо `true`, до чега ће доћи само ако је број k био дељив свим бројевима од 1 до n .

```
// provera da li je k deljiv svim brojevima od 1 do n
bool deljivOd1Don(int k, int n) {
    for (int d = 2; d <= n; d++) // ispitujemo sve delioce od 1 do n
        if (k % d != 0) // ako k nije deljiv sa d
            return false; // nije deljiv sa s svim brojevima
    return true; // k jeste deljiv sa svim brojevima
}

int main() {
    int n, k;
    cin >> k >> n;
    if (deljivOd1Don(k,n))
        cout << "da" << endl;
    else
        cout << "ne" << endl;
    return 0;
}
```

Види грујачија решења овој задатку.

Задатак: Прва негативна температура

Познате су температуре за сваки дан неког периода. Написати програм којим се одређује редни број дана у том периоду када је температура први пут била негативна.

Улаз: Прва линија стандарног улаза садржи природан број n ($3 \leq n \leq 365$) који представља број дана периода. Свака од наредних n линија садржи по један цео број из интервала $[-5, 40]$, бројеви представљају температуре редом за n дана периода.

Израз: У првој линији стандарног излаза приказати редни број дана периода када је температура први пут била негативна (дани се броје од 1 до n), ако такав дан не постоји приказати -1.

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

Пример 1

Улаз	Изназ
5	3
12	
10	
-2	
-3	
4	

Пример 2

Улаз	Изназ
3	-1
12	
10	
14	

Решење

У задатку **Негативан број** питали смо се да ли међу бројевима постоји број који је негативан, а у овом задатку тражимо мало више од тога - тражимо да одредимо прву позицију на којој се јавља неки негативан број. Решење опет можемо решити алгоритмом линеарне претраге, који можемо имплементирати на било који од начина приказаних у задатку **Негативан број**.

Уместо логичке променљиве којом региструјемо да ли се међу учитаним бројевима налази неки негативан, користимо целобројну променљиву којом ћемо регистровати редни број (позицију тј. индекс) дана у којем се јавила негативна температура. Вредност те променљиве иницијализујемо на -1 што указује да се до тог тренутка још није читала негативна вредност. Након тога, учитавамо температуре једну по једну и ако је тренутна температура негативна, региструјемо редни број дана (то ће бити вредност бројачке променљиве у петљи, чије су вредности од 1 до n) и прекидамо петљу. На крају, штампамо променљиве у којој се региструје редни број (ако се није наишло на негативну температуру, она и даље има вредност -1).

```
int n;          // ukupan broj dana
cin >> n;
int dan = -1;  // redni broj dana sa negativnom temperaturom
               // u pocetku pretpostavljamo da takav dan ne postoji
for (int i = 1; i <= n; i++) { // obradjujemo redom sve dane
    int temperatura;          // ucitavamo temperaturu
    cin >> temperatura;
    if (temperatura < 0) {    // ako je ona negativna
        dan = i;              // pamtimo redni broj dana
        break;                // prekidamo petlju
    }
}
cout << dan << endl; // ispisujemo redni broj dana
```

Могуће је избећи коришћење помоћне променљиве. Редом читамо температуру по температуру, при томе бројачком променљивом пратимо редни број текућег дана периода (почињемо бројање од 1). Приликом наилазак на негативну температуру, петљу прекидамо наредбом `break`. Након петље проверавамо да је услов петље испуњен (да ли је вредност бројачке променљиве мања или једнака броју дана периода n) и ако јесте, значи да је дошло до раног прекида и тада вредност бројачке променљиве садржи тражени број дана. У супротном можемо да прикажемо -1, јер се дошло до краја петље и није извршен рани прекид, што значи да међу учитаним температурама није постојала негативна.

```
int n;          // ukupan broj dana
cin >> n;
// analiziramo redom dane od 1 do n
int i;
for (i = 1; i <= n; i++) {
    int temperatura; // ucitavamo temperaturu za dan broj i
    cin >> temperatura;
    if (temperatura < 0) // ako je ona negativna
        break;          // prekida se petlja
}
if (i <= n)             // ako je uslov petlje ispunjen
    cout << i << endl; // petlja je prekinuta u koraku i
else                    // u suprotnom
    cout << -1 << endl; // dosli smo do kraja i nije pronadjena
                       // negativna temperatura
```


Види групачија решења овој задатку.

Задатак: Последња негативна температура

Познате су температуре за сваки дан неког периода. Написати програм којим се одређује редни број дана у том периоду када је температура последњи пут била негативна (дани се броје од 1 до n).

Улаз: Прва линија стандардног улаза садржи природан број n ($3 \leq n \leq 365$) који представља број дана периода. Свака од наредних n линија садржи по један цео број из интервала $[-10, 40]$, бројеви представљају температуре редом за n дана периода.

Издаз: У првој линији стандардног излаза приказати редни број дана периода када је температура последњи пут била негативна, ако такав дан не постоји приказати -1.

Пример 1		Пример 2	
Улаз	Издаз	Улаз	Издаз
5	4	3	-1
2		12	
-8		10	
-2		14	
-3			
4			

Решење

Решење овог задатка је веома слично решењу задатка **Прва негативна температура**. И овај пут користимо алгоритам линеарне претраге, са веома сличном имплементацијом. Једина разлика је то што се након проналажења прве негативне температуре петља не прекида, већ је увек неопходно учитати и обрадити све температуре. Целобројном променљивом пратимо редни број дана када је температура последњи пут међу учитаним температурама била негативна. На почетку вредност те променљиве је -1 , јер још није учитана ни једна негативна температура. У петљи чија бројачка променљива узима вредности од 1 до n тј. која представља редни број дана учитавамо редом температуре за сваки дан. За сваку учитану температуру проверавамо да ли је негативна и ако јесте коригујемо вредност променљиве којом памтимо редни број дана када је температура последњи пут била негативна. Корекцију вршимо постављањем вредности те променљиве на вредност бројачке променљиве (јер је то редни број дана периода у којем је била управо прочитана температура).

```
int n;           // број дана који се анализирају
cin >> n;
int dan = -1;   // редни број дана са негативном температуром
for (int i = 1; i <= n; i++) { // обрађујемо n дана
    int temperatura;         // учитавамо температуру
    cin >> temperatura;
    if (temperatura < 0)     // ако је негативна
        dan = i;           // азурирамо дан и настављамо петљу
}
cout << dan << endl;       // исписујемо последњи дан
```

Види групачија решења овој задатку.

Задатак: Парно непарни

Пера се игра картама. Све карте које је имао у руци је сложио тако да прво иду све карте са парним бројевима, а затим оне са непарним бројевима (могуће је и да је Пера имао само парне или само непарне карте). Напиши програм који проверава да ли је Пера исправно сложио карте.

Улаз: Са стандардног улаза учитавају се бројеви карата (природни бројеви између 2 и 10) све док се не дође до краја улаза (он се може унети са $\text{Ctrl}+\text{Z}$ тј. $\text{Ctrl}+\text{D}$). Карата има најмање две, а највише десет.

Издаз: На стандардни излаз исписати да ако је Пера добро сложио карте тј. не у супротном.

Пример 1

Улаз	Издаз
2	da
6	
4	
5	
3	

Пример 2

Улаз	Издаз
2	ne
6	
3	
5	
4	

Решење

Промена стања

Током провере наш програм се налази у два стања. У почетку смо у стању у којем прихватамо парне бројеве. Када се у том стању први пут појави неки непаран број прелазимо у стање у којем надаље смео да прихватамо само непарне бројеве. Ако се у том стању појави неки паран број, онда низ не задовољава тражени услов. Тренутно стање се може регистровати једном логичком променљивом (на пример, променљива `parni` која ће бити иницијализована на `false` и постављена на `true` први пут када се прочита непаран број). Дакле, у петљи учитавамо број по број све док не дођемо до краја улаза (начине да се то уради смо видели у задатку [Читање до краја улаза](#)). Логичком променљивом `OK` коју иницијализујемо на `true` бележимо да ли бројеви задовољавају услов задатка. У услову петље захтевамо да је вредност променљиве `OK` тачно, јер у супротном знамо да бројеви не задовољавају тражени распоред и нема потребе вршити даље провере. У телу петље, након учитавања наредног броја проверавамо да ли смо наишли на непаран број у тренутку док променљива `parni` има вредност `false`. Када се то догоди вредност променљиве `parni` постављамо на `true`. Такође, проверавамо да ли је прочитан паран број док је вредност променљиве `parni` једнака `true` и у том случају променљиву `OK` постављамо на `false` (јер знамо да се појавио паран број након непарног). На крају резултат пријављујемо у зависности од вредности `OK`.

Приметимо да се овде заправо у две фазе примењује алгоритам линеарне претраге, који смо приказали у задатку [Негативан број](#). У првом стању се врши претрага за непарним бројем и када се он нађе, прелази се у друго стање. У другом стању се врши претрага за парним бројем и ако се он нађе, пријављује се да низ не задовољава услов.

```
bool OK = true; // da li je serija za sada ispravna
bool parni = false; // da li smo u presli u deo sa neparnim brojevima
int x; // tekuci element
while (cin >> x && OK) { // ucitavamo brojeve do kraja ulaza i dok
    // je procitani deo ispravno slozen
    if (x % 2 != 0) // ako je broj neparan
        parni = true; // prelazimo na citanje neparnih
    else
        if (parni) // paran broj se pojavio u neparnom delu niza
            OK = false; // konstatujemo da je doslo do greske
}
cout << (OK ? "da" : "ne") << endl; // ispisujemo rezultat
```

Види групаџија решења овог задатка.

Задатак: Први и последњи приступ

Са рачунара у школи на сајт школе се логују разни ђаци. Сваки пут када се неко улогује то се памти у бази. Напиши програм који за дати број рачунара одређује који је први, а који је последњи ђак који се логовао.

Улаз: Са стандардног улаза се у првој линији уноси редни број рачунара који нас занима (цео број од 1 до 100), у другој линији се налази укупан број n ($1 \leq n \leq 50000$) логовања, а затим се у наредних n линија налазе подаци о појединачним логовањима: број рачунара са којег се неки ђак логовао (цео број од 1 до 100) и корисничко име тог ђака (ниска од највише 20 малих слова енглеске абетецеде).

Издаз: На стандардни издаз исписати име првог и име последњег ђака који се логовао на тај рачунар (свако у посебном реду) или текст `neма` ако се са тог рачунара нико није логовао.

Пример

Улаз	Излаз
3	jovana
5	milica
1 zika	
3 jovana	
2 ana	
3 pega	
3 milica	

Задатак: Провера тробојке

Напиши програм који проверава да ли су бројеви који се уносе уређени тако да прво иду негативни, затим нуле и на крају позитивни бројеви (могуће је и да било којих од наведених бројева нема).

Улаз: Са стандардног улаза се уноси број бројева n ($1 \leq n \leq 20$), а након тога и сами бројеви, сваки у посебном реду.

Излаз: На стандардни излаз исписати да ако су бројеви уређени како је наведено тј. не у супротном.

Пример

Улаз	Излаз
5	da
-3	
-1	
0	
4	
2	

3.2.9 Позициони запис броја

У поглављу о малим серијама видели смо како итеративним алгоритмом можемо одредити цифре броја (сдесна налево) и како од цифара броја можемо одредити вредност броја (било да оне долазе слева надесно или сдесна налево). У овом поглављу ћемо те алгоритме уопштити на веће бројеве тј. бројеве чији се број цифара не зна унапред (стога ћемо у имплементацији користити условне петље).

У наредним задацима алгоритме за рад са цифрама у позиционом запису комбиноваћемо са раније приказаним алгоритмима за обраду бројевних серија (израчунавање статистика, претрага, пресликавање и слично).

Задатак: Број и збир цифара броја

Написати програм којим се одређује број и збир цифара декадног записа природног броја.

Улаз: Са стандардног улаза се учитава цео број од 0 до 1000000000.

Излаз: На стандардни излаз се исписују број цифара и збир цифара читаног броја.

Пример 1

Улаз	Излаз
23645	5 20

Пример 2

Улаз	Излаз
0	1 0

Решење

Серију цифара у декадном запису броја (здесна налево) можемо одредити операцијом целобројног дељења са 10, као што смо то видели, на пример, у задатку [Збир цифара четвороцифреног броја](#). Последња декадна цифра броја одређује се као остатак при дељењу са 10, а уклања се када се израчуна целобројни количник са 10. Подсетимо се, један начин да одредимо последње три цифре броја био је следећи итеративни поступак.

```
c0 = n mod 10; n = n div 10;
```

```
c1 = n mod 10; n = n div 10;
```

```
c2 = n mod 10; n = n div 10;
```

При том, ако је почетни број био троцифрен, тада ће вредност променљиве n након извршавања овог фрагмента кода бити 0.

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

За разлику од ранијих задатака у којима је број цифара био фиксиран (и обично мали), у овом задатку није унапред познат број цифара броја. Зато је цифре броја потребно одређивати у петљи. У сваком кораку текућу цифру одређујемо као $n \bmod 10$, а затим је уклањамо тако што вредност променљиве n замењујемо вредношћу $n \div 10$. Петља се завршава када се уклоне све цифре и када вредност променљиве n постане нула. Да би се и у случају када је број n једнак 0 ова цифра обрадила мора се организовати петља са постусловом `do-while`. У неким случајевима, на пример, код бројања цифара, то утиче на исправност алгоритма, док у неким другим, на пример, код сабирања свих цифара, то није неопходно (наиме обрада цифре подразумева њено додавање на збир, а сасвим је свеједно да ли ће се збир увећати за цифру нула или ће остати непромењен - ефекат је исти). Тако се добија следећи алгоритам.

```
do {
    c = n % 10;
    ...
    n /= 10;
} while (n != 0);
```

Када знамо како се одређује серија цифара, онда је лако одредити њене статистике - број и збир. Број цифара се израчунава бројањем елемената серије, на стандардни начин (слично као, на пример, у задатку **Читање до нуле** или **Просек одличних**). Бројач се пре петље иницијализује на нулу и увећава се за један у телу петље, приликом одређивања сваке наредне цифре. Збир се израчунава сабирањем елемената серије, опет на веома стандардан начин (слично као, на пример, у задатку **Збир n бројева**). У почетку се променљива која треба да садржи збир иницијализује на нулу (неутрални елемент за сабирање), а затим се у сваком кораку петље увећава за вредност тренутно одређене цифре броја (она је одређена вредношћу $n \bmod 10$).

```
int n;
cin >> n;
int brojCifara = 0, zbirCifara=0;
do {
    brojCifara++;           // uvecava se broj cifara
    int cifra = n % 10;    // izdvaja cifru
    zbirCifara += cifra;   // uvecava zbir cifara
    n /= 10;               // uklanja cifru broja
} while (n > 0);
cout << brojCifara << " " << zbirCifara << endl;
```

Број и збир цифара могуће је одредити и рекурзивном функцијом.

```
#include <iostream>
using namespace std;

int brojCifara(int n) {
    if (n < 10)
        return 1;
    else
        return brojCifara(n / 10) + 1;
}

int zbirCifara(int n) {
    if (n < 10)
        return n;
    else
        return zbirCifara(n / 10) + n % 10;
}

int main() {
    int n;
    cin >> n;
    cout << brojCifara(n) << " "
         << zbirCifara(n) << endl;
    return 0;
}
```

Број и збир цифара могуће је одредити и репно-рекурзивним функцијама.

```
#include <iostream>
using namespace std;

void brojIZbirCifara_rek(int n, int& broj, int& zbir) {
    if (n > 0) {
        broj++;
        zbir += n % 10;
        brojIZbirCifara_rek(n / 10, broj, zbir);
    }
}

void brojIZbirCifara(int n, int& broj, int& zbir) {
    broj = 1; zbir = n % 10;
    brojIZbirCifara_rek(n / 10, broj, zbir);
}

int main() {
    int n;
    cin >> n;
    int broj, zbir;
    brojIZbirCifara(n, broj, zbir);
    cout << broj << " " << zbir << endl;
}
```

Задатак: Да ли запис броја садржи цифру

Написати програм којим се испитује да ли декадни запис природног броја садржи дату цифру?

Улаз: У првој линији стандардног улаза уноси се природан број од 0 до 1000000000, а у другој декадна цифра.

Излаз: На стандардни излаз се исписује текстуални одговор, облика који је приказан у наредним примерима.

Пример 1

```
Улаз      Излаз
29384     број 29384 sadrzi cifru 3
3
```

Пример 2

```
Улаз      Излаз
29384     број 29384 ne sadrzi cifru 7
7
```

Решење

Провера да ли број садржи цифру заснива се на алгоритму издвајања цифара броја слесна налево који смо видели у задатку **Број и збир цифара броја** и на алгоритму претраге којим се проверава да ли у серији постоји тражени елемент тј. елемент са траженим својством, који смо видели, на пример, у задатку **Негативан број**.

Као што смо видели, постоје разни начини да се алгоритам претраге имплементира. Један од најлакших је да се претрага имплементира у засебној функцији. Из задатог броја n (параметра функције), све док је већи од 0, у петљи се издваја цифра јединица ($n \% 10$), проверава се да ли је издвојена цифра једнака задатој цифри (параметру функције). Ако јесте, прекида се извршавање функције и враћа логичка вредност `true`, а ако није, из броја се уклања издвојена цифра ($n /= 10$). Петља се понавља док се не пронађе тражена цифра или док n не постане 0, када се прекида извршавање, јер су задатом броју уклоњене све цифре. Пошто цифра није пронађена (јер би се иначе функција раније прекинула и вратила `true`), након петље функција треба да врати логичку вредност `false`. Пошто функција исправно треба да јави да број 0 садржи цифру 0, пожељно је осигурати да ће се за сваки број бар једна цифра проверити, што је могуће ако се употреби петља са провером услова на крају.

Обратимо пажњу на то да је вредност `false` могуће вратити тек по завршетку петље, када су све цифре броја испитане. Честа грешка је да се она врати у грани `else` наредбе `if` у којој се упоређују текућа и тражена

цифра.

```
// provera da li se cifra cifra sadrzi u dekadnom zapisu broja n
bool sadrziCifru(int n, int cifra) {
    do {
        if (n % 10 == cifra) return true;
        n /= 10;
    } while (n > 0);
    return false;
}

int main() {
    int n, cifra;
    cin >> n >> cifra;
    cout << "broj " << n << (sadrziCifru(n, cifra) ? " " : " ne ")
        << "sadrzi cifru " << cifra << endl;
    return 0;
}
```

Претрага би се могла имплементирати и без засебне функције. У том случају би се користила логичка променљива која би на почетку имала вредност `false`, и која би се у тренутку проналаска тражене цифре постављала на `true` (ефикасности ради, у том тренутку би се петља могла прекинути, било наредбом `break` било ојачавањем услова петље у којем би се тражило да претрага траје само док је вредност те логичке променљиве `false`).

Рецимо и да уобичајени алгоритам издвајања цифара броја сдесна постепено мења његову вредност. Зато, ако је потребно након претраге имати оригиналну вредност броја на располагању (а у овом задатку јесте, да би се могла исписати одговарајућа порука), онда је потребно направити његову копију пре претраге. У решењу са функцијом, то није било неопходно радити јер се целобројни параметри у функцију преносе по вредности, што значи да се копирају током позива функције.

```
int n, cifra;
cin >> n >> cifra;
int m = n;
bool sadrzi = false;
do {
    if (m % 10 == cifra) {
        sadrzi = true;
        break;
    }
    m /= 10;
} while (m > 0);
cout << "broj " << n << (sadrzi ? " " : " ne ")
    << "sadrzi cifru " << cifra << endl;
```

Задатак: Различите цифре

Написати програм којим се испитује да ли су све цифре у декадном запису датог природног броја различите?

Улаз: Са стандардног улаза уноси се природан број од 0 до $2 \cdot 10^9$.

Излаз: На стандардном излазу исписује се текстуални одговор DA или NE.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
67569	NE	1234567890	DA

Решење

Постоје различити начини да се провери да ли су све цифре броја различите.

Претрага појављивања сваке цифре у префиксу

Један је да се за сваку цифру броја провери да ли се јавља у префиксу броја испред себе (делу броја испред те цифре). На пример, за број 132345 проверавамо да ли се цифра 5 јавља у броју 13234, пошто се не јавља, проверавамо да ли се цифра 4 јавља у броју 1323, пошто се не јавља, проверавамо да ли се цифра 3 јавља у броју 132 и пошто се јавља, можемо да закључимо да полазни број нема све различите цифре. Дакле, у петљи у којој се одређују цифре броја слесна налево (коју смо описали у задатку **Број и збир цифара броја**), за сваку цифру одређену као $n \% 10$ проверавамо да ли је садржана у запису броја $n / 10$ добијеног њеним уклањањем. У основни лежи алгоритам претраге (испитујемо да ли постоји цифра која задовољава услов да се појављује), и најлакше га је имплементирати у засебној функцији (као што је показано, на пример, у задатку) - први пут када се у петљи наиђе на цифру која је садржана у префиксу испред ње, функција која испитује да ли су све цифре различите враћа вредност `false`, а након петље враћа вредност `true`. Испитивање да ли запис броја садржи дату цифру можемо реализовати у засебној функцији, како је описано у задатку **Да ли запис броја садржи цифру**.

```
bool razluciteCifre(int n) {
    while (n >= 10) {
        if (sadrziCifru(n / 10, n % 10))
            return false;
        n /= 10;
    }
    return true;
}
```

Види групачија решења овој задатку.

Задатак: Армстронгов број

Написати програм који за дати природан број n проверава да ли је тај број Армстронгов. k -то цифрен број је Армстронгов ако је једнак суми k -тих степена својих цифара. На пример, 370 је Армстронгов јер је $370 = 3^3 + 7^3 + 0^3$, 1634 је Армстронгов јер је $1634 = 1^4 + 6^4 + 3^4 + 4^4$, док 12 није Армстронгов јер је $12 \neq 1^2 + 2^2$.

Улаз: Са стандардног улаза се учитава природан број n ($1 \leq n \leq 100000$).

Израз: На стандардном излазу исписују се порука `DA` ако учитан број јесте Армстронгов, тј. `NE` ако учитан број није Армстронгов.

Пример 1		Пример 2	
Улаз	Израз	Улаз	Израз
1002	NE	370	DA

Решење

Проверу да ли је број Армстронгов можемо разложити на неколико подзадатака.

Прво, на основу дефиниције Армстронговог броја изложилац на који ће се степеновати све цифре директно зависи од броја цифара броја (на пример, за троцифрене бројеве рачуна се збир кубова цифара, а за петоцифрене бројеве збир петих степена цифара). Зато је потребно да пребројимо цифре учитаног броја, што радимо на исти начин као у задатку **Број и збир цифара броја**.

Друго, потребно је да умемо да рачунамо степене цифара. Један начин да то урадимо је да искористимо библиотечку функцију за степеновање (`pow`), међутим, пошто она ради са реалним бројевима испрограмираћемо функцију која врши специјалан случај степеновања када су основа и изложилац природни бројеви, коришћењем алгоритма приказаног у задатку **Степен**. Ако рачунамо x^n , потребно је да израчунамо производ у којем се n пута јавља чинилац x . Зато је потребно да производ иницијализујемо на 1, а затим да га у петљи чије се тело извршава n пута помножимо са x у којем се такође израчунава производ).

Када умемо да рачунамо степен, потребно је да израчунамо збир степена цифара броја. Серију цифара броја одређујемо на уобичајени начин (као, на пример, у задатку **Број и збир цифара броја**), слесна на лево, коришћењем целобројног дељења са 10. За сваку издвојену цифру израчунавамо степен и добијени резултат додајемо на текући збир, који пре петље иницијализујемо на нулу. Приметимо да ово веома личи на израчунавање збира цифара које смо приказали у задатку **Број и збир цифара броја**, једино што се на збир не додају цифре, већ њихови степени. Дакле, у сваком кораку пресликавамо текућу цифру степеном функцијом и сабирамо серију добијених слика, тј. примењујемо алгоритам сабирања пресликане серије.

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

На крају веома једноставно можемо проверити да ли је број Армстронгов тако што га упоредимо са добијеним збиром степена.

Један начин имплементације је да сваку од наведених функционалности имплементирамо у засебној функцији. Тако бисмо дефинисали функцију која за дати број израчунава број цифара броја, функцију која израчунава степен, функцију која израчунава збир степена бројева и функцију која проверава да ли је број Армстронгов.

```
// funkcija izracunava broj cifara broja n
int brojCifara(int n) {
    int broj = 0;
    do {
        broj++;
        n /= 10;
    } while (n > 0);
    return broj;
}

// funkcija izracunava stepen x^n
int stepen(int x, int n) {
    int s = 1;
    for (int i = 0; i < n; i++)
        s *= x;
    return s;
}

// funkcija izracunava zbir k-tih stepena cifara broja n
int zbirKtihStepenaCifara(int n, int k) {
    int zbir = 0;
    do {
        zbir += stepen(n%10, k); // ili (int)pow(n%10, k)
        n /= 10;
    } while (n > 0);
    return zbir;
}

// funkcija proverava da li je dati broj Armstrongov tj. ako broj ima k
// cifara, da li mu je zbir k-tih stepena cifara jednak njemu samom
bool armstrongov(int n) {
    return zbirKtihStepenaCifara(n, brojCifara(n)) == n;
}

int main() {
    int n;
    cin >> n;
    if (armstrongov(n))
        cout << "DA" << endl;
    else
        cout << "NE" << endl;
    return 0;
}
```

Ако бисмо се одлучили да све имплементирамо без помоћних функција, тада је потребно нагласити да се током извршавања неколико наведених корака број мења (на пример, приликом бројања цифара број у сваком кораку делимо са 10, све док не постане једнак нули). Зато је потребно да уведемо посебну променљиву у којој се чува оригинална вредност броја, да бисмо након измене броја могли да повратимо његову праву вредност (алтернативно тумачење исте имплементације је да се пре спровођења алгорита оригинална вредност копира у помоћну променљиву која се током спровођења алгорита мења).

```
// broj za koji proveravamo da li je Armstrongov
```



```

int n;
cin >> n;

// pamtimo originalnu vrednost broja n, jer ce se n menjati tokom
// sprovođenja algoritama
int n0 = n;

// broj cifara broja n
int brCifara = 0;
while (n > 0) {
    brCifara++;
    n /= 10;
}

// vracamo n na pocetnu vrednost
n = n0;

// suma k-tih stepena cifara broja x, gde k broj cifara broja x
long zbirStepena = 0;
while (n != 0) {
    // izdvajamo i uklanjamo poslednju cifru broja
    int cifra = n % 10; n /= 10;
    // racunanje k-tog stepena cifara broja x gde k broj cifara broja x
    // stepen = pow(cifra, brCifara);
    int stepen = 1;
    for (int j = 1; j <= brCifara; j++)
        stepen *= cifra;
    // uvecavamo sumu za izracunati stepen
    zbirStepena += stepen;
}

if (zbirStepena == n0)
    cout << "DA" << endl;
else
    cout << "NE" << endl;

```

Задатак: Трансформација броја у производ цифара

Природан број се трансформише тако што се замени производом својих цифара. Написати програм којим се испишују све трансформације броја док се не добије једноцифрен број. Одредити и колико трансформација је при томе потребно извршити.

Улаз: Једна линија стандардног улаза садржи један природан број.

Излаз: На стандардном излазу приказати у свакој линији по једну трансформацију датог броја (природан број), до једноцифреног броја. У последњој линији стандардног излаза приказати број трансформација.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
3274	168	5	0
	48		
	32		
	6		
	4		

Решење

Израчунавање производа цифара датог броја реализоваћемо у посебној функцији. Да би се он израчунао, потребно је одредити серију цифара броја (као што смо то описали у задатку **Број и збир цифара броја**) и пронаћи производ елемената те серије (као што смо то описали у задатку **Факторијел**). У петљи, производ, чија је вредност на почетку 1, ћемо множити, редом, цифрама броја, почев од цифре јединица. Цифру јединица

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

броја одређујемо као остатак при дељењу броја са 10. Дељењем броја са 10 уклањамо цифру јединица па у следећем извршавању тела петље, приступамо следећој цифри. Понављамо поступак док полазни број не постане 0.

У главној функцији, у петљи, броју додељујемо производ његових цифара док не постане једноцифрен (мањи од 10). Број извршавања тела петље је тражени број трансформација (њена ћемо одредити коришћењем посебног бројача који ћемо иницијализовати на нулу и увећавати у сваком кораку петље).

```
// računa proizvod cifara broja n
int proizvodCifara(int n) {
    int p = 1;          // proizvod cifara
    do {
        p *= n % 10;    // mnozimo proizvod poslednjom cifrom broja n
        n /= 10;        // uklanjamo poslednju cifru broja n
    } while (n != 0);  // dok n ne postane 0
    return p;
}

int main() {
    int n;              // broj koji analiziramo
    cin >> n;
    int brojac = 0;     // broj transformacija dok ne postane jednocifren
    while (n >= 10) {   // dok je broj visecifren
        n = proizvodCifara(n); // menjamo ga proizvodom njegovih cifara
        cout << n << endl;    // ispisujemo dobijeni rezultat
        brojac++;           // uvecavamo brojac transformacija
    }
    cout << brojac << endl; // ispisujemo konacan broj transformacija
    return 0;
}
```

Задатак: Најмањи број са највећим збиром парних цифара

Уносе се природни бројеви (укључујући и 0) док се не унесе -1. Међу унетим бројевима који садрже бар једну парну цифру, наћи најмањи број са највећим збиром парних цифара.

Улаз: Линије стандардног улаза, њих највише милион, садрже по један природан број. Последња линија стандардног улаза садржи број -1.

Излаз: Прва и једина линија стандардног излаза садржи најмањи број од унетих бројева са највећим збиром парних цифара. Ако међу унетим бројевима нема оних који садрже парне цифре, на излазу приказати -1.

Пример 1

Улаз	Излаз
137	4
20	
143	
221	
0	
22	
4	
-1	

Пример 2

Улаз	Излаз
137	-1
39	
155	
791	
731	
31	
-1	

Решење

У овом задатку се комбинују разне технике приказане у ранијим задацима. Прво, потребно је уносити серију бројева, све док се не прочита број -1, што је могуће урадити слично као у задатку **Читање до нуле**. Међу учитаним бројевима анализирамо само оне који садрже бар једну парну цифру, тј. вршимо филтрирање серије на основу услова да садржи бар једну парну цифру. Испитивање да ли број садржи бар једну парну цифру засновано је на комбинацији алгорита одређивања цифара броја (који смо видели у задатку **Број и збир цифара броја**) и алгоритму претраге, који смо користили, на пример, у задатку **Негативан број**. Међу бројевима који садрже парне цифре одређујемо максимум (врши се одређивање максимума тј. минимума

филтриране серије). Максимум се одређује у односу на релацију лексикографског поретка, слично као што је показано, на пример, у задатку **Најближи датом целом броју**. Прво се пореди збир парних цифара. Цифре броја издвајамо како је приказано у задатку **Број и збир цифара броја**, а затим сабирамо оне које су парне, што је алгоритам одређивања статистике филтриране серије, какав смо видели у задатку **Просек одличних**.

Основна варијанта алгоритма

Имајући речено у виду, можемо описати основну варијанту алгоритма.

Имплементираћемо функцију која одређује да ли декадни запис датог броја садржи неку парну цифру. У петљи одређујемо и уклањамо цифру по цифру слесна коришћењем целобројног дељења са 10. Ако је цифра парна (тј. ако јој је остатак при дељењу са 2 једнак нули), тада функција може да врати вредност `true`. Након завршетка петље, на крају функције вратићемо вредност `false`, јер током извршавања петље није пронађена парна цифра. Нагласимо да посебну пажњу треба обратити на случај броја нула, који садржи парну цифру - један начин да се тај случај исправно обради је да се употреби петља са провером услова на крају или у телу петље.

Имплементираћемо и функцију која одређује збир цифара декадног записа датог броја (збир који ћемо иницијализовати на нулу ћемо увећавати за једну по једну цифру броја), исто као у задатку **Број и збир цифара броја**.

У главном програму можемо у бесконачној петљи учитавати број по број и након сваког учитаног броја проверавати да ли је учитана специјална вредност -1. Ако јесте, прекидаћемо петљу. За сваки учитани број провераваћемо да ли садржи неку парну цифру (помоћу функције коју смо имплементирали) и број ћемо даље обрађивати само ако садржи парну цифру. Одржаваћемо променљиву `max` која чува вредност до сада најбољег учитаног елемента (најмањег до сада учитаног броја који садржи бар једну парну цифру и има збир парних цифара већи или једнак збировама парних цифара свих до сада учитаних бројева). Одржаваћемо и логичку променљиву која ће указивати на то да ли је до сада учитан неки број који садржи парну цифру, тј. да ли је променљива `max` постављена (друга могућност је да максимум иницијализујемо на неку специјалну, негативну вредност). За сваки учитани број за који је утврђено да садржи парну цифру, провераваћемо да ли је бољи од свих раније учитаних тј. да ли вредност променљиве `max` треба ажурирати и поставити на тај број. Постоји неколико случајева када је то потребно урадити.

- Први случај је када је ово први од свих до сада учитаних бројева који садржи парну цифру (што лако испитујемо на основу вредности одговарајуће логичке променљиве), и у том случају нема потребе за поређењима са вредношћу променљиве `max` (она у том тренутку ни нема постављену праву вредност),
- други случај је када је збир парних цифара тренутног броја (који можемо израчунати коришћењем функције коју смо имплементирали) мањи од вредности збира парних цифара броја `max`,
- трећи случај је када су та два збира једнака, али је тренутни број мањи од броја `max`.

У првом случају треба ажурирати и вредност логичке променљиве и поставити је на вредност `true` (а не смета да се то уради и у осталим случајевима, увек када се ажурира вредност `max`).

На крају, по завршетку петље у којој се учитавају сви бројеви, на основу логичке променљиве знамо да ли је био учитан бар један број чији декадни запис садржи парну цифру и ако јесте, исписујемо вредност променљиве `max`, а ако није, исписујемо специјалну вредност -1.

```
// vraca true ako i samo ako n sadrzi bar jednu parnu cifru u svom
// dekadnom zapisu
bool imaParnihCifara(int n) {
    do {
        if (n % 2 == 0)
            return true;
        n /= 10;
    } while (n != 0);
    return false;
}

// izracunava zbir parnih cifara broja n
int zbirParnihCifara(int n) {
    int zbir = 0;
    do {
```

```
    if (n % 2 == 0)
        zbir += n % 10;
    n /= 10;
} while (n != 0);
return zbir;
}

int main() {
    // najbolji do sada procitani broj (najmanji broj sa najvećim zbirom
    // parnih cifara)
    int max;
    // logicka promenljiva koja govori da li je do sada pronadjen broj
    // koji sadrzi parnu cifru tj. da li promenljiva max ima validnu
    // vrednost
    bool postavljenMax = false;

    // ponavljamo sledeci postupak
    while (true) {
        // ucitavamo naredni broj
        int n;
        cin >> n;
        // ako smo ucitali -1, prekidamo izvršavanje
        if (n == -1)
            break;
        // ako broj n sadrzi bar jednu parnu cifru
        if (imaParnihCifara(n)) {
            if (!postavljenMax ||
                // ako je n prvi takav broj
                zbirParnihCifara(n) > zbirParnihCifara(max) ||
                // ili mu je zbir parnih cifara veci od prethodno najboljeg
                zbirParnihCifara(n) == zbirParnihCifara(max) && n < max
                // ili je jednak prethodno najboljem, ali je sam broj manji
            ) {
                // azuriramo najbolji broj i postavljamo ga na trenutni
                max = n;
                // registrujemo da je pronadjen bar jedan broj koji sadrzi
                // parnu cifru tj. da promenljiva max ima pravu vrednost
                postavljenMax = true;
            }
        }
    }
    // ispisujemo rezultat (-1, ako ne postoji broj sa parnim ciframa)
    cout << (postavljenMax ? max : -1) << endl;
    return 0;
}
```

Могуће оптимизације

Овако дефинисан алгоритам исправно ради, али га је могуће оптимизовати у неколико аспеката. Један од проблема је то што се приликом сваког упоређивања изнова рачуна и збир цифара броја `max` и збир цифара тренутно учитаног броја. Зато има смисла уз променљиву `max` чувати и променљиву која садржи збир њених парних цифара. Приликом обраде сваког новог броја који садржи бар једну парну цифру, треба израчунати њен збир парних цифара, проверити да ли је испуњен неки од наведених услова за ажурирање вредности `max` и ако јесте, уз ажурирање вредности `max` на вредност текућег броја, променљивој која чува збир парних цифара броја `max` треба доделити раније израчунату вредност збира парних цифара текућег броја.

Још једно место могуће оптимизације је замена две независне функције (једне која проверава да ли број садржи парну цифру и друге која рачуна збир парних цифара) једном функцијом која ради оба задатка. Та функција враћа и збир парних цифара и податак о томе да ли број садржи парну цифру. Функција може обе вредности да врати преко излазних параметара, може једну да врати преко редовне повратне вредности,

а другу преко повратног параметара. Још једна могућност је да функција врати неку специјалну негативна вредност ако број не садржи ниједну парну цифру (јер се зна да су регуларне вредности збира парних цифара увек ненегативне).

Ипак не може се очекивати да ће ово довести до значајних убрзања, јер се захваљујући раном прекиду приликом претраге, детекција постојања парних цифара изврши доста брже него обилазак и сабирање свих парних цифара броја.

```
// vraca zbir parnih cifara broja ako broj sadrzi bar jednu parnu
// cifru i vrednost -1 ako broj ne sadrzi parnu cifru
int zbirParnihCifara(int n) {
    // da li broj sadrzi parnu cifru?
    bool imaParnihCifara = false;
    // zbir parnih cifara
    int zbir = 0;
    do {
        if (n % 2 == 0) { // skraceno od (n % 10) % 2 == 0
            // pronadjena je parna cifra
            imaParnihCifara = true;
            // uvecavamo zbir
            zbir += n % 10;
        }
        n /= 10;
    } while (n != 0);
    // vracamo ili izracunati zbir ili -1, ako parna cifra nije nadjena
    return imaParnihCifara ? zbir : -1;
}

int main() {
    // najbolji do sada procitani broj (najmanji broj sa najvećim zbirom
    // parnih cifara) - vrednost -1 ukazuje na to da do sada nije
    // pronadjen nijedan broj koji sadrzi parnu cifru
    int max = -1;
    // zbir parnih cifara broja max
    int zbirParnihCifaraMax;

    // ponavljamo sledeci postupak
    while (true) {
        // ucitavamo naredni broj
        int n;
        cin >> n;
        // ako smo ucitali -1, prekidamo izvršavanje
        if (n == -1)
            break;
        // izracunavamo zbir parnih cifara broja n
        int zbirParnihCifaraN = zbirParnihCifara(n);
        // ako broj n sadrzi bar jednu parnu cifru
        if (zbirParnihCifaraN != -1) {
            if (max == -1 ||
                // ako je on prvi takav broj
                zbirParnihCifaraN > zbirParnihCifaraMax ||
                // ili mu je zbir parnih cifara veci od prethodno najboljeg
                zbirParnihCifaraN == zbirParnihCifaraMax && n < max
                // ili je jednak prethodno najboljem, a sam broj je manji
            ) {
                // azuriramo najbolji broj i postavljamo ga na trenutni
                max = n;
                // azuriramo i njegov zbir parnih cifara
                zbirParnihCifaraMax = zbirParnihCifaraN;
            }
        }
    }
}
```

```

    }
  }
}
// ispisujemo rezultat (-1, ako ne postoji broj sa parnim ciframa)
cout << max << endl;
return 0;
}

```

Задатак: Број формиран од датих цифра с лева на десно

Написати програм којим се формира природан број од учитаних цифара, ако се цифре броја читавају слева на десно (редом од цифре највеће тежине до цифре јединица).

Улаз: Свака линија стандарног улаза, њих највише 9, садржи по једну цифру.

Излаз: На стандарном излазу приказати формиран број.

Пример

Улаз	Излаз
4	4109
1	
0	
9	

Решење

Читамо цифру по цифру са стандардног улаза, све док не дођемо до краја и формирамо број додавајући му прочитану цифру на десну страну (као цифру најмање тежине). Формирање броја на основу цифара с лева надесно разматрали смо и раније (на пример, у задатку [Јарди, стопе и инчи](#)). Алгоритам се назива *Хорнерова шема* и на основу тог алгоритма број се гради тако што се у сваком кораку претходна вредност броја помножи са 10 и добијени производ се увећа за вредност наредне цифре (тако се цифра допише на десни крај претходног броја). Анализирајмо пример у којем читавамо редом цифре 3, 2, 7 и 5 и треба да добијемо број 3275. На основу дефиниције позиционог записа тај број је једнак вредности израза $3 \cdot 10^3 + 2 \cdot 10^2 + 7 \cdot 10 + 5$. Међутим, претходни израз можемо израчунати на следећи начин $((3 \cdot 10 + 2) \cdot 10 + 7) \cdot 10 + 5$, што доводи до Хорнеровог поступка. Ако се он имплементира итеративно, променљива n којом се представља вредност броја узима редом вредности 0, 3, 32, 327 и 3275.

Пошто број цифара није унапред познат, цифре ћемо читавати у петљи којом читавамо бројеве до краја стандардног улаза. Та техника је приказана у задатку [Читање до краја улаза](#).

Број који формирамо памтићемо у променљивој n коју иницијализујемо на нулу. Када прочитамо цифру, додајемо је као цифру јединица на до сада формиран број. Додавање цифре јединица на број n , постижемо тако што помножимо броја n са 10 и додамо му читану цифру. Ако унесемо k цифара, на описан начин цифру коју смо прву прочитали множићемо са 10 тачно $k - 1$ пут, другу прочитану цифру множимо са 10 тачно $k - 2$ пута, и тако редом, последњу прочитану цифру не множимо са 10 (то је цифра јединица).

Докажимо и формално коректност Хорнерове шеме. Претпоставимо да ће се редом читавати цифре $a_{k-1}, a_{k-2}, \dots, a_1, a_0$. Инваријанта петље је то да је n вредност броја који се добија записом до сада прочитаних и обрађених цифара. Након i извршавања тела петље то су цифре од a_{k-1} до a_{k-i} и тврдимо да важи

$$n = (a_{k-1} \dots a_{k-i})_{10} = a_{k-1}10^{i-1} + a_{k-2}10^{i-2} + \dots + a_{k-i}10^0.$$

- *База.* Након 0 извршавања тела петље важи да је $n = 0$ и $i = 0$. Број $(a_{k-1} \dots a_{k-i})_{10} = (a_{k-1} \dots a_k)_{10}$ нема ниједну цифру и вредност му је нула.
- *Корак.* Претпоставимо да инваријанта важи на уласку у тело петље. Нека је $i' = i + 1$. Из тела петља види се да је $n' = 10 \cdot n + a_{k-i-1}$. Пошто на основу индуктивне хипотезе важи $n = a_{k-1}10^{i-1} + a_{k-2}10^{i-2} + \dots + a_{k-i}10^0$, важи и да је

$$n' = 10(a_{k-1}10^{i-1} + a_{k-2}10^{i-2} + \dots + a_{k-i}10^0) + a_{k-i-1} = a_{k-1}10^i + a_{k-2}10^{i-1} + \dots + a_{k-i}10 + a_{k-i-1}.$$

Пошто је $i' = i + 1$, важи да је

$$n' = a_{k-i}10^{i'-1} + a_{k-2}10^{i'-2} + \dots + a_{k-i'+1}10 + a_{k-i'},$$

па је инваријанта очувана.

Када се петља заврши, биће учитано k цифара, па ће важити да је $i = k$, тј.

$$n = (a_{k-1} \dots a_{k-i})_{10} = a_{k-1}10^{i-1} + a_{k-2}10^{i-2} + \dots + a_1 \cdot 10^1 + a_0 \cdot 10^0,$$

што значи да n садржи декадну вредност броја формираног од свих k цифара.

```
int cifra;
int n = 0;
while (cin >> cifra)
    n = n * 10 + cifra;
cout << n << endl;
```

Задатак: Број формиран од датих цифара здесна на лево

Написати програм којим се формира природан број од учитаних цифара, ако се цифре броја читавају здесна на лево (редом од цифре јединица до цифре највеће тежине).

Улаз: Свака линија стандардног улаза (њих највише 9) садржи по једну цифру.

Излаз: На стандардном излазу приказати формиран број.

Пример

Улаз	Излаз
0	92300
0	
3	
2	
9	

Решење

Читамо цифру по цифру са стандардног улаза, све док не дођемо до краја и формирамо број додавајући му прочитану цифру на леву страну (као цифру највеће тежине). Формирање броја на основу цифара здесна налево разматрали смо и раније, у задатку [Цифре слесна](#). Цифре се читавају здесна на лево па су њихове позиције редом цифра јединица, цифра десетица, стотина и тако даље. На пример ако читамо цифре редом 3, 2, 7 и 5, треба да добијемо број 5723. Тај број можемо формирати као $3 + 2 \cdot 10 + 7 \cdot 10^2 + 5 \cdot 10^3$. Тежину текуће цифре можемо одредити уз помоћ променљиве у којој памтимо одговарајући степен броја 10.

Пошто број цифара није унапред познат, цифре ћемо читавати у петљи којом читавамо бројеве до краја стандардног улаза (веома слично као што је приказано у задатку [Читање до краја улаза](#)). Број који формирамо памтићемо у променљивој n коју иницијализујемо на нулу, а тежину цифре, тј. степен броја 10 у променљивој s коју иницијализујемо на 1. Цифру додајемо броју n са лева тако што број n увећамо за производ те цифре и одговарајућег степена броја 10 (променљиве s). У сваком кораку вредност променљиве s коригујемо множећи је са 10 (тако ће, с обзиром да је иницијализована на вредност 1, она редом узимати вредности 1, 10, 100, 1000 ...).

```
int cifra;
int n = 0;
int st10 = 1;
while (cin>>cifra) {
    n = n + cifra*st10;
    st10 *= 10;
}
cout << n << endl;
```

Задатак: Замени цифре 0 са 5

Написати програм који у датом природном броју свако појављивање цифре 0 замењује цифром 5.

Улаз: Са стандардног улаза се читава природан број у границама од 0 до 1000000000.

Излаз: На стандардном излазу се исписује добијени број.

Пример

Улаз Излаз
20240607 25245657

Решење

У решењу овог задатка комбинујемо алгоритам одређивања серије цифара броја здесна налево, заснован на целобројном дељењу са 10, који смо објаснили у задатку **Број и збир цифара броја** и алгоритам формирања броја од серије његових цифара здесна налево који смо објаснили у задатку **Број формиран од датих цифара здесна на лево**. Издвајаћемо цифру по цифру полазног броја и додаваћемо их на почетак новог броја, при чему ћемо проверавати да ли је тренутна цифра једнака цифри коју треба заменити. Ако јесте, на почетак броја додаће се цифра којом се мења цифра коју треба заменити, а ако није, додаће се тренутна цифра.

Имплементираћемо функцију која је мало општија од постављеног задатка и која мења дату цифру a цифром b (у главном програму ћемо је позвати за вредности $a=0$ и $b=5$). Променљиву која ће садржати модификовани број (у коме је a замењена са b) ћемо пре петље иницијализовати на нулу. Увешћемо и променљиву која одређује тежину наредне цифре и иницијализоваћемо је на 1 (тежина цифре јединица). У петљи ћемо издвајати последњу цифру c текуће вредности броја n (одређивањем остатка при дељењу са 10), уклањати је из броја (одређивањем целобројног количника броја n са 10) и проверавати да ли је цифра јединица једнака a . Ако јесте, нови број ћемо увећавати за производ цифре b и текуће тежине, а у супротном за производ цифре c и текуће тежине. Након тога ћемо тежину увећати 10 пута тј. постављаћемо је на тежину наредне цифре. Тело петље се понавља док се из датог броја n не издвоје све цифре, тј. док вредност n не постане 0.

```
// u broju n zamenjuje svako pojavljivanje cifre a sa cifrom b
int zameni(int n, int a, int b) {
    int t = 1; // teжина cifre, uzima redom vrednosti 1, 10, 100, ...
    int noviBroj = 0; // broj koji se dobija nakon zamene cifara
    // prolazimo kroz sve cifre broja n, sdesna nalevo
    do {
        int c = n % 10; // izdvajamo cifru jedinica
        // u novi broj, na pocetak upisujemo
        // - cifru b, ako je c == a,
        // - cifru c, inace
        noviBroj += (c == a ? b : c) * t;
        n /= 10; // uklanjamo cifru jedinica iz broja
        t *= 10; // izracunavamo tezinu naredne cifre
    } while (n > 0);
    return noviBroj;
}

int main() {
    int n;
    cin >> n;
    cout << zameni(n, 0, 5) << endl;
    return 0;
}
```

Задатак: Децимале броја $1/n$

Одреди првих k децималних цифара вредности $\frac{1}{n}$ за дати позитиван природан број n . Претпоставити да се након k децимала врши одсецање тј. да се број заокружује наниже.

Улаз: Прва линија стандардног улаза садржи природан број k ($1 \leq k \leq 1000$) а друга природан број n ($2 \leq n \leq 1000$).

Излаз: На стандардном излазу, приказујемо количник са траженим бројем децимала (потребно је употребити децималну запету, а не децималну тачку).

Пример

Улаз Излаз
4 0,0303
33

Решење

Због јако великог броја децимала (k може бити и 1000) не можемо користити рад са реалним бројевима (тип `double` подржава највише 15-16 децимала).

Децимале одређујемо стандардним поступком дељења бројева. Размотримо, на пример, како се одређују прве три децимале броја $\frac{1}{7}$.

```

1 : 7 = 0.142
10
 7
--
 30
 28
--
  20
  14
  --
   6
    
```

Кренули смо од броја 1, помножили смо га са 10 (спустили наредну нулу), одредили смо количник са 7 и тако смо добили прву децималу 1. Након тога смо од броја 10 одузели производ броја 1 и броја 7 и тако добили број 3. Приметимо да је број 3 заправо остатак при дељењу броја 10 бројем 7. Након тога, помножили смо број 3 бројем 10, одредили смо количник са 7 и тако добили наредну децималу 4. Након тога, смо од броја 30 одузели количник броја 4 и броја 7 и тако добили број 2, што је заправо остатак при дељењу броја 30 бројем 7. Помножили смо 2 бројем 10, одредили количник са 7 и добили наредну децималу 2. Уједно смо припремили и наредни корак тако што смо од броја 20 одузели производ бројева 2 и 7 и тако добили број 6, што је заправо, остатак при дељењу бројева 20 и 7.

Приметимо да се у сваком кораку понавља исти поступак. Претходни број множимо са 10, а наредну децималу и наредни број одређујемо као целобројни количник и остатак добијеног производа бројем n . Овај корак треба поновити тачно k пута.

```

број = 1;
ponovi k puta:
  decimala = (10 * број) div n;
  број = (10 * број) mod n;
    
```

Покажимо и формално да се на овај начин исправно добијају децимале броја. Претпоставимо да је $0 \leq x < n$, и да је

$$\frac{x}{n} = 0, a_1 a_2 \dots a_k \dots,$$

Тада је

$$10 \cdot \frac{x}{n} = a_1, a_2 \dots a_k \dots,$$

Пошто је $0, a_2 \dots a_k \dots < 1$, а $0 \leq a_1 < 10$, важи да је

$$a_1 = \left\lfloor \frac{10 \cdot x}{n} \right\rfloor = (10 \cdot x) \text{ div } n.$$

На основу дефиниције целобројног количника и остатка (види задатак [Разломак у мешовит број](#)) важи да је $10 \cdot x = ((10 \cdot x) \text{ div } n) \cdot n + (10 \cdot x) \text{ mod } n$. Зато је

$$\frac{(10 \cdot x) \text{ mod } n}{n} = \frac{10 \cdot x}{n} - ((10 \cdot x) \text{ div } n) = (a_1, a_2 \dots a_k \dots) - a_1 = 0, a_2 \dots a_k \dots$$

Дакле, прва децимала броја $\frac{x}{n}$ се може одредити као $(10 \cdot x) \text{ div } n$ док се остале децимале могу одредити тако што се исти поступак примени на број $\frac{x'}{n}$, где је $x' = (10 \cdot x) \text{ mod } n$.

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

Приметимо да решење овог задатка има великих сличности са решењем задатка **К децимала у бинарном запису**.

```
int k, n;
cin >> k >> n;

cout << "0,";
int broj = 1;
for (int i = 0; i < k; i++) {
    cout << (10 * broj) / n;
    broj = (10 * broj) % n;
}
cout << endl;
```

Задатак: К децимала у бинарном запису

Написати програм којим се исписује k децимала бинарног записа реалног броја x ($0 \leq x < 1$).

Улаз: Прва линија стандардног улаза садржи реалан број из интервала $[0,1)$. Друга линија стандардног улаза садржи природни број k ($1 \leq k \leq 100$).

Издаз: У првој и јединој линији стандардног излаза приказати бинарни запис броја x , заокружен на k децимала.

Пример

Улаз	Издаз
0.8	0.1100
4	

Решење

Пошто тражени број децимала може бити велики, не можемо користити системску репрезентацију реалних бројева (која јесте бинарна), тј. резултат не можемо добити ишчитавањем битова мантисе. Наиме, мантиса у запису података типа `double` има око 50 битова, а у задатку се тражи и до 1000 битова.

Да бисмо објаснили бинарни запис реалних бројева, подсетимо се прво дефиниције позиционог записа броја. Као што је то раније показано запис $(x_n x_{n-1} \dots x_1)_b$ има вредност $x_n \cdot b^n + x_{n-1} \cdot b^{n-1} + \dots + x_1 \cdot b + x_0$. На пример, декадни запис 123 означава $1 \cdot 10^2 + 2 \cdot 10 + 3$, док бинарни запис $(1001)_2$ означава $1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2 + 1 = 9$.

Слично, децимални запис $0,123$ представља број $1 \cdot \frac{1}{10} + 2 \cdot \frac{1}{100} + 3 \cdot \frac{1}{1000}$. Заиста, децимални запис $(0, x_1 x_2 \dots x_k \dots)_b$ има вредност

$$x_1 \cdot b^{-1} + x_2 \cdot b^{-2} + \dots + x_k b^{-k} + \dots = x_1 \cdot \frac{1}{b} + x_2 \cdot \frac{1}{b^2} + \dots + x_k \cdot \frac{1}{b^k} + \dots$$

Зато запис броја x ($0 \leq x < 1$) у бинарном систему $0, x_1 x_2 x_3 \dots x_k \dots$ значи да је $x = \frac{x_1}{2} + \frac{x_2}{2^2} + \frac{x_3}{2^3} + \dots + \frac{x_k}{2^k} + \dots$

На пример, запис $0,101$ означава $1 \cdot \frac{1}{2} + 0 \cdot \frac{1}{4} + 1 \cdot \frac{1}{8} = 0,5 + 0,125 = 0,625$. Множењем са 2 добијамо $x \cdot 2 = x_1 + \frac{x_2}{2} + \frac{x_3}{2^2} + \dots + \frac{x_k}{2^{k-1}} + \dots$. Пошто је $\frac{x_2}{2} + \frac{x_3}{2^2} + \dots + \frac{x_k}{2^{k-1}} + \dots < 1$, закључујемо да је x_1 цео део производа $x \cdot 2$, тј. $x_1 = \lfloor x \cdot 2 \rfloor$, и да разломљени (децимални) део броја $x \cdot 2$ има вредност $\frac{x_2}{2} + \frac{x_3}{2^2} + \dots + \frac{x_k}{2^{k-1}}$, тј. да је његов бинарни запис $0, x_2 \dots x_k$. Неколико начина да се децимални број раздвоји на свој цео и разломљени део приказано је у задатку **Динари и паре** (на пример, цео део можемо израчунати функцијом `floor` или конверзијом у цео број, а можемо употребити и функцију `modf`).

Настављајући исти поступак за разломљени део броја $x \cdot 2$ добијамо и остале цифре бинарног записа.

Дакле, решење формулишемо тако што у петљи која се извршава k пута понављамо следеће:

- израчунавамо производ броја x и броја 2,
- приказујемо цео део добијеног производа (то је наредна цифра у бинарном запису),
- постављамо број x на вредност разломљеног дела нађеног производа, припремајући се тако за наредну итерацију.

```

double x;
int k;
cin >> x >> k;
cout << "0.";
for (int i = 0; i < k; i++) {
    x = x*2;
    cout << (int)x;
    x = x - (int)x;
}
cout << endl;

```

Задатак: Бројеви који не садрже цифру 5

Напиши програм који одређује колико бројева на стандардном улазу не садржи цифру 5.

Улаз: Свака линија стандардног улаза садржи један природан број из интервала $[0, 1000000000]$.

Излаз: На стандардни излаз исписати колико од учитаних бројева не садржи цифру 5.

Пример

<i>Улаз</i>	<i>Излаз</i>
12	2
15	
155	
14	

Задатак: Комбинација два броја минимумом и максимумом одговарајућих цифара

Два дата природна броја се комбинују тако што се потпишу један испод другог, као када се сабирају, али се свака цифра резултата одређује као мања или као већа од две цифре (ако су бројеви различите дужине тада се испред краћег броја допише онолико нула колико је потребно да се бројеви изједначе по дужини). Нпр. за бројеве 174 и 38, добија се 34 ако се узима мања од две цифре тј. 178 ако се узима већа.

Улаз: Свака од две линије стандарног улаза садржи по један природан број.

Излаз: На стандардном излазу приказати два природна броја - један који се добија комбиновањем два броја узимајући мању, а други узимајући већу цифру на свакој позицији.

Пример

<i>Улаз</i>	<i>Излаз</i>
174	34
38	178

3.2.10 Однос суседних елемената серије**Задатак: Максимална разлика суседних**

Написати програм којим се за n учитаних целих бројева одређује по апсолутној вредности највећа разлика два суседна елемента.

Улаз: У првој улазној линији учитава се природан број n ($2 \leq n \leq 100$), а у следећим n целих бројева у интервалу $[-100, 100]$.

Излаз: Исписује се тражени број који представља највећу апсолутну разлику два суседна броја.

Пример

<i>Улаз</i>	<i>Излаз</i>
5	85
-20	
30	
5	
90	
70	

Решење

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

Задатак решавамо применом алгоритма одређивања максимума серије бројева (приказаног, на пример, у задатку **Најнижа температура**) на серију апсолутних разлика суседних елемената низа.

Да бисмо одредили све апсолутне разлике суседних елемената, довољно је да у сваком тренутку помоћу две променљиве памтимо два суседна елемента оригиналне серије. Поступамо на један од следећа два начина, које ћемо употребљавати и у наредним задацима, када год је потребно да поредимо свака два суседна елемента у серијама.

Ако поступамо на први начин, на почетку први елемент учитавамо као текућу вредност, а затим, у петљи која ће се извршити $n - 1$ пута, пре читавања нове текуће вредности тренутну текућу вредност памтимо као претходну, учитавамо текућу вредност и након тога обрађујемо претходну и текућу вредност.

```
ucitaj tekuci
ponovi n-1 puta:
  prethodni = tekuci
  ucitaj tekuci
  obradi(prethodni, tekuci)
```

У овом конкретном задатку комбинујемо овај алгоритам са алгоритмом одређивања максимума. Пошто апсолутна разлика суседних елемената не може бити мања од 0, ову вредност, пре петље, постављамо као почетни максимум (који ће бити и крајњи резултат ако су сви елементи који се читавају једнаки). У сваком кораку петље рачунамо апсолутну вредност разлике између претходног и текућег елемента и проверавамо да ли је она већа од текуће вредности максимума. Ако јесте, максимум добија вредност њихове апсолутне разлике, у противном не мења вредност.

```
// broj elemenata koji se ucitava
int n;
cin >> n;
// u svakom trenutku pamtimo dva elementa serije:
// tekuci i njemu prethodni
int prethodni, tekuci;
// ucitavamo prvi element
cin >> tekuci;
// inicijalizujemo maksimum apsolutnih razlika na nulu
int max = 0;
for (int i = 2; i <= n; i++) {
  // tekuci element postaje prethodni,
  // pre nego sto ucitamo novi tekuci
  prethodni = tekuci;
  cin >> tekuci;
  // ako je tekuca razlika veca od do tada maksimalne,
  // azuriramo maksimum
  if (abs(tekuci - prethodni) > max)
    max = abs(tekuci - prethodni);
}
cout << max << endl;
```

Ако поступамо на други начин, на почетку први елемент учитавамо као претходну вредност, а затим, у петљи која ће се извршити $n - 1$ пута, учитавамо текућу вредност, обрађујемо претходну и текући вредности и након тога текућу вредност памтимо као претходну, припремајући се тако за евентуалну наредну итерацију у којој ће текући елемент заиста бити претходни елементу који ће тада бити читан.

```
ucitaj prethodni
ponovi n-1 puta:
  ucitaj tekuci
  obradi(prethodni, tekuci)
  prethodni = tekuci
```

У том облику решење може бити испрограмирано на следећи начин.

```
// broj elemenata koji se ucitava
int n;
cin >> n;
```

```

// u svakom trenutku pamtimo dva elementa serije:
// tekuci i njemu prethodni
int prethodni, tekuci;
// prvi element ce biti prethodni drugom
cin >> prethodni;
// inicijalizujemo maksimum apsolutnih razlika na nulu
int max = 0;
for (int i = 2; i <= n; i++) {
    cin >> tekuci;
    // ako je tekuca razlika veca od tada maksimalne,
    // azuriramo maksimum
    if (abs(tekuci - prethodni) > max)
        max = abs(tekuci - prethodni);
    // tekuci element postaje prethodni (njemu narednom)
    prethodni = tekuci;
}
cout << max << endl;

```

Види груичија решења овој задатку.

Задатак: Провера монотоности

Током неколико дана маратонац се спрема за предстојећу трку. За сваки дан познато је растојање које је тог дана претрчао. Одредити да ли је тркач правио напредак тј. да ли су учитани бројеви уређени у строго растућем редоследу.

Улаз: Прва линија стандардног улаза садржи природан број N ($1 \leq N \leq 50$) који представља број дана, а у свакој од наредних N линија налази се по један природан број који представља број метара који је тркач претрчао одговарајућег дана.

Излаз: На стандардном излазу исписати одговор Да ако су бројеви у строго растућем поретку и одговор Не у супротном.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
4	Да	4	Не
19		20	
20		20	
22		23	
23		23	

Решење

Желимо да утврдимо да су елементи серије претрчаних растојања тркача поређани у строго растућем поретку, тј. да проверимо да ли су сви елементи серије од другог до последњег строго већи од свог претходника. Решење се, дакле, заснива на примени алгоритма претраге, који смо увели у задатку **Негативан број**.

Довољно је упоредити свака два суседна елемента серије (претходни и текући) и проверити да ли је текући већи од претходног. За чување претходног и текућег елемента употребљавамо две променљиве. Можемо користити било који од два приступа описана у задатку **Максимална разлика суседних**. На пример, пре почетка петље, променљиву која чува претходни елемент низа иницијализоваћемо на први елемент серије (он постоји јер је серија по претпоставци задатка непразна). Затим у петљи чије се тело извршава $n - 1$ пута читавамо један по један елемент серије у променљиву која чува текући елемент серије. На крају тела петље, елемент који је био текући проглашавамо претходним (да бисмо наредни елемент читали у променљиву у којој се памти текући елемент и упоредили га са тренутно текућим елементом који постаје претходни). У телу петље поредимо претходни и текући. Ако је текући већи, актуелна растућа серија се наставља и не мењамо логички индикатор тј. променљиву `rastuci`, коју иницијализујемо на вредност `true`. Ако није, актуелна растућа серија је завршена, те мењамо логички индикатор тј. променљиву `rastuci` прогласимо за `false`. Пошто није неопходно учитати све елементе серије, чим установимо да је низ није растући можемо изаћи из петље (један начин да то урадимо је да услов петље ојачамо додавањем логичког индикатора `rastuci`, а други да употребимо наредбу `break`). По завршетку петље испитујемо вредност индикатора `rastuci` и у зависности од ње исписујемо одговарајућу поруку.

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

```
int N; // broj dana za koje se meri vreme plivaca
cin >> N;
```

```
// prethodni i tekuci clan niza vremena
int prethodno, tekuce;
bool rastuci = true;
cin >> prethodno;
for (int i = 1; i < N && rastuci; i++) {
    cin >> tekuce;
    if (prethodno >= tekuce)
        rastuci = false;
    prethodno = tekuce;
}
```

```
if (rastuci)
    cout << "Da" << endl;
else
    cout << "Ne" << endl;
```

Види групација решења овој задатку.

Задатак: Парно непарни

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Провера уређености остатака

Други начин да се провери дати услов је да се провери да ли остаци учитаних бројева при дељењу са два чине неопдајућу серију (серију у којој иду прво нуле, а затим јединице и никада се после јединице не јавља нула). Провера да ли је низ сортиран приказана је у задатку **Провера монотоности** и алгоритам је могуће засновати на некој од техника приказаних у том задатку.

```
bool OK = true; // da li je serija za sada ispravna
int prethodna; // prethodna karta
cin >> prethodna;
while (OK) { // dok ne naidjemo na gresku
    // ucitavamo tekucu kartu i proveravamo da je kraj ulaza
    int tekuca;
    if (!(cin >> tekuca))
        break;
    // ako je prethodna neparna, a tekuca parna
    if (prethodna % 2 > tekuca % 2)
        OK = false; // karte nisu dobro slozene
    prethodna = tekuca; // tekuca karta postaje prethodna
}
cout << (OK ? "da" : "ne") << endl; // ispisujemo rezultat
```

Види групација решења овој задатку.

Задатак: Растуће цифре

Написати програм којим се испитује да ли су цифре декадног записа природног броја растуће идући од цифре најмање тежине (цифре јединица).

Улаз: Са стандардног улаза се учитава природан број од 0 до 100000.

Израз: На стандардном излазу се исписује текстуални одговор DA или NE.

Пример 1

Улаз Излаз
97642 DA

Пример 2

Улаз Излаз
79653 NE

Решење

Решење задатка комбинује алгоритам одређивање серије цифара броја сдесна налево који смо описали у задатку **Број и збир цифара броја** и алгоритам линеарне претраге којим проверавамо да ли у серији цифара броја постоји цифра која је није већа (већ је мања или једнака) од претходне (слично као у задатку **Провера монотоности**).

Током претраге, потребно је да одржавати две променљиве – једну која садржи вредност претходне и другу која садржи вредност текуће цифре броја. Вредност претходне цифре иницијализоваћемо пре петље. Нај-природније решење је да се она иницијализује на вредност последње цифре броја (одређене као остатак при дељењу броја са 10) и да се пре уласка у петљу та последња цифра уклони (целобројним дељењем са 10).

У телу петље, које се извршава док број n има цифра тј. док не постане једнак нули, одређује и уклања се текућа цифра (последња цифра у тренутној вредности броја n). Ако се утврди да та цифра није већа од претходне, прекида се извршавање функције и враћа се вредност `false` - цифре нису у растућем поретку. На крају тела, врши се припрема за следећу итерацију тако што текућа цифра постаје претходна (вредност променљиве која чува текућу цифру додељује се променљивој која чува претходну цифру). Ако се не прекине извршавање петље (функције) зато што се појавила цифра која није већа од претходне, већ зато што су све цифре броја уклоњене (након што су подвргнуте траженој провери), функција враћа логичку вредност `true` - цифре су уређене у растућем поретку.

```
bool rastuceCifre(int n) {
    int prethodnaCifra = n % 10; n /= 10; // uklanjamo poslednju cifru
    while (n > 0) { // dok broj ne postane 0
        int tekucaCifra = n % 10; n /= 10; // uklanjamo poslednju cifru
        if (tekucaCifra <= prethodnaCifra) // pronasli smo uzastopne cifre
            return false; // u pogresnom redosledu
        prethodnaCifra = tekucaCifra; // pripremamo se za sledeci korak
    }
    return true;
}

int main() {
    int n;
    cin >> n;
    cout << (rastuceCifre(n) ? "DA" : "NE") << endl;
    return 0;
}
```

Задатак: Поређани датуми

Милица је направила списак рођендана својих другарица. Поређала је датуме њиховог рођења хронолошки (од најдавнијег до најскоријег). Напиши програм који проверава да ли је Милица то добро урадила.

Улаз: Са стандардног улаза учитава се број n ($1 \leq n \leq 100$). У наредних n линија налази се n исправних, међусобно различитих датума. Датуми су записани тако што су записани дан, месец и година, раздвојени размацима.

Излаз: У јединој линији стандардног излаза исписати DA ако су датуми исправно (растуће) поређани, тј. NE ако нису.

Пример

Улаз Излаз
3 NE
3 7 2005
14 8 2006
10 5 2006

Решење

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

Приметимо да је алгоритам који је потребно применити у овом задатку заправо идентичан оном примењеном у задатку **Провера монотоности**, једино што се уместо поређења бројева у овом задатку захтева поређење датума.

Потребно је проверити да ли у серији датума постоји неки који није после њему претходног. За то примењујемо алгоритам линеарне претраге (слично као у задатку **Негативан број**). Пошто се захтева поређење свака два суседна датума, потребно је да одржавамо претходни и текући датум. Као што је показано у задатку **Максимална разлика суседних**, на почетку је могуће иницијализовати претходни датум првим учитаним, на почетку петље учитавати текући датум, а онда на крају петље постављати претходни на тај текући или иницијализовати текући датум првим учитаним, на почетку петље постављати претходни датум на текући и након тога учитавати текући датум. У телу петље, након учитавања сваког текућег датума провераваћемо да ли је већи од претходног (да ли представља дан који долази хронолошки касније). Током претраге уводимо логичку променљиву која показује да ли су сви до сада учитани датуми у исправном редоследу и коју иницијализујемо `true`. Ако текући датум није већи од претходног, тада ту променљиву постављамо на `false` и прекидамо петљу (на пример, наредбом `break`). На крају петље, резултат пријављујемо у зависности од вредности те логичке променљиве.

Поредак између датума је лексикографски, ако се прво гледа година, затим месец и на крају дан (као што је то показано у задатку **Пунолетство**). Један начин да се ово реализује је да се сваки датум представи са три одвојена броја и да се, прегледности ради, поређење два датума издвоји у засебну функцију.

```
bool posle(int d1, int m1, int g1,
          int d2, int m2, int g2) {
    return g1 > g2 ||
           (g1 == g2 && m1 > m2) ||
           (g1 == g2 && m1 == m2 && d1 > d2);
}

void citajDatum(int& d, int& m, int& g) {
    cin >> d >> m >> g;
}

int main() {
    int n;
    cin >> n;
    int dPrethodni, mPrethodni, gPrethodni;
    citajDatum(dPrethodni, mPrethodni, gPrethodni);
    bool poredjani = true;
    for (int i = 1; i < n; i++) {
        int dTekuci, mTekuci, gTekuci;
        citajDatum(dTekuci, mTekuci, gTekuci);
        if (!posle(dTekuci, mTekuci, gTekuci,
                  dPrethodni, mPrethodni, gPrethodni)) {
            poredjani = false;
            break;
        }
        dPrethodni = dTekuci; mPrethodni = mTekuci; gPrethodni = gTekuci;
    }
    cout << (poredjani ? "DA" : "NE") << endl;
    return 0;
}
```

Види групачија решења овог задатка.

Задатак: Тестераст низ

За низ целих бројева одредити да ли је тестераст. Кажемо да је низ тестераст ако за његове чланове важи поредак $a_1 < a_2, a_2 > a_3, a_3 < a_4, a_4 > a_5$ итд. или $a_1 > a_2, a_2 < a_3, a_3 > a_4, a_4 < a_5$ итд.

Улаз: У првој линији стандардног улаза налази се природан број N ($2 \leq N \leq 100$). У следећих N линија налази се по један цео број из интервала од -10^9 до 10^9 . Бројеви представљају елементе низа.

Излаз: На стандардни излаз приказати поруку DA ако низ јесте тестераст. Иначе приказати поруку NE.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
6	DA	6	NE
2		1	
1		2	
3		3	
2		4	
4		5	
3		6	

Решење

Ово је још један од задатака у којем се проверава однос суседних елемената и који представља уопштење задатка **Провера монотоности**. Поново елементе читамо у петљи, одржавајући вредност претходног и текућег елемента серије и користимо алгоритам линеарне претраге (уведен у задатку **Негативан број**) да проверимо да ли у серији постоји елемент који нарушава задати однос. Разлика је у томе што уместо да стално проверавамо да ли је претходни елемент мањи од текућег, у овом случају наизменично проверавамо да ли је претходни елемент мањи од текућег, па да ли је претходни елемент већи од текућег итд.

Једно могуће решење је да се уведе помоћна логичка променљива која одређује да ли у наредном поређењу текући елемент треба да буде мањи или већи од претходног. Почетну вредност ове променљиве одређујемо на основу односа прва два елемента серије (њих морамо обрадити пре петље), а онда је у сваком кораку петље негирамо (ако се у једном кораку очекивало да претходни број буде мањи од текућег, у наредном кораку се очекује да претходни корак треба да буде већи од текућег). Приликом претраге поређење елемената се врши на основу текуће вредности те променљиве.

```
int n; // broj clanova
cin >> n;
int prethodni, tekuci; // dva uzastopna clana niza
cin >> prethodni >> tekuci; // ucitavamo prva dva clana
// za sada nismo naisli na elemente ciji je odnos pogresan i za niz
// mozemo reci da trenutno jeste testerast
bool testerast = true;
// ako je veci=true, tada naredni element treba da bude veci
// od prethodnog, a u suprotnom treba da bude manji od prethodnog
bool veci;
// na osnovu odnosa prva dva elementa odredjujemo kakav treba da
// bude odnos drugog i treceg elementa
if (prethodni > tekuci)
    veci = false;
else if (prethodni < tekuci)
    veci = true;
else
    testerast = false;
// obradjujemo ostale elemente dok ne obradimo svih n ili dok
// ne naidjemo na dva elementa u pogresnom redosledu
for (int i = 2; i < n && testerast; i++) {
    prethodni = tekuci; // ucitavamo naredni element
    cin >> tekuci;
    // da li je odnos trenutna dva ucitana elementa pogresan
    if ((veci && !(prethodni > tekuci)) ||
        (!veci && !(prethodni < tekuci)))
        testerast = false;
    // odnos naredna dva broja treba da bude suprotan
    veci = !veci;
}

// prijavljujemo rezultat
if (testerast)
```

3.2. ОСНОВНИ АЛГОРИТМИ НАД СЕРИЈАМА ЕЛЕМЕНАТА

```
cout << "DA" << endl;
else
    cout << "NE" << endl;
```

Пошто се у низу налазе бројеви, можемо понудити и мало другачије решење. Наиме, важи да је $x > y$ ако и само ако је $(-1) \cdot x < (-1) \cdot y$. Ако је $a_1 < a_2, a_2 > a_3, a_3 < a_4$ итд., тада је $1 \cdot a_1 < 1 \cdot a_2, (-1) \cdot a_2 < (-1) \cdot a_3, 1 \cdot a_3 < 1 \cdot a_4$ итд. Обратно, ако важи $a_1 > a_2, a_2 < a_3, a_3 > a_4$ итд., тада важи $(-1) \cdot a_1 < (-1) \cdot a_2, 1 \cdot a_2 < 1 \cdot a_3, (-1) \cdot a_3 < (-1) \cdot a_4$ итд. То значи да се провера да ли је низ тестераст своди на итеративно поређење релацијом мање узастопних елемената помножених наизменично са 1 па -1, односно -1 па 1 (те вредности памтимо у посебној бројевној променљивој чија почетна вредност зависи од односа прва два елемента низа, а која у сваком кораку петље мења знак). Нагласимо још и да је ово решење могло бити примењено зато што је условима задатка гарантовано да се на улазу не може јавити најмањи негативан број (проблем са њим је то што се множењем са -1 не може добити њему супротан број, јер се тај број обично не може исправно репрезентовати).

```
cin >> prethodni >> tekuci; // učitavamo prva dva člana
// za sada nismo naisli na elemente čiji je odnos pogresan i za niz
// mozemo reci da trenutno jeste testerast
bool testerast = true;
// znak odredjuje pozeljan odnos izmedju naredna dva elementa
// vrednost 1 ukazuje na to da prethodni treba da bude manji od tekuceg,
// a -1 da prethodni treba da bude veci od tekuceg
int znak;
// na osnovu odnosa prva dva elementa odredjujemo znak za odnos
// drugog i treceg i tako nadalje
if (prethodni > tekuci)
    znak = 1;
else if (prethodni < tekuci)
    znak = -1;
else
    testerast = false;

// obradjujemo ostale elemente dok ne obradimo svih n ili dok
// ne naidjemo na dva elementa u pogresnom redosledu
for (int i = 2; i < n && testerast; i++) {
    prethodni = tekuci; // učitavamo naredni element
    cin >> tekuci;
    // da li je odnos trenutna dva ucitana elementa pogresan
    if (znak * prethodni >= znak * tekuci)
        testerast = false;
    // odnos naredna dva broja treba da bude suprotan
    znak = -znak;
}
```

Још једно решење је да се у сваком тренутку памте три узастопна елемента низа (рецимо леви, средњи и десни) и да се проверава да ли је у сваком кораку знак разлике између десног и средњег различит од знака разлике између средњег и левог (испитивање да ли су два броја истог знака смо дискутовали у задатку [Два броја истог знака](#)).

```
int n; // broj clanova
cin >> n;
int levi, srednji, desni; // tri uzastopna člana niza
cin >> levi >> srednji; // učitavamo prva dva člana
// za sada nismo naisli na elemente čiji je odnos pogresan i za niz
// mozemo reci da trenutno jeste testerast (osim ako su prva dva
// elementa jednaka)
bool testerast = levi != srednji;
for (int i = 3; i <= n && testerast; i++) {
    cin >> desni; // učitavamo naredni element
    // proveravamo tekuci "zubac testere"
```

```

if (!razlicitogZnaka(srednji - levi, desni - srednji))
    testerast = false;
// pripremamo se za naredni korak
levi = srednji; srednji = desni;
}

```

Задатак: Продужавање титлова

Станислав је ђак првог разреда и тек учи да чита. Покушава да прочита титлове на цртаном филму који гледа током карантина, али му често недостаје неколико секунди да би их прочитао до краја. Његова сестра жели да му помогне тако што ће продужити трајање сваког титла за пет секунди. Међутим, некада је пауза до наредног титла краћа од 5 секунди, па да се титлови не би преклапали, она титл продужава тачно до почетка наредног. Последњи титл се сигурно може продужити за 5 секунди. Пошто је филм дугачак, ово је напоран посао. Напиши програм који би Станислављевој сестри помогао да овај посао брже заврши.

Улаз: Са стандардног улаза се уноси укупан број титлова n ($1 \leq n \leq 1000$), и након тога у наредних n линија подаци о титловима. За сваки титл се уноси време почетка и време краја (у односу на почетак филма), раздвојени размаком. Свако време задато је у облику `hh:mm:ss`. Филм не траје дуже од 3 сата.

Излаз: На стандардни излаз исписати податке о продуженим титловима (у истом формату у ком су унети).

Пример

Улаз	Излаз
5	5
00:01:43 00:01:48	00:01:43 00:01:53
00:01:56 00:01:59	00:01:56 00:02:04
00:02:17 00:02:23	00:02:17 00:02:24
00:02:24 00:02:26	00:02:24 00:02:30
00:02:30 00:02:38	00:02:30 00:02:43

Решење

Подацима о времену најједноставније можемо баратати ако их претворимо у секунде (протекле од претходне поноћи), слично као у задатку [Поноћ](#). Једноставности ради, издвојићемо функције за учитавање и испис времена и података о титлу (време почетка и краја). Након учитавања података претворићемо их у секунде, а пре исписивања ћемо их претворити у сате, минуте и секунде.

Титлове ћемо обрађивати у петљи и у сваком тренутку ћемо одржавати податке о текућем и наредном титлу. Текући титл ћемо учитати пре петље, а затим ћемо у петљи $n - 1$ пута учитавати наредни титл, поправљати текући (знајући време почетка наредног) и након тога наредни проглашавати текућим за следећу итерацију. Сваки текући титл у петљи обрађујемо тако што препишемо време почетка и мање од увећаног времена завршетка и времена почетка наредног титла. Последњи титл ћемо обрадити након петље, тако што ћемо преписати време почетка и увећано време завршетка.

```

int uSekunde(int h, int m, int s) {
    return h * 60 * 60 + m * 60 + s;
}

void odSekundi(int S, int& h, int& m, int& s) {
    s = S % 60; S /= 60;
    m = S % 60; S /= 60;
    h = S;
}

// ucitavamo vreme sa standardnog ulaza u formatu hh:mm:ss
int ucitajVreme() {
    int h, m, s;
    char c;
    cin >> h >> c >> m >> c >> s;
    return uSekunde(h, m, s);
}

```

3.3. УГНЕЖЂЕНЕ ПЕТЉЕ

```
// ucitavamo podatke o titlu (pocetak i kraj u formatu hh:mm:ss)
void ucitajTitl(int& start, int& end) {
    start = ucitajVreme();
    cin >> ws;
    end = ucitajVreme();
    cin >> ws;
}

// ispisujemo podatke o vremenu u formatu hh:mm:ss
void ispisiVreme(int S) {
    int h, m, s;
    odSekundi(S, h, m, s);
    cout << setfill('0') << setw(2) << h << ":"
         << setfill('0') << setw(2) << m << ":"
         << setfill('0') << setw(2) << s;
}

// ispisujemo podatke o titlu (pocetak i kraj u formatu hh:mm:ss)
void ispisiTitl(int start, int end) {
    ispisiVreme(start);
    cout << " ";
    ispisiVreme(end);
    cout << endl;
}

int main() {
    // ukupan broj titlova
    int n;
    cin >> n;
    cout << n << endl;
    // ucitavamo podatke o tekucem titlu
    int start_t, end_t;
    ucitajTitl(start_t, end_t);
    for (int i = 0; i < n - 1; i++) {
        // ucitavamo podatke o narednom titlu
        int start_n, end_n;
        ucitajTitl(start_n, end_n);
        // obradjujemo podatke o tekucem titlu, uzevsi u obzir pocetak narednog
        ispisiTitl(start_t, min(end_t + 5, start_n));
        // naredni postaje tekuci za sledecu iteraciju
        start_t = start_n; end_t = end_n;
    }
    // poslednji titl se sigurno produzava
    ispisiTitl(start_t, end_t+5);
}
```

3.3 Угнежђене петље

Унутар тела петље може се налазити друга петља. За такве петље кажемо да су *угнежђене*. У сваком кораку (свакој итерацији) спољашње петље унутрашња петља се извршава изнова. У наредним задацима приказаћемо употребу угнежђених петљи.

3.3.1 Генерисање свих могућности (варијације, комбинације, пермутације, партиције)

Многи практични задаци решавају се тако што се наброје и испитају све могућности. Свака могућност се често може представити неком уређеном торком елемената. На пример, ако посматрамо могуће исходе бацања две коцкице, сваки исход се може представити уређеним паром бројева од 1 до 6. Када је торка мале дужине

(двочлана, трочлана, четворочлана), тада све могућности можемо излистати помоћу угнежђених петљи. Ту ћемо технику приказати помоћу наредних задатака (кроз њих ћемо се и упознати са основним комбинаторним објектима: варијацијама, комбинацијама, партицијама, цикличким пермутацијама).

Задатак: Бројеви у датој основи

Бројеви у основи b се могу записати помоћу цифара $0, 1, \dots, b - 1$. Ако је основа већа од 10, тада се уместо цифара користе слова енглеске абетеде (цифра 10 записује се са a , цифра 11 са b итд.). Напиши програм који исписује све троцифрене бројеве у датој основи.

Улаз: Са стандардног улаза се учитава основа b ($2 \leq b \leq 16$).

Излаз: На стандардни излаз исписати све троцифрене бројеве у основи b , поређане растући по вредности (бројеве исписати са водећим нулама).

Пример

Улаз	Излаз
2	000
	001
	010
	011
	100
	101
	110
	111

Решење

На сваком од три места могуће је исписати било коју цифру од 0 до $b - 1$. Стога решење можемо једноставно постићи помоћу три угнежђене петље - по једну за сваку од три цифре. Тројке цифара ће бити лексикографски растући уређене, што ће тачно одговарати редоследу бројева по величини, како се тражи у задатку.

Пошто ће се користити и основе веће од 10, потребно је да направимо и функцију која ће на основу нумеричке вредности цифре одредити карактер који представља ту цифру. Имплементација те функције захтева баратање са карактерским типом података (слично као, на пример, у задацима [Трансформација карактера](#) и [Абецедно огледало](#)).

- Ако је вредност цифре мања од 10, карактер добијамо тако што саберемо кôд (ASCII тј. Unicode) карактера 0 са вредношћу цифре.
- Ако је вредност цифре већа од 10, карактер добијамо тако што саберемо кôд карактера a са вредношћу цифре умањеном за 10.

Рецимо и да се у овом задатку заправо врши генерисање и исписивање свих *варијација са понављањем* дужине 3 са b елемената. Варијације без понављања размотрене су у задатку [Варијације тројки](#).

```
#include <iostream>
```

```
using namespace std;
```

```
char cifra(int c) {
    return c < 10 ? '0' + c : 'a' + (c - 10);
}
```

```
int main() {
    int b;
    cin >> b;
    for (int c2 = 0; c2 < b; c2++)
        for (int c1 = 0; c1 < b; c1++)
            for (int c0 = 0; c0 < b; c0++)
                cout << cifra(c2) << cifra(c1) << cifra(c0) << endl;
    return 0;
}
```

3.3. УГНЕЖЂЕНЕ ПЕТЉЕ

Задатак: Варијације тројки

Сваки од три другара има одређени број јабука, али никоја два од њих немају исти број јабука. Ако се зна највећи могући број јабука који сваки од другара може да има, напиши програм који исписује све могуће тројке бројева јабука које они могу да имају.

Улаз: Са стандардног улаза се уноси број n ($1 \leq n \leq 20$) - највећи број јабука које сваки од другара може да има.

Излаз: На стандардни излаз исписати све могуће бројеве јабука које другови могу да имају, уређене лексикографски.

Пример

Улаз	Излаз
2	0 1 2
	0 2 1
	1 0 2
	1 2 0
	2 0 1
	2 1 0

Решење

У задатку се тражи набрајање свих *варијација без понављања* дужине 3 од n елемената. Најлакши начин да се оне наброје је да се употребе три угнежђене петље - по једна за сваког од три другара. Бројач сваке петље (обележимо их са d_1 , d_2 и d_3) узима редом вредности од 0 до n . Ако би се у телу унутрашње петље вршио испис вредности променљивих, излистале би се све варијације (па и оне са понављањем). Да би се избегло понављање, потребно је пре исписа проверити да ли у вредности све три променљиве различите тј. да ли важи да је $d_1 \neq d_2$ и $d_1 \neq d_3$ и $d_2 \neq d_3$. Приметимо да се овде заправо ради о филтрирању, тј. издвајању само оних елемената који задовољавају неко дато својство (слично као у задатку). Услов $d_1 \neq d_2$ има смисла проверити пре него што се уопште уђе у трећу петљу.

```
// najveci broj jabuka nekog drugara
int n;
cin >> n;
for (int d1 = 0; d1 <= n; d1++) // broj jabuka prvog
    for (int d2 = 0; d2 <= n; d2++) // broj jabuka drugog
        if (d1 != d2)
            for (int d3 = 0; d3 <= n; d3++) // broj jabuka treceg
                if (d1 != d3 && d2 != d3)
                    cout << d1 << " " << d2 << " " << d3 << endl;
```

Задатак: Мали лото

У једном одељењу су одлучили да у склопу новогодишње приредбе организују мало извлачење игре лото. Да би повећали шансе за добитак, одлучили су да се извлаче само три куглице. Напиши програм који исписује које све комбинације могу бити извучене, ако се зна да у бубњу има n различитих куглица обележених бројевима од 1 до n .

Улаз: Са стандардног улаза се уноси број n ($4 \leq n \leq 20$).

Излаз: На стандардни излаз испиши све комбинације, при чему су бројеви у свакој комбинацији сортирани растуће, а комбинације су лексикографски сортиране.

Пример

Улаз	Излаз
4	1 2 3
	1 2 4
	1 3 4
	2 3 4

Решење

Већ је у тексту задатка наглашено да се у овом задатку тражи исписивање свих *комбинација* (пошто су у игри лото сви бројеви различити, ради се о *комбинацијама без понављања*).

Најједноставније решење подразумева три угнежђене петље, по једну за сваку од три лоптице. Спољашња петља набрајаће редом бројеве на првој лоптици, средња на другој, а унутрашња на трећој, при чему су лоптице сортиране растући, како се тражи у тексту задатка. Стога бројач средње петље увек мора бити већи од бројача спољашње, а бројач унутрашње петље увек мора бити већи од бројача средње петље. Зато границе за прву бројачку променљиву b_1 можемо поставити на 1 до n , средњу b_2 на $b_1 + 1$ до n , а унутрашњу b_3 на $b_2 + 1$ до n . Заправо, горње границе су мало мање (за спољашњу је то $n - 2$, а средњу $n - 1$), међутим, и за границе n програм ће радити коректно, јер ће нека од унутрашњих петљи бити празна (на пример, за $b_1 = n$, b_2 креће од $n + 1$ и траје док је $b_2 \leq n$, што је у самом старту нарушено).

```
int n;
cin >> n;
for (int b1 = 1; b1 <= n; b1++) // broj na loptici 1
    for (int b2 = b1 + 1; b2 <= n; b2++) // broj na loptici 2
        for (int b3 = b2 + 1; b3 <= n; b3++) // broj na loptici 3
            cout << b1 << " " << b2 << " " << b3 << endl;
```

Задатак: Коцкице за јамб

Исписати све резултате бацања три коцкице за јамб у којима је збир бројева једнак задатом броју X (природан број од 3 до 18) ако је редослед коцкица битан (на пример, 2, 2, 3 није исто као 2, 3, 2).

Улаз: У једној линији стандардног улаза наводи се природан број X ($3 \leq X \leq 18$) који представља збир бројева на којима су се зауставиле коцкице.

Издаз: У свакој линији стандардног излаза исписују се три природна броја (бројеви од 1 до 6) који представљају резултат бацања три коцкице. Резултате приказати сортиране лексикографски (у растућем редоследу троцифрених бројева који се формирају од три цифре на коцкицама).

Пример

Улаз	Издаз
5	1 1 3 1 2 2 1 3 1 2 1 2 2 2 1 3 1 1

Решење

Ако са a , b и c обележимо бројеве који су добијени на три коцкице важи да је $a + b + c = X$, и да је $1 \leq a \leq 6$, $1 \leq b \leq 6$ и $1 \leq c \leq 6$. Проблем разлагања броја на збир неколико бројева назива се *партиционисање* тј. проналажења *партиција* броја. У овом случају разматрамо партиције дужине 3 (партиције на тачно три сабирка).

Три петље

Наивно решење је да се помоћу три угнежђене петље наброје све могућности од 1 до 6 за a , b и c и да се у телу унутрашње петље проверава да ли је $a + b + c = X$.

```
// zbir tri broja na kockicama
int X;
cin >> X;
for (int a = 1; a <= 6; a++) // vrednost na prvoj kockici
    for (int b = 1; b <= 6; b++) // vrednost na drugoj kockici
        for (int c = 1; c <= 6; c++) // vrednost na trecoj kockici
            if (a + b + c == X) // ako je zbir jednak datom
                cout << a << " " << b << " " << c << endl; // ispisujemo resenje
```

Две петље

Бољи приступ је испробавати само све могућности за a и b , а онда израчунавати c као $X - a - b$ и проверавати да ли добијени резултат припада интервалу од 1 до 6.

Пажљивијом анализом можемо границе у петљама додатно оптимизовати, али пошто је укупан број могућности мали и ово решење је сасвим довољно ефикасно.

3.3. УГНЕЖЂЕНЕ ПЕТЉЕ

```
// zbir tri broja na kockicama
int X;
cin >> X;
for (int a = 1; a <= 6; a++) // vrednost na prvoj kockici
    for (int b = 1; b <= 6; b++) { // vrednost na drugoj kockici
        int c = X - a - b; // broj takav da je a+b+c = X
        if (1 <= c && c <= 6) // ako moze da bude vrednost na trecoj kockici
            // ispisujemo resenje
            cout << a << " " << b << " " << c << endl;
    }
}
```

Задатак: Комбинације поена

На кошаркашкој утакмици кошеви се бодују са 1, 2 или 3 поена. Потребно је одредити све комбинације броја кошева који су бодовани са 1 поеном, са 2 и са 3 поена ако је познат укупан број поена једног тима на кошаркашкој утакмици.

Улаз: У првој линији стандардног улаза налази се укупан број поена на кошаркашкој утакмици (природан број између 40 и 150).

Издаз: Свака линија стандардног излаза треба да садржи број кошева који су бодовани са 3, затим са 2 и на крају са 1. Комбинације треба да буду уређене лексикографски.

Пример

Улаз	Издаз
6	6
	0 0 6
	0 1 4
	0 2 2
	0 3 0
	1 0 3
	1 1 1
	2 0 0

Решење

Овај задатак је донекле сличан, на пример, задатку **Коцкице за јамб** (и овај пут се ради о партиционисању, али овај пут су сабирцима придружене тежине 1, 2 и 3), па ће и технике које ћемо применити у решењу донекле бити сличне онима примењеним у тим задацима.

Три петље

Једно решење је да помоћу три угнежђене петље испитамо све могућности за број кошева са три поена (t), број кошева са два поена (d) и број кошева за један поен (j). Ако је број поена p , границе ових променљивих веома грубо можемо одредити ако приметимо да број кошева са 1 поеном j може имати вредности целих бројева од 0 до p , пошто мора да важи да је $2 \cdot d \leq p$, број кошева са 2 поена d може имати вредности целих бројева од 0 до $\lfloor \frac{p}{2} \rfloor$, тј. до вредности целобројног количника бројева p и 2 (види задатак **Поклони**), а пошто мора да важи да је $3 \cdot t \leq p$, број кошева са 3 поена може имати вредности од 0 до $\lfloor \frac{p}{3} \rfloor$.

Због траженог редоследа приказивања прво фиксирамо број t узимајући све могуће вредности од најмање (то је 0) до највеће (то је $\lfloor \frac{p}{3} \rfloor$), па затим број (d) од најмање вредности (то је 0) до највеће (то је $\lfloor \frac{p}{2} \rfloor$) и на крају број кошева са 1 поеном (j) од најмање вредности (то је 0) до највеће (то је p). У телу унутрашње петље проверавамо да ли је укупан број поена једнак вредности p (тј. проверавамо да ли је услов $3 \cdot t + 2 \cdot d + j = p$ испуњен).

```
int poeni;
cin >> poeni;
for (int t = 0; t <= poeni / 3; t++) // trojke
    for (int d = 0; d <= poeni / 2; d++) // dvojke
        for (int j = 0; j <= poeni; j++) // jedinice
            if (3 * t + 2 * d + j == poeni)
                cout << t << " " << d << " " << j << endl;
```

Две петље

Мало прецизнијом анализом можемо добити ефикаснији програм. Наиме, горњу границу променљиве t одређујемо из услова $3 \cdot t \leq p$ и добијамо вредност $\lfloor \frac{p}{3} \rfloor$. Пошто мора да важи да је $3 \cdot t + 2 \cdot d \leq p$, за фиксирану вредност t , границу за променљиву d можемо одредити као $\lfloor \frac{p-3 \cdot t}{2} \rfloor$. Такође, ако су познате вредности t , d и p , из услова $3 \cdot t + 2 \cdot d + j = p$ можемо израчунати да је $j = p - 3 \cdot t - 2 \cdot d$, чиме избегавамо унутрашњу (трећу) петљу и непотребно испитивање различитих нетачних вредности за j .

```
int poeni;
cin >> poeni;
for (int t = 0; t <= poeni / 3; t++) // 3*t <= poeni
    for (int d = 0; d <= (poeni - 3*t) / 2; d++) { // 3*t+2*d <= poeni
        int j = poeni - (3*t + 2*d);
        cout << t << " " << d << " " << j << endl;
    }
}
```

Напоменимо и да није неопходно било израчунавати експлицитно горње границе, и итерација може бити контролисана оригиналним условима.

```
int poeni;
cin >> poeni;
for (int t = 0; 3 * t <= poeni; t++) // trojke
    for (int d = 0; 3 * t + 2 * d <= poeni; d++) { // dvojke
        int j = poeni - (3*t + 2*d); // jedinice
        cout << t << " " << d << " " << j << endl;
    }
}
```

Задатак: Цикличне пермутације

Цикличним померањем за једно место улево низа бројева $x_1, x_2, x_3, \dots, x_n$ добијамо $x_2, x_3, \dots, x_n, x_1$, ако вршимо циклично померање за два места улево добијамо $x_3, \dots, x_n, x_1, x_2$. Добијени низови представљају цикличне пермутације полазног низа.

Написати програм којим се за дати природан број n приказују низови бројева добијени цикличним померањем низа бројева $1, 2, \dots, n$ редом за $0, 1, 2, \dots, n - 1$ места улево.

Улаз: Прва линија стандардног улаза садржи природан број $n \leq 30$.

Излаз: Стандардни излаз садржи n линија, у којима су приказани тражени низови бројева, бројеви у низовима међусобно су одвојени бланко знаком.

Пример

Улаз	Излаз
4	1 2 3 4
	2 3 4 1
	3 4 1 2
	4 1 2 3

Решење

Потребно је за свако i која узима вредности од 1 до n приказати низ бројева облика $i, i + 1, \dots, n, 1, 2, \dots, i - 1$.

Задатак можемо решити петљом у којој коришћењем бројачке променљиве i која узима вредности од 1 до n , прво прикажемо једном петљом бројеве од i до n , а затим другом петљом бројеве од 1 до $i - 1$.

```
int n;
cin >> n;
for (int i = 1; i <= n; i++) {
    for (int j = i; j <= n; j++)
        cout << j << " ";
    for (int j = 1; j < i; j++)
        cout << j << " ";
    cout << endl;
}
```

Приметимо да бројеве $i, i + 1, \dots, n, 1, 2, \dots, i - 1$ можемо приказати у једној петљи, увећавањем бројачке променљиве редом за $0, 1, \dots, n - 1$ и коришћењем остатка при дељењу са n (пошто петља креће од 1, а не

3.3. УГНЕЖЉЕНЕ ПЕТЉЕ

од 0, од збира i и j потребно је одузети 1, пронаћи остатак, а затим додати 1, слично као што је то приказано у задатку Car).

```
int n;
cin >> n;
for (int i = 1; i <= n; i++) {
    for (int j = 0; j < n; j++)
        cout << (i + j - 1) % n + 1 << " ";
    cout << endl;
}
```

Такође, у унутрашњој петљи можемо водити посебну променљиву која чува вредност која се исписује. Ту променљиву постављамо на вредност i и након сваког исписа је увећавамо за 1, при чему проверавамо да ли је након увећања вредност већа од n . Ако јесте, враћамо је на вредност 1.

```
int n;
cin >> n;
for (int i = 1; i <= n; i++) {
    int p = i;
    for (int j = 1; j <= n; j++) {
        cout << p << " ";
        p++;
        if (p > n)
            p = 1;
    }
    cout << endl;
}
```

Задатак: Троуглови целобројних страница, задатог обима

Написати програм којим се одређују дужине страница троуглова a , b , c , такве да су a , b и c природни бројеви за које важи да је $a \leq b \leq c$ и да је обим троугла $a + b + c$ једнак датом природном броју O .

Улаз: У једној линији стандардног улаза задат је природан број O ($3 \leq O \leq 250$) који представља обим троугла.

Изназ: У свакој линији стандардног излаза налазе се три природна броја одвојена празнином, који представљају могуће дужине страница троугла задатог обима. Тројке треба да буду лексикографски сортиране.

Пример

Улаз	Изназ
7	1 3 3 2 2 3

3.3.2 Сви подсегменти серије

Задатак: Сви суфикси низа бројева од 1 до n

Својевремено је постојала реклама за фирму “Југодрво” у којој се појављивала песмица “Југодрво-угодрво-годрво-одрво-дрво-рво-во-о” у којој су се одређивали сви суфикси дате речи. Слично, али мало једноставније је исписати све суфиксе низа природних бројева од 1 до n . Напиши програм који за дато n исписати све чланове низа $1, 2, 3, \dots, n, 2, 3, 4, \dots, n, 3, 4, \dots, n, \dots, n - 1, n, n$.

Улаз: Са стандардног улаза се учитава природан број n у границама од 1 до 30.

Изназ: Бројеви се исписују на стандардни излаз, раздвојени размацима, сваки суфикс у посебном реду.

Пример

Улаз	Изназ
4	1 2 3 4 2 3 4 3 4 4

Решење

Ако је n учитана вредност (број сегмената), прво је потребно исписати све бројеве од 1 до n , затим бројеве од 2 до n и тако даље, док се на крају не испишу бројеви од $n - 1$ до n и затим бројеви од n до n тј. само број n . Дакле, за сваку вредност i од 1 до n треба исписати све вредности од i до n . Пошто се све вредности од i до n могу исписати исписати коришћењем петље (као у задатку **Бројеви од а до b**), задатак можемо решити коришћењем угнежђених петљи. Спољашњом петљом, чија променљива i узима вредности од 1 до n , обезбеђујемо n извршавања унутрашње петље чија променљива j узима вредности од i до n , док у сваком кораку унутрашње петље исписујемо вредност његове променљиве j .

```
int n; // broj segmenata
cin >> n;
for (int i = 1; i <= n; i++) { // redni broj segmenta
    for (int j = i; j <= n; j++) // ispis elemenata i-tog desnog segmenta
        cout << j << " ";
    cout << endl;
}
```

Задатак: Сви префикси низа бројева од 1 до n

За дато n исписати све префиксе низа природних бројева који почињу са 1 и завршавају се са бројем мањим или једнаким од n .

Улаз: Са стандардног улаза се учитава цео број n ($1 \leq n \leq 20$).

Излаз: На стандардни излаз исписати све тражене префиксе, сваки у посебном реду у растућем редоследу дужина. Иза сваког елемента сваког префикса треба да буде исписан по један размак.

Пример

Улаз	Излаз
3	1
	1 2
	1 2 3

Решење

Задатак је веома сличан задатку и једноставно се решава помоћу две угнежђене петље. Спољна петља набраја све десне крајеве i префикса (то су редом бројеви од 1 до n), а у унутрашњој се набрајају елементи префикса (то су редом бројеви од 1 до i).

```
int n;
cin >> n;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= i; j++)
        cout << j << " ";
    cout << endl;
}
```

Задатак: Све подречи

Напиши програм који ће исписати све подречи дате речи и то у растућем редоследу почетних позиција и растућем редоследу дужине. Напомена: у већини језика је слово на позицији i речи s могуће прочитати изразом $s[i]$.

Улаз: Са стандардног улаза се учитава једна реч састављена само од малих слова енглеске абетецеде.

Излаз: На стандардни излаз исписати тражене подречи, сваку у посебном реду.

Пример

Улаз	Излаз
abc	a
	ab
	abc
	b
	bc
	c

Решење

Пошто је редослед такав да се прво исписују све подречи које почињу на позицији 0, затим све подречи на позицији 1 итд., у спољашњој петљи вршићемо итерацију кроз почетни индекс сваке подречи и бројачка променљива i ће узимати редом вредности од 0 до $n - 1$, где је n дужина речи. У језику C++ њу можемо одредити помоћу методе `size()`. У унутрашњој петљи ћемо вршити итерацију кроз завршни индекс сваке подречи и бројачка променљива те петље ће редом узимати вредности од i па до $n - 1$.

Када је фиксиран сегмент $[i, j]$ подреч можемо исписивати карактер по карактер у новој петљи.

```
string s;
cin >> s;
for (int i = 0; i < s.size(); i++)
    for (int j = i; j < s.size(); j++) {
        for (int k = i; k <= j; k++)
            cout << s[k];
        cout << endl;
    }
```

Види групаџија решења овој задајџка.

Задатак: Све подречи дужине n

Дата је реч и природан број n . Напиши програм који исписује све њене подречи (сегменте узастопних карактера) дужине n .

Улаз: Први ред стандардног улаза садржи реч састављену од малих слова енглеског алфабета, а други садржи број n .

Излаз: На стандардни излаз исписати све тражене подречи, с лева на десно, сваку у посебном реду.

Пример

Улаз	Излаз
abcdef	abc
3	bcd
	cde
	def

Решење

Претпоставимо да је реч дужине m и да се траже подречи дужине n . Ако подреч почиње на позицији i тада су њени карактери на позицијама $i, i + 1, \dots, i + n - 1$. Почетне позиције i подречи крећу од нуле и увећавају се све док је последњи карактер подречи унутар речи тј. док је $i + n - 1 < m$ односно $i < m - n + 1$.

У унутрашњој петљи можемо итерацију вршити по j које узима вредности од i до $i + n - 1$ или по j које узима вредности од 0 до $n - 1$, али се тада исписује карактер на позицији $i + j$.

```
string s;
cin >> s;
int n;
cin >> n;
for (int i = 0; i + n - 1 < s.size(); i++) {
    for (int j = 0; j < n; j++)
        cout << s[i + j];
    cout << endl;
}
```

Задатак: Све подречи по опадајућој дужини

За унету реч исписати све подречи у редоследу опадајуће дужине и растућих левих граница за речи исте дужине.

Улаз: Са стандардног улаза се уноси једна реч састављена само од малих слова енглеске абецеде.

Излаз: На стандардни излаз исписати тражене подречи, сваку у посебном реду.

Пример

Улаз	Изназ
abc	abc
	ab
	bc
	a
	b
	c

Решење

Бројачка променљива d спољашње петље одређиваће дужину подречи и узимаће вредности од n па уназад до 1, док ће унутрашња бројачка променљива i одређивати индекс почетка подречи и узимаће вредности од 0 па све док је $i + d < n$.

Знајући почетак и дужину подречи саму подреч можемо исписати помоћу нове петље у којој се исписује један по један њен карактер.

```
string s;
cin >> s;
// duzina reci
for (int d = s.size(); d > 0; d--)
    // rec duzine d je odredjena indeksima [i, i + d - 1]
    for (int i = 0; i + d - 1 < s.size(); i++) {
        for (int j = i; j < i + d; j++)
            cout << s[j];
        cout << endl;
    }
```

Задатак: Цикличне подречи

Напиши програм који исписује све подречи које се могу добити читањем слова дате речи кренувши од неког слова, у круг, назад до тог слова.

Улаз: Са стандардног улаза се учитава реч састављена од малих слова, не дужа од 100 карактера.

Изназ: На стандардни излаз исписати тражене цикличне пермутације те речи.

Пример

Улаз	Изназ
zdravo	zdravo
	dravoz
	ravozd
	avozdr
	vozdra
	ozdrav

3.3.3 Цртежи помоћу карактера**Задатак: Квадрат од звездица**

Напиши програм који исцртава квадрат од карактера * (као што је приказано у примерима).

Улаз: Са стандардног улаза се учитава природан број n ($1 \leq n \leq 20$), који представља димензију квадрата (број звездица у свакој врсти и колони).

Изназ: На стандардни излаз исписати тражени цртеж.

Пример

Улаз	Изназ
4	****

Решење

3.3. УГНЕЖЂЕНЕ ПЕТЉЕ

Можемо приметити да се квадрат састоји од n редова, а да се у сваком реду налази n звездица. Зато програм можемо организовати помоћу угнежђених петљи. Спољна петља се извршава n пута и у њеном телу се испишује један ред. То се врши тако што се у телу унутрашње петље која се извршава n пута испишује једна звездица, а након те унутрашње петље се испишује прелазак у нови ред. У језику C++ се звездица испишује помоћу `cout << '*'`; а прелазак у нови ред помоћу `cout << endl`;

```
int n;
cin >> n;
for (int i = 0; i < n; i++) { // ponavljamo n puta
    for (int j = 0; j < n; j++) // iscrtavamo n zvezdica
        cout << "*";
    cout << endl;           // prelazimo u novi red
}
```

Задатак: Троугао од звездица

Напиши програм који исцртава троугао какав је приказан у примеру.

Улаз: Са стандардног улаза се учитава један природан број n ($1 \leq n \leq 30$), који представља висину троугла (број редова које цртеж садржи).

Издаз: На стандардни издаз се испишује тражени троугао, исписивањем карактера *, размака (карактера бланко) и преласка у нови ред. После сваке последње звездице у свакој врсти прећи у наредни ред (не исписивати размаке после звездица).

Пример

Улаз	Издаз
4	* *** **** *****

Решење

За дати број n троугао садржи n редова. Сваки ред ћемо исписивати у петљи у којој променљива i мења вредност од 0 до $n - 1$. Размотримо случај $n = 4$. У првом реду (када је $i = 0$) потребно је исписати три размака и једну звездицу, у другом (када је $i = 1$) потребно је исписати два размака и три звездице, у трећем (када је $i = 2$) потребно је један размак и пет звездица и у четвртном (када је $i = 3$) потребно је исписати нула размака и седам звездица. Може се закључити да је број размака једнак $n - i - 1$, а број звездица једнак $2i + 1$ (иако је извођење ових релација прилично очигледно, можемо приметити да број размака чини аритметички низ који креће од $n - 1$ и чија је разлика суседних елемената -1 , док број звездица чини аритметички низ који креће од 1 и чија је разлика суседних елемената 2 и везе извести на основу формула за i -ти члан аритметичког низа). Исписивање датог броја појављивања неког карактера можемо реализовати једноставно у петљи. Тако задатак можемо решити помоћу две унутрашње петље (једне у којој се испишују размаци и једне у којој се испишују звездице).

```
int n;
cin >> n;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n-i-1; j++)
        cout << ' ';
    for (int j = 0; j < 2*i+1; j++)
        cout << '*';
    cout << endl;
}
```

Једноставности ради исписивање датог броја појављивања неког карактера можемо реализовати у засебној функцији, коју ћемо онда позвати у различитим контекстима (једном за исписивање $n - i - 1$ размака, а други пут за исписивање $2i + 1$ звездица). Након исписа звездица, као последњи корак тела спољашње петље потребно је да пређемо у наредни ред.

```
int n;
cin >> n;
```

```

int brojZvezdica = 1, brojRazmaka = n-1;
for (int i = 0; i < n; i++) {
    ispisiNPutu(' ', brojRazmaka);
    ispisiNPutu('*', brojZvezdica);
    cout << endl;
    brojRazmaka--; brojZvezdica += 2;
}

```

Број звездица и размака није неопходно експлицитно израчунавати, већ је могуће одржавати две посебне променљиве у којима се памте ови бројеви.

```

int n;
cin >> n;

int brojRazmaka = n-1;
int brojZvezdica = 1;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < brojRazmaka; j++)
        cout << ' ';
    brojRazmaka -= 1;
    for (int j = 0; j < brojZvezdica; j++)
        cout << '*';
    brojZvezdica += 2;
    cout << endl;
}

```

Задатак: Троугао од речи

Напиши програм који исцртава троугао чије су ивице састављене од карактера дате речи. Реч се добија читањем слова са леве и десне ивице троугла наниже, док се на доњој ивици налази палиндром чија је десна половина та дата реч.

Улаз: Са стандардног улаза се учитава реч дужине између 3 и 20 карактера.

Излаз: На стандардни излаз исписати тражени троугао.

Пример

Улаз	Излаз
bravo	<pre> b g g a a v v ovarbravo </pre>

Решење

Можемо приметити да постоје три суштински различита случаја. Нека је n дужина речи.

У првом реду се исписује $n - 1$ размака и прво слово речи.

Нека су наредни редови обележени бројевима од 1 до $n - 2$ тј. нека у петљи променљива i узима вредности од 1 до $n - 2$. У телу те петље исписујемо прво $n - i - 1$ размака, затим слово речи на позицији i , након тога $2 \cdot i - 1$ размака и поново слово на позицији i .

На крају, у последњем реду исписујемо слова речи на позицијама од $n - 1$ до 0 и затим од 1 до $n - 1$.

```

string s;
cin >> s;
int n = s.size();

// први ред садржи само једно слово
ispisiNPutu(' ', n-1);
cout << s[0] << endl;

```

3.3. УГНЕЖЂЕНЕ ПЕТЉЕ

```
// srednji redovi sadrže po tačno dva slova
for (int i = 1; i < n - 1; i++) {
    ispisiNPuti(' ', n-1-i);
    cout << s[i];
    ispisiNPuti(' ', 2*i-1);
    cout << s[i];
    cout << endl;
}

// poslednji red sadrži 2n-1 slova
for (int i = n-1; i > 0; i--)
    cout << s[i];
for (int i = 0; i < n; i++)
    cout << s[i];
cout << endl;
```

Задатак: Ромб од звездица

Напиши програм који исписује ромб направљен од звездица (како је приказано у примеру).

Улаз: Са стандардног улаза се уноси природан број n ($3 \leq n \leq 20$), који представља димензију ромба (број врста и колона).

Излаз: На стандардни излаз исцртати ромб.

Пример

Улаз	Излаз
5	***** ***** ***** ***** *****

Задатак: Ћилим од звездица

Напиши програм који испрџава ћилим састављен од звездица (квадрат из чије је средине уклоњен ромб, како је приказано у примерима).

Улаз: Са стандардног улаза се учитава број n ($1 \leq n \leq 20$).

Излаз: На стандардни излаз исписати ћилим.

Пример 1

Улаз	Излаз
3	***** ** ** * * ** ** *****

Пример 2

Улаз	Излаз
4	***** *** ** ** ** * * ** ** *** ** *****

Задатак: V од звездица

Напиши програм који испрџава латиничко слово V састављено од звездица. Слово се простире у n редова и n колона.

Улаз: Са стандардног улаза се учитава број n ($1 \leq n \leq 20$).

Излаз: На стандардни излаз исписати звездице, као у примерима.

Пример 1

Улаз Излаз
 3 * *
 * *
 * *
 * *
 **
 **

Пример 2

Улаз Излаз
 4 * *
 * *
 * *
 * *
 * *
 * *
 **
 **

3.3.4 Генерисање неких правилних серија

Задатак: Карирана застава

На часу ликовног ђаци су добили задатак да за предстојећу трку око центра града, од већ постојећих квадратних комада старих застава, направе правоугаону заставу димензија која ће, као што је на аутомобилским тркама обичај бити карирана, црно-бела. Након завршеног шивења, видели су да им није баш све ишло од руке. Циљ који треба да постигну је да ниједно црно поље нема за суседно бело, и обрнуто (суседна поља су лево, десно, горње и доње) и зато морају још да префарбају поља заставе која нису ваљана са што мање боје. Напиши програм који одређује колико најмање поља морају да офарбају.

Улаз: У првој линији улаза налазе се два цела броја одвојена размаком r и k ($1 \leq r, k \leq 3000$), број редова и број колона заставе. Следећих r редова садржи по k знакова С или В (без размака).

Излаз: У једини ред излаза треба исписати један цео број који представља минималан број квадратних делова заставе које је потребно префарбати да би се добила црно бела карирана застава.

Пример 1

Улаз Излаз
 3 3 4
 ССС
 ССС
 ССС

Пример 2

Улаз Излаз
 3 4 0
 СВСВ
 ВСВС
 СВСВ

Пример 3

Улаз Излаз
 4 2 2
 ВС
 ВС
 СВ
 ВС

Решење

Постоје само два начина да се застава тако офарба, па да она буде црно бела карирана: поље на позицији (0, 0) може да буде бело и може да буде црно.

У варијанти у којој је поље на позицији (0, 0) бело, сва бела поља на позицијама (i, j) где је $i + j$ паран број и сва црна поља на позицијама (i, j) где је $i + j$ непаран број су добро обојена. Ако знамо број таквих поља m , тада можемо једноставно одредити тражено решење задатка. Наиме, ако је поље на позицији (0, 0) бело, тада треба префарбати само сва остала поља (њих $R \cdot C - m$), а ако је поље на позицији (0, 0) црно, тада треба офарбати само сва пребројана поља (њих m). Решење је, дакле, мањи од бројева $R \cdot C - m$ и m .

Пребројавање можемо вршити директно приликом учитавања карактера и нема потребе да се подаци о распореду поља памте (нпр. у матрици). Учитавање можемо вршити карактер по карактер, помоћу две угнеђене петље. Након учитаних k карактера сваког реда, потребно је још прочитати и карактер који обележава прелазак у нови ред и занемарити га. То се најједноставније постиже коришћењем `cin >> ws` (`ws` служи да се пресоче све белине тј. `whitespace`).

```
// U varijanti kada je polje (0, 0) belo, OK su belja polja na
// parnim zbirovima koordinata i crna polja na neparnim zbirovima
// kordinata. Prebrojavamo takva polja.
int m = 0;
int r, k;
cin >> r >> k;
// citamo r redova
for (int i = 0; i < r; i++) {
    // citamo C karaktera tekuceg reda
    for (int j = 0; j < k; j++) {
        char c;
```

3.3. УГНЕЖЂЕНЕ ПЕТЉЕ

```
cin >> c;
if (c == 'B' && (i+j) % 2 == 0) m++;
if (c == 'C' && (i+j) % 2 == 1) m++;
}
// preskacemo oznaku za prelazak u ovi red
cin >> ws;
}
// U varijanti kada je polje (0, 0) belo, treba prefarbati polja
// koja nisu prebrojana (njih R*C - m), a u varijanti kada je polje
// (0, 0) crno, treba prefarbati polja koja jesu prebrojana (njih
// m). Biramo varijantu koja je povoljnija.
cout << min(r * k - m, m) << endl;
```

Задатак: Серије 123

За дато n исписати чланове низа 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, ..., n , ..., n (низ садржи једну јединицу, две двојке, три тројке итд.).

Улаз: Са стандардног улаза се учитава природан број n ($1 \leq n \leq 30$).

Излаз: Чланове траженог низа исписати на стандардни излаз (иза сваког исписати један размак).

Пример

Улаз	Излаз
4	1 2 2 3 3 3 4 4 4 4

Задатак: Таблица множења

Напиши програм који исписује таблицу множења.

Улаз: Са стандардног улаза се уносе два цела броја m и n ($1 \leq m, n \leq 9$), сваки у посебном реду.

Излаз: На стандардни излаз исписати таблицу множења са m врста и n колона, како је приказано у примеру. Између колона штампати табулатор (карактер Tab).

Пример

Улаз	Излаз
5	1 * 1 = 1 1 * 2 = 2 1 * 3 = 3 1 * 4 = 4 1 * 5 = 5
5	2 * 1 = 2 2 * 2 = 4 2 * 3 = 6 2 * 4 = 8 2 * 5 = 10
	3 * 1 = 3 3 * 2 = 6 3 * 3 = 9 3 * 4 = 12 3 * 5 = 15
	4 * 1 = 4 4 * 2 = 8 4 * 3 = 12 4 * 4 = 16 4 * 5 = 20
	5 * 1 = 5 5 * 2 = 10 5 * 3 = 15 5 * 4 = 20 5 * 5 = 25

Задатак: Серије непарни парни

За дато n исписати елементе низа 1, 2, 4, 5, 7, 9, 10, 12, 14, 16, 17, ... који се формира тако што се полазећи од броја 1 – приказује један непаран природни број, затим следећа два парна – 2, 4; па следећа 3 непарна – 5, 7, 9; следећа 4 парна – 10, 12, 14, 16, ... итд. Последња серија садржи n елемената.

Улаз: Са стандардног улаза се учитава природан број n у границама од 1 до 40.

Излаз: На стандардни излаз исписати елементе траженог низа, сваку подсерију у посебном реду, при чему се иза сваког броја исписује један размак.

Пример

Улаз	Излаз
5	1
	2 4
	5 7 9
	10 12 14 16
	17 19 21 23 25

Глава 4

Детаљнији преглед основних типова података

4.1 Рад са целим бројевима

У овом поглављу ћемо детаљније изучити различите целобројне типове.

4.1.1 Распон целобројних типова

Многи програмски језици разликују више типова података чији је скуп вредности неки подскуп скупа целих бројева. Ти типови се разликују по меморијском простору одвојеном за запис сваког броја (а отуд и по распону вредности које се могу исправно представити типом). Ово значи да се пажљивим избором одговарајућег целобројног типа (најмањег типа чији је распон довољан да се представе све вредности које променљива може узети током извршавања програма), програм може да заузме мање меморије и брже приступа подацима. Обратно, некад је потребно рачунати са веома великим бројевима. Тада треба водити рачуна о томе да уобичајени, најчешће коришћени целобројни тип податка (тип `int`) може да буде недовољан за смештање таквих података. У таквој ситуацији важно је да се употреби неки тип чији је распон довољно широк да омогући представљање довољно великих целих бројева. Необраћање пажње на ове ситуације доводи до покушаја да се сувише велики позитиван (или сувише мали негативан) број смести у недовољно велики меморијски простор (тј. да покуша да се запише помоћу мање цифара него што је то потребно). Такав број ће бити неисправно записан у предвиђеном простору и таква ситуација се назива **прекорачење** (или у случају веома малог броја **подкорачење**). Проблем неће бити сигнализирани ни на који начин, програм наставља да ради као да је све у реду, али при читању неисправно записаног броја се добија погрешна вредност, што доводи до неочекиваног резултата рада читавог програма.

У неким ситуацијама знамо да ће све вредности које нека променљива моћи да има бити ненегативне, док некада знамо да могу бити и негативне. **Неозначени** (енгл. `unsigned`) типови података су они који омогућавају само запис ненегативних вредности, док **означени** (енгл. `signed`) типови података омогућавају запис и позитивних и негативних вредности.

Иако смо за представљање целих бројева до сада користили само тип `int`, у језику C++ постоје следећи целобројни типови.

тип	распон
<code>short</code>	$[-2^{15}, 2^{15} - 1] = [-32768, 32767]$
<code>unsigned short</code>	$[0, 2^{16} - 1] = [0, 65535]$
<code>int</code>	$[-2^{31}, 2^{31} - 1] \approx [-2.1 \cdot 10^9, 2.1 \cdot 10^9]$
<code>unsigned int</code>	$[0, 2^{32} - 1] \approx [0, 4, 2 \cdot 10^9]$
<code>long long</code>	$[-2^{63}, 2^{63} - 1] \approx [-9 \cdot 10^{18}, 9 \cdot 10^{18}]$
<code>unsigned long long</code>	$[0, 2^{64} - 1] \approx [0, 1.8 \cdot 10^{19}]$

Најмању и највећу дозвољену вредност која се може представити неким типом можемо добити функ-

4.1. РАД СА ЦЕЛИМ БРОЈЕВИМА

цијама `numeric_limits<type>::min()` и `numeric_limits<type>::max()` декларисаним у заглављу `<limits>` (нпр. за тип `int` те вредности можемо прочитати позивима `numeric_limits<int>::min()` и `numeric_limits<int>::max()`) или читањем одговарајућих вредности из заглавља `<limits>` (нпр. за тип `int` то су вредности `INT_MAX` и `INT_MIN`).

Избор типа обично зависи од ограничења задатих у тексту задатка. Ако се применом неке аритметичке операције добије резултат који се не може представити одабраним типом података кажемо да долази до *прекорачења*. На пример, ако сабирамо две и две милијарде представљене променљивама типа `int` резултат четири милијарде је превелики да би се могао представити типом `int` и уместо њега, као резултат се добија неки негативан број.

У наредних неколико задатака биће потребно обратити пажњу на одабир целобројних типова и грешке прекорачења.

Задатак: Множење великих бројева

Жути мрави желе да одреде површину ливаде по којој се крећу. Пошто су они јако мали, дужина и ширина ливаде су изражени у микрометрима.

Улаз: Са стандардног улаза се уносе два цела броја a и b ($0 \leq a, b \leq 4 \cdot 10^9$) који представљају димензије ливаде у микрометрима.

Излаз: На стандардни излаз испиши површину ливаде у микрометрима квадратним.

Пример 1

Улаз	Излаз
1234567890	3971049186085674300
3216549870	

Пример 2

Улаз	Излаз
4000000000	16000000000000000000
4000000000	

Решење

Јасно је да је алгоритам који треба применити у овом задатку тривијалан - потребно је помножити два цела броја, јер је површина правоугаоника једнака $P = a \cdot b$. Цела проблематика у овом задатку потиче од тога што су бројеви који се множе веома велики (горње ограничење улазних параметара је $4 \cdot 10^9$). За њихово репрезентовање потребно је користити 32-битни, неозначени запис бројева (јер је $2^{31} < 4 \cdot 10^9 < 2^{32}$). У језику C++ могуће је употребити тип `unsigned int`. Максимална вредност њиховог производа је $16 \cdot 10^{18}$, а њу је могуће исправно репрезентовати само помоћу неозначених 64-битних бројева. У језику C++ могуће је употребити тип `unsigned long long`. Означени 64-битни бројеви достижу само $2^{63} - 1$, што је мање од $16 \cdot 10^{18}$. Једноставности ради, могуће је и улазне податке представити 64-битним неозначеним бројевима.

```
unsigned long long a, b;  
cin >> a >> b;  
cout << a * b << endl;
```

Задатак: Јабуке у троуглу

Јабуке су послане у троугао следећег облика:

```
*  
**  
***  
****
```

Напиши програм који одређује укупан број јабука у троуглу.

Улаз: Са стандардног улаза уноси се цео број n ($3 \cdot 10^9 \leq n \leq 6 \cdot 10^9$) који одређује број редова троугла.

Излаз: На стандардни излаз исписати један цео број који представља укупан број јабука у троуглу.

Пример 1

Улаз	Излаз
3500000000	6125000001750000000

Пример 2

Улаз	Излаз
5000000000	12500000002500000000

Решење

Збир јабука у троуглу једнак је збиру првих n природних бројева, што је на основу Гаусове формуле приказане у задатку $\frac{n(n+1)}{2}$. На основу граница за број n јасно је да у решењу задатка није могуће користити

променљиве типа `int` ни `unsigned int` нити за број n нити за тражени укупан број. Ако се употреби тип `unsigned long long`, тада је допуштени распон бројева бар од 0 до око $1.84 \cdot 10^{19}$. Пошто n може највише бити око $6 \cdot 10^9$, тада $\frac{n(n+1)}{2}$ може највише бити што је око $1.80 \cdot 10^{19}$ што се уклапа у ограничење типа. Међутим, приликом извођења аритметичких операција треба бити обазрив и не сме се догодити да при израчунавања међурезултата дође до неког прекорачења. То би се заправо десило ако би се прво помножили бројеви n и $n + 1$ (резултат тог множења могао би да буде $3.6 \cdot 10^{19}$, што прекорачује опсег). Зато је потребно прво извршити дељења са 2, а тек онда множење. Остаје питање који од бројева n и $n + 1$ делити са 2. Пошто се ради са целим бројевима дељење је целобројно и да би се добио исправан резултат, потребно је поделити онај број који је паран (од два узастопна природна броја, увек је један паран).

```
unsigned long long n;
cin >> n;
cout << (n%2 == 0 ? (n/2)*(n+1) : n*((n+1)/2)) << endl;
```

Задатак: Средина интервала

На планети М у улици М живе мрави. Све кућице у улици су означене природним бројевима. Мравица Ана живи у кућици број a , а мрав Борис иза ње, у кућици број b . Мрав Виктор живи у кућици тачно на средини између Ане и Бориса. Напиши програм који одређује број кућице у којој живи Виктор (ако је између Ане и Бориса паран број кућица, Виктор живи у оној која је ближа Ани).

Улаз: Са стандардног улаза уносе се два цела броја:

- a ($0 \leq a \leq 2^{64} - 3$) - број кућице у којој живи Ана
- b ($a + 2 \leq b \leq 2^{64} - 1$) - број кућице у којој живи Борис

Излаз: На стандардни излаз исписати један цео број - број кућице у којој живи Виктор.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
432457	497389	2	5
562321		9	

Пример 3	
Улаз	Излаз
9999999999999999999	9999999999999999999
10000000000000000000	

Решење

Између Ане и Бориса постоји тачно $b - a - 1$ кућица. На пример, ако Ана живи у кућици број 3, а Борис у кућици број 9, онда између њих постоји 5 кућица (то су кућице број 4, 5, 6, 7 и 8).

Ако је то непаран број, онда је Викторова кућица тачно између Анине и Борисове. Када се Викторова кућица изузме, од преосталих кућица између Анине и Борисове кућице, пола њих је ближе Аниној, а пола ближе Борисовој (у претходном примеру, пет кућица је између, у средини је Викторова, док су кућице број 4 и 5 ближе Аниној, а кућице број 7 и 8 ближе Борисовој). Број Викторове кућице добијамо тако што на број Анине кућице a додамо пола кућица које су ближе Аниној кућици тј. $\frac{b-a-2}{2}$ кућица и на крају Викторову кућицу. Дакле, број Викторове кућице једнак је $a + \frac{b-a-2}{2} + 1 = a + \frac{b-a}{2}$ (у примеру добијамо $3 + \frac{9-3}{2} = 6$, што је управо број Викторове кућице).

Ако је број кућица између Ане и Бориса паран, онда је пола њих ближе Аниној, а пола њих ближе Борисовој кућици, а Виктор живи у последњој кућици у низу кућица које су ближе Аниној. Ако Ана, на пример, живи у кућици број 3, а Борис у кућици број 8, тада је између њих четири кућице (број 4, 5, 6 и 7). Број Викторове кућице добијамо тако што на број Анине кућице a додамо половину броја кућица између Анине и Борисове, а то је $\frac{b-a-1}{2}$. Дакле, број Викторове кућице једнак је $a + \frac{b-a-1}{2}$ (у примеру добијамо $3 + \frac{8-3-1}{2} = 5$, што је управо број Викторове кућице).

С обзиром на ограничења дата у тексту задатка, потребно је користити тип података за запис 64-битних целих бројева.

```
unsigned long long a, b;
cin >> a >> b;
unsigned long long v;
if ((b - a - 1) % 2 != 0)
```

4.1. РАД СА ЦЕЛИМ БРОЈЕВИМА

```
v = a + (b - a) / 2;  
else  
v = a + (b - a - 1) / 2;  
cout << v << endl;
```

Аналізу случајева можемо избећи ако приметимо да важи да је и у првом и у другом случају решење једнако $a + \lfloor \frac{b-a}{2} \rfloor$ тј. $a + (b-a) \operatorname{div} 2$. Наиме, ако је $b-a-1$ непарно, тада је $b-a$ парно тј. дељиво са 2, па важи да је $\frac{b-a}{2}$ цео број и да је $\lfloor \frac{b-a}{2} \rfloor = \frac{b-a}{2}$. Ако је $b-a-1$ парно, тада је $\lfloor \frac{b-a}{2} \rfloor = \frac{b-a-1}{2}$. Наиме, $\frac{b-a-1}{2}$ је тада цео број, а тривијално важи да је $b-a-1 < b-a < b-a+1$, одакле следи да је $\frac{b-a-1}{2} \leq \frac{b-a}{2} < \frac{b-a-1}{2} + 1$, што је потребно и довољно да се закључи да је $\lfloor \frac{b-a}{2} \rfloor = \frac{b-a-1}{2}$. Пошто је $\lfloor \frac{b-a}{2} \rfloor$ целобројни количник бројева $b-a$ и 2, тј. $(b-a) \operatorname{div} 2$ (види задатак [Поклони](#)), решење је могуће имплементирати веома једноставно и без анализе случајева.

```
unsigned long long a, b;  
cin >> a >> b;  
unsigned long long v = a + (b - a) / 2;  
cout << v << endl;
```

Размотримо и наредно решење. Ако замислимо да су све кућице ширине 1 и да су распоређене дуж реалне праве тако да се средина сваке кућице налази код њеног целог броја, тада се тачка на средини између Анине и Борисове кућице може израчунати као $a + \frac{b-a}{2} = \frac{a+b}{2}$. Ако је то цео број, онда је то баш број Викторове кућице, а ако није, онда је број Викторове кућице за 0.5 мањи од тог броја. У оба случаја је број Викторове кућице $a + \lfloor \frac{b-a}{2} \rfloor = \lfloor \frac{a+b}{2} \rfloor$. Прво решење $a + \lfloor \frac{b-a}{2} \rfloor$ се поклапа са оним које смо добили у полазном разматрању, док је и друго решење $\lfloor \frac{a+b}{2} \rfloor$ такво да омогућава директну имплементацију (коришћењем целобројног количника $(a+b) \operatorname{div} 2$). Напоменимо да, међутим, то решење може да доведе до нетачних резултата, јер приликом сабирања бројева a и b може да дође до прекорачења (иако су и a и b и резултат унутар опсега коришћеног типа података, међурезултат $a+b$ не мора да буде) и стога га је пожељно избегавати.

```
unsigned long long a, b;  
cin >> a >> b;  
unsigned long long v = (a + b) / 2;  
cout << v << endl;
```

Задатак: Провера прекорачења

Напиши програм који одређује да ли ће доћи до прекорачења или поткорачења приликом сабирања и одузимања два дата броја типа `int`.

Улаз: Са стандардног улаза се читавају два броја, сваки у посебној линији. Сваки број се исправно може репрезентовати типом `int` тј. `Integer` на рачунару на којем се врши тестирање.

Излаз: У првој линији стандардног излаза исписати да ако се та два броја могу исправно сабрати тј. не ако приликом сабирања долази до прекорачења или поткорачења. У другој линији стандардног излаза исписати да ако се та два броја могу исправно одузети тј. не ако приликом одузимања долази до прекорачења или поткорачења.

У наредним примерима се претпоставља да се за запис типа `int` користе 32 бита (то не мора увек бити тако).

Пример 1		Пример 2		Пример 3	
Улаз	Излаз	Улаз	Излаз	Улаз	Излаз
1000000000	da	2000000000	ne	2000000000	da
1000000000	da	2000000000	da	-2000000000	ne

Задатак: Факторијел 64бит

Напиши програм који израчунава вредности факторијела датог целог броја.

Улаз: Са стандардног улаза уноси се број n ($1 \leq n \leq 20$).

Излаз: На стандардни излаз исписује се један број $n!$.

Пример

```
Улаз      Излаз  
20        2432902008176640000
```

4.1.2 Модуларна аритметика

Извршавање аритметичких операција по модулу n подразумева да се након примене операције одреди остатак при дељењу са бројем n . Важе следеће релације:

$$\begin{aligned}(a + b) \bmod n &= (a \bmod n + b \bmod n) \bmod n \\(a \cdot b) \bmod n &= (a \bmod n \cdot b \bmod n) \bmod n \\(b - a) \bmod n &= (b \bmod n - a \bmod n + n) \bmod n\end{aligned}$$

Докажимо релацију о производу (она о збиру се доказује још једноставније). Подсетимо се да је $x \bmod y = r$ ако и само ако постоји q такав да је $x = q \cdot y + r$ и ако је $0 \leq r < y$. Претпоставимо да је $a = q_a \cdot n + r_a$ и $b = q_b \cdot n + r_b$ за $0 \leq r_a, r_b < n$. Тада важи да је $a \cdot b = (q_a \cdot n + r_a) \cdot (q_b \cdot n + r_b) = (q_a \cdot q_b \cdot n + q_a \cdot r_b + r_a \cdot q_b) \cdot n + r_a \cdot r_b$. Ако важи да је $r_a \cdot r_b = q \cdot n + r$ за $0 \leq r < n$, тада је $a \cdot b = (q_a \cdot q_b \cdot n + q_a \cdot r_b + r_a \cdot q_b + q) \cdot n + r$, па је $(a \cdot b) \bmod n = r$. Важи да је $(a \bmod n \cdot b \bmod n) \bmod n = (r_a \cdot r_b) \bmod n = r$, чиме је тврђење доказано.

Докажимо релацију о разлици. Додавање броја n служи да се избегне дељење негативних бројева. Подсетимо се да је $x \bmod y = r$ ако и само ако постоји q такав да је $x = q \cdot y + r$ и ако је $0 \leq r < y$. Нека је $a = q_a \cdot n + r_a$ и $b = q_b \cdot n + r_b$, за $0 \leq r_a, r_b < n$. Зато је $a \bmod n = r_a$ и $b \bmod n = r_b$. Нека је $r_b - r_a + n = q \cdot n + r$ за неко $0 \leq r < n$. Зато је $(a \bmod n - b \bmod n + n) \bmod n = (r_b - r_a + n) \bmod n = r$. Такође, важи и да је $b - a = (q_b - q_a) \cdot n + (r_b - r_a) = (q_b - q_a - 1) \cdot n + (r_b - r_a + n) = (q_b - q_a - 1 + q) \cdot n + r$, па је и $(b - a) \bmod n = r$, чиме је тврђење доказано.

У наставку ћемо приказати неколико задатака у којима се примењује модуларна аритметика.

Задатак: Операције по модулу

Напиши програм који одређује последње три цифре збира и последње три цифре производа четири унета цела броја.

Улаз: У сваком реду стандардног улаза уноси се по један цео број из интервала $[1..999]$.

Излаз: На стандардни излаз се исписују број одређен са последње три цифре збира и број одређен са последње три цифре производа унетих бројева (евентуалне водеће нуле се не морају исписати).

Пример

Улаз	Излаз
999	996
999	1
999	
999	

Решење

Идеја која природно прва падне напамет је да се израчунају збир и производ унетих бројева, а онда да се последње три цифре одреде израчунавањем остатка при целобројном дељењу са 1000. Када се у обзир узме распон бројева који се уносе, такво решење би давало коректне резултате у случају збира, међутим, производ бројева може бити много већи од самих бројева и може прекорачити распон типа `int` који користимо за репрезентовање целих бројева и зато резултат може бити погрешан.

Зато је за израчунавање последње три цифре производа потребно применити мало напреднији алгоритам који користи чињеницу да су за последње три цифре производа релевантне само последње три цифре сваког од чинилаца. Зато је пре множења могуће одредити остатак при дељењу са 1000 сваког од чинилаца, израчунати производ, а онда одредити његове три последње цифре израчунавањем остатка његовог при дељењу са 1000. Аналогну тактику могуће је применити и на израчунавање збира.

Функције за множење и сабирање по модулу засноваћемо на релацијама $(a + b) \bmod n = (a \bmod n + b \bmod n) \bmod n$ и $(a \cdot b) \bmod n = (a \bmod n \cdot b \bmod n) \bmod n$, а онда ћемо их примењивати тако што ћемо производ (тј. збир) по модулу n свих претходних бројева применом функција комбиновати са новим бројем. Дакле, ако функцију за сабирање по модулу n означимо са $+_n$, а за множење по модулу n са \cdot_n , израчунаваћемо $((a +_n b) +_n c) +_n d$ тј. $((a \cdot_n b) \cdot_n c) \cdot_n d$.

Нагласимо да је израчунавање целобројног количника и остатка временски захтевна операција (најчешће се извршава доста спорије него основне аритметичке операције), тако да је у пракси пожељно избећи је када је

4.1. РАД СА ЦЕЛИМ БРОЈЕВИМА

то могуће. На пример, ако сте потпуно сигурни да ће $a + b$ бити могуће репрезентовати одабраним типом података, тада је уместо $(a \bmod n + b \bmod n) \bmod n$ ипак боље користити $(a + b) \bmod n$.

```
int plus_mod(int a, int b, int n) {
    return ((a % n) + (b % n)) % n;
}

int puta_mod(int a, int b, int n) {
    return ((a % n) * (b % n)) % n;
}

int main() {
    int a, b, c, d;
    cin >> a >> b >> c >> d;

    cout << plus_mod(plus_mod(plus_mod(a, b, 1000), c, 1000), d, 1000)
          << endl;
    cout << puta_mod(puta_mod(puta_mod(a, b, 1000), c, 1000), d, 1000)
          << endl;
    return 0;
}
```

Уместо у једном изразу, збир и производ можемо рачунати итеративно тако што их иницијализујемо на нулу тј. јединицу, а затим у сваком кораку текући збир мењамо збиром по модулу текућег збира и текућег броја, а текући производ мењамо производом по модулу текућег производа и текућег броја.

Задатак: Монопол

Монопол је игра у којој се играчи крећу по пољима која су постављена у круг. Играчи се увек крећу у смеру казаљке на сату. Претпоставимо да су поља означена редним бројевима који крећу од 0, да су на почетку игре оба играча на том пољу и да се током игре играчи нису кретали унатраг. Ако се зна број поља које је први играч прешао од почетка игре и број поља које је други играч прешао од почетка игре напиши програм који израчунава колико корака први играч треба да направи да би дошао на поље на ком се налази други играч.

Улаз: Са стандардног улаза се уносе три природна броја. У првој линији дат је број поља на табли, у другој број поља које је од почетка игре прешао први, а у трећој број поља које је од почетка игре прешао други играч.

Излаз: На стандардни излаз треба исписати колико корака унапред играч треба да направи да би стигао на жељено поље.

Пример 1

Улаз	Излаз
10	4
3	
7	

Пример 2

Улаз	Излаз
10	6
7	
3	

Решење

Означимо укупан број поља са n , број поља које је први играч прешао са A и број поља које је други играч прешао са B . Означимо број поља на коме се први играч тренутно налази са a , а број поља на које треба да стигне са b . Током свог кретања он је прешао k_a пуних кругова ($k_a \geq 0$) у којима је прешао по n поља и након тога још a поља, тако да је $A = k_a \cdot n + a$. Дакле, пошто је $0 \leq a < n$, на основу дефиниције целобројног дељења важи да је $a = A \bmod n$. Слично је и $b = B \bmod n$.

Ако је $b \geq a$ тада се број корака може израчунати као $b - a$. Међутим, могуће је и да важи $b < a$ и у том случају играч мора прећи преко поља Start које је означено бројем 0. Број корака које играч треба да направи да би од поља на ком се налази стигао до поља Start је $n - a$, а број корака потребних да од поља Start стигне до жељеног поља је b , тако да је укупан број корака једнак $n - a + b$. На основу ове дискусије једноставно је направити програм који анализом ова два случаја стиже до решења.

```
int brojPolja;
int presaoPrvi, presaoDrugi;
```



```
cin >> brojPolja >> presaoPrvi >> presaoDrugi;
int saPolja = presaoPrvi % brojPolja;
int naPolje = presaoDrugi % brojPolja;
cout << (naPolje >= saPolja ?
        naPolje - saPolja :
        naPolje - saPolja + brojPolja)
    << endl;
```

Још једно гледиште које доводи до истог решења је могућност да се у случају да је $b < a$ промене ознаке на пољима иза старта. Тако би се поље Start уместо са 0 означило са n , следеће би се уместо са 1 означило са $1 + n$ итд., док би се поље b означило са $b + n$. Дакле, решење добијамо тако што умањилац увећавамо за n уколико је мањи од умањеника, и израчунавамо разлику.

```
int brojPolja;
int presaoPrvi, presaoDrugi;
cin >> brojPolja >> presaoPrvi >> presaoDrugi;
int saPolja = presaoPrvi % brojPolja;
int naPolje = presaoDrugi % brojPolja;
if (naPolje < saPolja)
    naPolje += brojPolja;
cout << naPolje - saPolja << endl;
```

Посматрано још из једног угла у овом задатку се тражи да се одреди разлика бројева B и A по модулу n . То сугерише да је уместо анализе случајева решење могуће добити израчунавањем вредности израза $(b - a + n) \bmod n$ тј. $(B \bmod n - A \bmod n + n) \bmod n$. Заиста, ако је $b \geq a$, вредност израза $b - a + n$ биће већа или једнака n и њен остатак при дељењу са n биће једнак вредности $b - a$ (тражење остатка ће практично поништити првобитно додавање вредности n). Са друге стране, ако је $b < a$ тада ће $b - a$ бити негативан број, па ће се додавањем вредности n добити број из интервала између 0 и $n - 1$. Зато проналажење остатка неће на крају променити резултат и добиће се вредност $b - a + n$ за коју смо рекли да је тражена вредност у овом случају.

```
int brojPolja;
int presaoPrvi, presaoDrugi;
cin >> brojPolja >> presaoPrvi >> presaoDrugi;
int saPolja = presaoPrvi % brojPolja;
int naPolje = presaoDrugi % brojPolja;
cout << (naPolje - saPolja + brojPolja) % brojPolja << endl;
```

Задатак: Сат

Претпоставимо да приликом исказивања времена за сате користимо само бројеве од 1 до 12. Тако за 17:45 кажемо 5:45, а за 0:32 кажемо да је 12:32. Ако је познат број сати и минута такав да је број сати у опсегу од 0 до 23, а минута од 0 до 59, исказати то време тако да је број сати у опсегу 1 до 12.

Улаз: Учитавају се два броја, сваки у посебној линији. У првој линији је број сати (између 0 и 23), а у другој линији је број минута (између 0 и 59).

Излаз: Исписује се једна линија у којој се приказује временски тренутак у формату h:m, где је број сати h између 1 и 12, а број минута између 0 и 59.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
0	12:32	5	5:35
32		35	

Решење

У задатку је кључно одредити пресликавање сати, док се минути исписују непромењено. Пошто бројање треба да буде од 1 до 12, потребно је да реализујемо следеће пресликавање:

0	1	2	...	11	12	13	14	15	...	22	23	h
12	1	2	...	11	12	1	2	3	...	10	11	$f(h)$

4.1. РАД СА ЦЕЛИМ БРОЈЕВИМА

Један од начина да се то уради је да се употреби гранање и да се вредност 0 преслика у вредност 12, да се вредности између 1 и 12 пресликају саме у себе, а да се од вредности веће од 12 пресликају одузимањем броја 12 од њих.

```
int h, m; // broj sati i minuta
cin >> h >> m;
// izracunavanje i ispis rezultata
if (h == 0)
    h = 12;
else if (h > 12)
    h = h - 12;
cout << h << ":" << m << endl;
```

Сати се периодично понављају на сваких 12 сати. Ако бисмо сате бројали од 0 до 11, тада би се одговарајуће пресликавање могло одредити једноставним проналажењем остатка при дељењу са 12.

0	1	2	...	11	12	13	14	15	...	22	23	h
0	1	2	...	11	0	1	2	3	...	10	11	$h \bmod 12$

Приметимо да након израчунавања остатка при целобројном дељењу са 12 проблем праве једино вредности 0 и 12 које су се пресликале у 0 уместо у 12.

Тај случај је могуће посебно испитати и поправити резултат.

```
int h, m; // broj sati i minuta
cin >> h >> m;
// izracunavanje i ispis rezultata
h = h % 12;
if (h == 0) h = 12;
cout << h << ":" << m << endl;
```

Ипак, циљ нам је да покажемо како је могуће решити овај задатак и без употребе гранања. Да бисмо добили вредност из интервала од 1 до 12, можемо на добијени остатак који је из интервала 0 до 11 додати 1. Међутим, да би пресликавање било коректно, потребно је ово увећање неутрализовати тако што ћемо пре одређивања остатка од полазног броја одузети 1. Дакле, број сати можемо одредити као $(h - 1) \bmod 12 + 1$. На пример, ако од 12 одузмемо 1, одредимо остатак добијеног броја 11 при дељењу са 12 и на добијени број 11 додамо 1 добићемо исправну вредност 12. Слично, ако од 13 одузмемо 1, одредимо остатак добијеног броја 12 са 12 и на добијени број 0 додамо 1 добићемо исправан резултат 1. Дакле, решење се реализује кроз следећа пресликавања

0	1	2	...	11	12	13	14	15	...	22	23	h
-1	0	1	...	10	11	12	13	14	...	21	22	$h - 1$
11	0	1	...	10	11	0	1	2	...	9	10	$(h - 1) \bmod 12$
12	1	2	...	11	12	1	2	3	...	10	11	$(h - 1) \bmod 12 + 1$

Једини број који може да прави проблем је 0 јер се почетним одузимањем броја 1 добија негативни број -1, а не желимо да се ослањамо на понашање остатка у случају негативног делиоца. То можемо поправити тако што на полазни број увек на почетку додамо 12 чиме обезбеђујемо да ће број увек бити позитиван, при чему се остатак при дељењу са 12 не мења. Дакле, тражено пресликавање је $(h + 12 - 1) \bmod 12 + 1$ или $(h + 11) \bmod 12 + 1$. Нагласимо да се ова техника додавања вредности модула користи када год је потребно одузети две вредности, по неком модулу. Наиме, подсетимо се да важи

$$(a - b) \bmod n = (a \bmod n - b \bmod n + n) \bmod n.$$

```
int h, m; // broj sati i minuta
cin >> h >> m;
// izracunavanje i ispis rezultata
cout << (h + 11) % 12 + 1 << ":" << m << endl;
```

Задатак: Збир бројева по модулу

Напиши програм који израчунава збир природних бројева од 1 до n по датом модулу m .

Улаз: Са стандардног улаза се уносе два цела броја, сваки у посебном реду:

- n ($10^2 \leq n \leq 2^{32} - 1$)
- m ($2 \leq m \leq 100$)

Излаз: На стандардни излаз исписати један цео број који представља тражени збир по модулу.

Пример

<i>Улаз</i>	<i>Излаз</i>
100	50
100	

Објашњење

Збир свих бројева од 1 до 100 је 5050 и остатак при дељењу са 100 је 50.

Задатак: Разбрајалица

Деца стоје у кругу и одређују ко ће од њих да жмури тако што изговарају неку разбрајалицу (на пример, еци-пеци-пец). Ако децу обележимо бројевима од 0 до $n - 1$, редом како стоје у кругу, ако је познат број речи (слогова) разбрајалице и ако је познат редни број детета од којег почиње разбрајање, напиши програм који приказује редни број детета које је одабрано да жмури (тј. на коме се разбрајање завршава).

Улаз: Уносе се три природна броја. Број слогова разбрајалице, број деце и редни број детета од којег почиње разбрајање. Претпоставићемо да је број деце мањи од 100, а да је број слогова мањи од 1000.

Излаз: Исписује се један природни број – редни број детета одабраног да жмури.

Пример

<i>Улаз</i>	<i>Излаз</i>
13	1
7	
3	

Објашњење

Претпоставимо да седморо деце у кругу користи разбрајалицу еци-пеци-пец, а да бројање креће од детета број 3, разбрајање ће тећи овако:

еци-пеци-пец-јасам-мали-зец-тиси-мала-препе-лица-еци-пеци-пец
 3 4 5 6 0 1 2 3 4 5 6 0 1

Задатак: Жмурке

Дете броји у игри жмурке 5, 10, 15, ..., 100, али пошто не зна бројеве веће од 100, после 100 оно креће да броји из почетка. Који је број рекао у n -том кораку?

Улаз: Са стандардног улаза учитава се један позитиван цео број n .

Излаз: На стандардни излаз исписује се број који дете изговара у n -том кораку.

Пример 1

<i>Улаз</i>	<i>Излаз</i>
5	25

Пример 2

<i>Улаз</i>	<i>Излаз</i>
1234	70

Задатак: Трајање вожње кроз два дана

Познати су сат, минут и секунд почетка и краја вожње аутобусом (могуће је и да је вожња почела у једном, а завршила се у наредном дану, али се зна да је трајала мање од 24 сата). Написати програм који одређује колико сати, минута и секунди је трајала та вожња.

Улаз: Са стандардног улаза учитава се 6 бројева (сваки у засебном реду). Прво сат, минут и секунд почетка вожње, а затим сат, минут и секунд краја вожње.

4.1. РАД СА ЦЕЛИМ БРОЈЕВИМА

Изназ: На стандардни излаз се исписује један ред у коме су три броја (сат, минут и секунд) раздвојена запетом.

Пример 1

Улаз	Изназ
23 59 59	0:0:2
0 0 1	

Пример 2

Улаз	Изназ
0 0 1	23:59:58
23 59 59	

Задатак: Навијање сата

Дочепо се мали брата очевога сата. Пошто је још мали, зна да гледа само сатну казаљку. Сат се може навијати тако што се једним дугметом сатна казаљка помера за један сат унапред (у смеру кретања казаљке на сату) а другим дугметом се помера за један сат уназад (у супротном смеру кретања казаљке на сату). Ако се зна позиција a на којој се тренутно налази казаљка, позиција b на којој брата жели да се нађе, напиши програм који одређује да ли се сат може навити тако што брата притисне једно од два дугмета тачно k пута.

Улаз: Са стандардног улаза учитавају се бројеви $0 \leq a, b \leq 11$ и број $0 \leq k \leq 1000$.

Изназ: На стандардни излаз написати текст `pargd` ако се сат може навити тако што се казаљка помера унапред, `pazad` ако се сат може навити тако што се казаљка помера уназад (ако је сат могуће навити на оба начина, исписати `pargd`, а затим `pazad` у следећем реду) тј. `ne moze` ако се сат не може навити.

Пример 1

Улаз	Изназ
1	pargd
7	pazad
18	

Пример 2

Улаз	Изназ
2	pazad
4	
10	

Пример 3

Улаз	Изназ
7	ne moze
9	
3	

Задатак: Време завршетка филма

Овај задатак је иновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Аутобус

Ако је познато трајање путовања и време када аутобус треба да стигне на одредиште, напиши програм који израчунава када аутобус треба да пође са полазишта (полазак може бити у претходном дану у односу на долазак).

Улаз: Са стандардног улаза учитава се четири цела броја:

- $trajanje_h$ ($0 \leq trajanje_h < 12$) и $trajanje_m$ ($0 \leq trajanje_m < 60$) - сати и минути трајања вожње
- $dolazak_h$ ($0 \leq dolazak_h < 24$) и $dolazak_m$ ($0 \leq dolazak_m < 60$) - сат и минут доласка

Изназ: На стандардни излаз исписати два цела броја раздвојена двотачком:

- $polazak_h$ ($0 \leq polazak_h < 24$) и $polazak_m$ ($0 \leq polazak_m < 60$) - сат и минут поласка

Пример 1

Улаз	Изназ
2	10:30
30	
13	
0	

Пример 2

Улаз	Изназ
12	20:0
0	
8	
0	

Задатак: Точак среће

Точак среће окренемо за одређени угао, па га поново окренемо у истом смеру за неки други угао. Точак је такав да се у једном окрету не може окренути више од 20 пуних кругова. Написати програм којим се одређује најмањи угао за који треба точак још једном да окренемо у истом смеру да би се дошао у почетни положај.

Улаз: Са стандардног улаза учитава се 6 бројева (сваки у засебном реду). Прво сат минут и секунд првог угла, а затим сат, минут и секунд другог угла за који се точак окрене.

Излаз: На стандардни излаз се исписује један ред у коме су три броја (сат, минут и секунд) раздвојена празнином.

Пример

Улаз	Излаз
120	108 19 0
50	
20	
130	
50	
40	

4.2 Везе између целих и реалних бројева

4.2.1 Заокруживање реалних бројева

Три најчешће коришћена начина заокруживања реалног броја x су:

- $\lfloor x \rfloor$ – заокруживање наниже, тј. на највећи цео број који је мањи или једнак од x . У језику C++ ово заокруживање врши се функцијом `floor` из заглавља `<cmath>`.
- $\lceil x \rceil$ – заокруживање навише, тј. на најмањи цео број који је већи или једнак од x . У језику C++ ово заокруживање врши се функцијом `ceil` из заглавља `<cmath>`.
- заокруживање на цео број који је најближи броју x . У језику C++ ово заокруживање врши се функцијом `round` из заглавља `<cmath>`. У случају када је реални број тачно између два цела броја (нпр. 3.5), различити програмски језици користе различита правила о заокруживању. У језику C++ подразумевано је правило да се број заокружује навише у случају позитивних тј. наниже у случају негативних бројева који су тачно између два цела броја (заокруживање се врши на број даљи од нуле).

Важе и следећа тврђења.

- Најмањи цео број строго већи од броја x једнак $\lfloor x \rfloor + 1$.
- Највећи цео број строго мањи од броја x једнак $\lceil x \rceil - 1$.

Докажимо прво тврђење и формално. Ако је $y = \lfloor x \rfloor$, тада је $y \leq x < y + 1$. Зато је $y + 1 = \lfloor x \rfloor + 1$ најмањи цео број који је строго већи од x (y , који је први број мањи од њега, није такав). Друго тврђење се доказује веома слично.

Заокруживање користимо да бисмо нашли целобројна решења неких неједначина. Ако су n и k позитивни бројеви, тада важе следећа тврђења.

- Највећи цео број x такав да је $x \cdot k \leq n$ једнак је $x = \lfloor \frac{n}{k} \rfloor$.
- Најмањи цео број x такав да је $x \cdot k \geq n$ једнак је $x = \lceil \frac{n}{k} \rceil$.
- Најмањи цео број x такав да је $x \cdot k > n$ једнак је $x = \lfloor \frac{n}{k} \rfloor + 1$
- Највећи цео број x такав да је $x \cdot k < n$ једнак је $x = \lceil \frac{n}{k} \rceil - 1$.

Докажимо формално прво тврђење. По дефиницији важи да је $\lfloor a \rfloor = b$ ако је $b \leq a < b + 1$. Докажимо да је $\lfloor \frac{n}{k} \rfloor$ највећи број цео такав да помножен бројем k даје вредност мању или једнаку од n . Ако је $x = \lfloor \frac{n}{k} \rfloor$, тада важи да је $x \leq \frac{n}{k} < x + 1$. Зато је $x \cdot k \leq n < (x + 1) \cdot k$ и $x = \lfloor \frac{n}{k} \rfloor$ је заиста највећи број такав да је $x \cdot k \leq n$.

Докажимо формално и друго тврђење. По дефиницији важи да је $\lceil a \rceil = b$ ако је $b - 1 < a \leq b$. Зато, ако је $x = \lceil \frac{n}{k} \rceil$ тада је $x - 1 < \frac{n}{k} \leq x$. Зато је $(x - 1) \cdot k < n \leq x \cdot k$, па је $x = \lceil \frac{n}{k} \rceil$ заиста најмањи цео број такав да важи $x \cdot k \geq n$.

И остала тврђења се доказују на сличан начин.

Касније ћемо показати да ако су n и k природни бројеви, тада се заокруживање њиховог количника може урадити и само помоћу целобројне аритметике (види задатке [Поклони](#) и [Лифт](#)).

4.2. ВЕЗЕ ИЗМЕЂУ ЦЕЛИХ И РЕАЛНИХ БРОЈЕВА

Задатак: Пертла

Од канапа дате дужине исецају се пертле дате дужине. Одредити колико пертли је могуће изрезати и колико канапа на крају преостане.

Улаз: Са стандардног улаза учитавају се два позитивна реална броја. У првом реду дата је дужина канапа, а у другом дата је дужина пертле. Бројеви су такви да одређена количина канапа увек преостане.

Изназ: На стандардни излаз исписати цео број који представља број пертли које је могуће исећи и у следећем реду реалан број који представља дужину преосталог канапа.

Пример

Улаз	Изназ
50.5	5
10	0.5

Решење

Нека је дужина канапа k , а пертле p . Број пертли x је највећи број такав да је $x \cdot p \leq k$. То је број $x = \left\lfloor \frac{k}{p} \right\rfloor$, где заграде $\lfloor \dots \rfloor$ означавају заокруживање наниже тј. највећи цео број који није већи од датог броја. Дакле, број пертли се одређује као цео део количника дужине канапа и пертле. Након тога се израчуна укупна дужина канапа потребна да се тај број пертли направи, а дужина преосталог канапа се израчунава тако што се та дужина одузме од укупне дужине канапа. На пример, ако је дужина канапа 50.5 cm, а дужина пертле 10.0 cm тада се може направити $\left\lfloor \frac{50.5 \text{ cm}}{10 \text{ cm}} \right\rfloor = \lfloor 5.05 \rfloor = 5$ пертли, за то је потребно $5 \cdot 10.0 \text{ cm} = 50.0 \text{ cm}$ канапа. Зато преостаје $50.5 \text{ cm} - 50.0 \text{ cm} = 0.5 \text{ cm}$. Дакле, број пертли се израчунава као $\left\lfloor \frac{k}{p} \right\rfloor$, а преостала дужина канапа као $k - \left\lfloor \frac{k}{p} \right\rfloor \cdot p$.

Најбољи начин да се одреди цео део количника $\left\lfloor \frac{a}{b} \right\rfloor$ два реална броја a и b је да се изврши обично дељење те две реалне вредности, а онда да се употреби библиотечка функција за заокруживање реалног броја наниже. У језику C++ то је функција `floor`, декларисана у заглављу `<cmath>`, којом се добија цео део датог реалног броја, али опет у облику реалног броја.

```
double duzina_kanapa, duzina_pertle;  
cin >> duzina_kanapa >> duzina_pertle;
```

```
double broj_pertli = floor(duzina_kanapa / duzina_pertle);  
double preostalo_kanapa = duzina_kanapa - (broj_pertli * duzina_pertle);
```

```
cout << broj_pertli << endl;  
cout << fixed << showpoint << setprecision(4)  
     << preostalo_kanapa << endl;
```

Један начин да се одреди цео део количника $\left\lfloor \frac{a}{b} \right\rfloor$ два реална броја a и b је да се изврши конверзија у целобројни тип тј. да се резултат додели целобројној променљивој. При оваквом решењу треба водити рачуна о опсегу променљивих јер је могуће да се неке вредности могу представити реалним, али не и целобројним типом података.

```
double broj_pertli = (int)(duzina_kanapa / duzina_pertle);  
double preostalo_kanapa = duzina_kanapa - (broj_pertli * duzina_pertle);
```

Задатак: Точак

Напиши програм који на основу пречника точка бицикла, брзине кретања бицикла и времена које се бицикл креће одређује колико пута се точак окренуо (рачунајући и окрет који још није потпуно завршен).

Улаз: Са стандардног улаза се учитавају реални бројеви који представљају пречник точка бицикла у cm, брзину кретања бицикла у $\frac{\text{m}}{\text{s}}$ и време кретања бицикла у s. Подаци су такви да точак није довршио последњи окрет који је започео.

Изназ: На стандардни излаз се записује један цео број који представља број започетих окрета бицикла.

Пример

<i>Улаз</i>	<i>Излаз</i>
60	14
5	
5	

Решење

У задатку ћемо користити реалну аритметику и све вредности ћемо представити реалним бројевима. Број окрета точка једнак је количнику пређеног пута бицикла и обима точка, при чему треба водити рачуна о коришћењу истих јединица мере (користићемо метре). Обим точка у метрима може се израчунати формулом $O = \frac{R}{100} \cdot \pi$, где је R пречник точка у сантиметрима (начини како да у свом програму употребите константу π описани су у задатку **Стољњак**). Пошто се ради о равномерном кретању, пређени пут s једнак је производу брзине и времена кретања (ово је описано у задатку **Путовање**). Број кругова k је најмањи број такав да важи $O \cdot k \geq s$. Тај број се може израчунати као $k = \lceil \frac{s}{O} \rceil$, где $\lceil a \rceil$ означава заокруживање навише, тј. најмањи цео број који је већи или једнак броју a . Дакле, пошто се рачунају и започети кругови, количник пређеног пута и обима точка је потребно заокружити навише. Пошто радимо са реалним бројевима, то је могуће урадити библиотечком функцијом заокруживања навише. У језику C++ то је функција `ceil`, декларисана у заглављу `<cmath>`, којом се добија вредност реалног броја заокруженог навише, али опет у облику реалног броја.

```
double precnik_tocka_u_cm, brzina_bicikla_u_m_s, vreme_u_s;
cin >> precnik_tocka_u_cm >> brzina_bicikla_u_m_s >> vreme_u_s;

double obim_tocka_u_m = (precnik_tocka_u_cm / 100) * M_PI;
double predjeni_put_u_m = brzina_bicikla_u_m_s * vreme_u_s;
double broj_zapocetih_krugova = ceil(predjeni_put_u_m / obim_tocka_u_m);

cout << broj_zapocetih_krugova << endl;
```

Задатак: Скалирање

Професор је обећао ученицима да ће број поена који су освојили на контролном задатку пропорционално увећати тако да Пера који је најбоље урадио и освојио 95 поена добије максимални број од 100 поена. Међутим, овом операцијом цели бројеви поена се претварају у реалне, могуће са пуно или чак бесконачно децимала. Зато је професор одлучио да увећани број поена заокружи са прецизношћу од пола поена. На пример, број 87.368 поена се заокружује на 87.5, 54.25 поена на 54.5, а 73.75 се заокружује на 74 поена. Приметимо да се када постоје две могућности заокружује увек навише (као у последња два примера). Написати програм који за унети број поена одређује његову вредност заокружену на овај начин.

Улаз: Са стандардног улаза учитава се један цео број између 0 и 95 који представља број поена ученика пре скалирања.

Излаз: На стандардни излаз исписати један реалан број који представља број поена ученика након скалирања.

Пример 1		Пример 2		Пример 3	
<i>Улаз</i>	<i>Излаз</i>	<i>Улаз</i>	<i>Излаз</i>	<i>Улаз</i>	<i>Излаз</i>
95	100	84	88.5	0	0

Решење

Ако је p број поена пре скалирања, а p' број поена након скалирања, p' се може израчунати из пропорције $p : p' = 95 : 100$ тј. $p' = p/95 \cdot 100$. Овај број је потребно заокружити на најближи број чија је једина децимала 0 или 5. Бројеви са таквим децималним записом су они који представљају цео број половина. Број половина у броју p' је $p' \cdot 2$ и то не мора бити цео број, али се може заокружити на најближи цео број. Када се добије цео број половина, тражена вредност броја се може добити дељењем тог броја половина са 2.

Ако допустимо коришћење реалних бројева заокруживање се може вршити библиотечком функцијом заокруживања реалног броја на њему најближи цео броју. У језику C++ то заокруживање се врши библиотечком функцијом `round`, декларисаном у заглављу `<cmath>`. При коришћењу функција за заокруживање реалних бројева, треба водити рачуна да због немогућности потпуно прецизног записа неких бројева може доћи до грешака.

```
int poeni;  
cin >> poeni;  
double skalirani_nezaokruzeni = (poeni / 95.0) * 100.0;  
double skalirani_zaokruzeni = round(skalirani_nezaokruzeni * 2.0) / 2.0;  
cout << fixed << showpoint << setprecision(1)  
      << skalirani_zaokruzeni << endl;
```

Један начин да се задатак реши је да се посматрају децимале броја p' . Њих је могуће одредити тако што се одреди $p' - \lfloor p' \rfloor$ (слично као у задатку [Динари и паре](#)). Ако је вредност тог израза у интервалу $[0, 0.25)$ тј. ако је та вредност већа или једнака нули, а строго мања од 0.25, тада се вредност треба заокружити на $\lfloor p' \rfloor$, ако је у интервалу $[0.25, 0.75)$, тада се вредност треба заокружити на $\lfloor p' \rfloor + 0.5$, а ако је у интервалу $[0.75, 1)$ тада се вредност треба заокружити на $\lfloor p' \rfloor + 1$. Оваква класификација на основу припадности дисјунктним интервалима може се реализовати гранањем (обично угнеђеним, коришћењем конструкције `else-if` како је објашњено у задатку [Агрегатно стање](#)).

Задатак: Кружна мета

Дата је кружна мета од n концентричних кругова полупречника од a cm до $n \cdot a$ cm. За погодак у унутрашњост најмањег круга добија се n поена, а за погодак у унутрашњост сваког следећег прстена добија се по поен мање. Написаи програм којим се одређује колико се добија поена за погодак у тачку $A(x, y)$ унутар мете.

Улаз: У првом реду стандардног улаза налази се број кругова, цео број n ($1 \leq n \leq 20$), у другом ширина једног круга, реалан број a , у трећем реду x -координата тачке која је погођена, а у другом њена y -координата. Координате су реални бројеви, а тачка се сигурно налази унутар мете ($\sqrt{x^2 + y^2} < n \cdot a$) и при томе није ни на једној од кружних линија које представљају гранцу кружних поља.

Израз: Једини ред стандардног израза треба да садржи освојени број поена.

Пример

Улаз	Израз
10	5
1.0	
1.2	
5.3	

Задатак: Успех ученика

Овај задатак је ионовљен у циљу увежбавања различитих техника решавања. [Види икекциј задатак](#).

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

[Види групација решења овог задатак](#).

4.2.2 Проблеми тачности записа реалних бројева

Већина реалних бројева има бесконачно много децимала (познати пример је број π). Такве бројеве не можемо да запишемо потпуно тачно ни на папиру ни у рачунару (нити било где друго) јер би за то било потребно бесконачно много простора и времена. Због тога, када рачунамо са реалним бројевима, ми у пракси скоро увек рачунамо са приближним вредностима мање или веће тачности.

Као што знамо, савремени рачунари су у стању да веома брзо изводе рачунске операције са реалним бројевима (оним који се могу записати у рачунару). Да би те операције биле тако брзе, реални бројеви се у рачунару памте на одређени стандардан начин који је условљен архитектуром рачунара и **не зависи од програмског језика** који користимо. Тај стандардан начин памћења реалних бројева подразумева да се користи простор одређене величине и у њему се памти неки коначан број значајних цифара броја. Значајне цифре броја су све његове цифре почевши од прве цифре која није нула. На пример, за број 0,0003400 записан у декадном систему, значајне цифре су 3400 (у овом примеру чињеница да се прва значана цифра појављује на четвртој децималу била би засебно памћена). Нуле на крају броја су у овом случају значајне јер показују тачност приближне вредности коју користимо.

Приликом рачунања са приближним вредностима реалних бројева може доћи до губитка тачности. На пример, ако рачунамо користећи 3 значајне цифре, дељењем 1 са 3 добијамо резултат 0,333 као приближну вредност једне трећине. Множењем овог резултата са 3 добијамо 0,999 уместо 1. Видимо да се појавила

грешка величине 0,001. Овај проблем се не може отклонити тако што рачунамо са већим бројем значајних цифара. Ако бисмо на пример рачунали са 100 значајних цифара, при истим операцијама бисмо добили грешку на стотој децимали. Према томе, грешке се појављују самим тим што нисмо у стању да операције са реалним бројевима изводимо потпуно тачно (било да користимо рачунар или не).

За памћење реалних бројева у рачунару се користи бинарни систем. Због тога чак и неки бројеви који у декадном систему имају врло кратак тачан запис, у рачунару не могу да буду тачно записани (јер тачан запис таквог броја у бинарном систему може да има бесконачно много цифара). На пример, иако бисмо очекивали да следећа наредба испише `da`, она ће исписати `ne` (проверите).

```
cout << (0.1 + 0.2 == 0.3 ? "da" : "ne");
```

Исписивањем вредности $0.1 + 0.2 - 0.3$ са 30 децимала добијамо 0.0000000000000000055511151231258. Овакав резултат се добија зато што се реални бројеви какве смо до сада користили у програмима (типа `double`), у рачунарима памте са 52 значајне цифре у бинарном систему, што је приближно 16 значајних цифара у декадном систему. При датом једноставном рачунању дошло је до грешке на позицији последње значајне цифре и тако добијамо нулте цифре после 16 децимала једнаких нули.

Овим објашњењем смо само назначили проблем са којим се суочавамо и овде се нећемо бавити његовим детаљнијим проучавањем. Ипак, већ из наведеног се могу извући неки савети за практичан рад.

Претпоставимо да су подаци са којима рачунамо по вредности између 0 и 100 и да су дати са 5 децимала. То значи да имамо до 7 значајних цифара у тим подацима (2 значајне цифре пре децималне тачке и 5 после ње). Уколико после неког рачунања добијемо изразе `a` и `b`, који се разликују на шестој значајној цифри, сматраћемо да је та разлика стварна и да дати изрази и треба да буду различити. Међутим, ако је разлика тек на четрнаестој значајној цифри, треба закључити да су изрази `a` и `b` у ствари једнаки, а да је до разлике дошло услед ограничене тачности записа реалних бројева у рачунару. Одавде следи да изразе `a` и `b` не треба поредити простим испитивањем услова `a == b`, него испитивањем да ли се вредности `a` и `b` разликују довољно мало (занемарљиво мало), дакле испитивањем услова облика

$$\text{abs}(a - b) < \text{eps}$$

где је `eps` (скраћено од грчког слова ε тј. епсилон) мала позитивна константа, коју треба одредити на основу очекиваних величина и тачности података. Напоменимо да се вредност 0.1 може писати и као $1e-1$, вредност 0.01 као $1e-2$, вредност 0.001 као $1e-3$ итд. У поменутом примеру за `eps` можемо да изаберемо на пример 0.000000001 или краће $1e-9$ (уместо 9 би могао да стоји и неки други број већи од 7 а мањи од 16).

Слично овоме, ако реалан број `x` (уз исте претпоставке о величини и тачности података и вредности `eps`) заокружимо на мањи или једнак цео број, уместо `floor(x)` боље је писати `floor(x + eps)`.

Исто важи и за заокруживање броја `x` на већи или једнак цео број, где би уместо `x` требало писати `x - eps`.

Задатак: Динари и пара

Напиши програм који за дати реални број динара одређује одговарајући цео број динара и цео број пара.

Улаз: Са стандардног улаза се уноси један реалан број, не већи од 1000, заокружен на две децимале, који представља износ новца.

Излаз: На стандардни излаз написати два цела броја, сваки у посебном реду, који представљају број динара и број пара.

Пример

Улаз	Излаз
2.30	2
	30

Решење

Број динара d се може добити тако што се унети износ k заокружи наниже ($d = \lfloor k \rfloor$). Број пара p може се добити тако што се од износа k одузме број динара и резултат помножи са 100 ($p = (k - \lfloor k \rfloor) \cdot 100$).

У језику C++ заокруживање реалног броја наниже врши се библиотечком функцијом `floor`, декларисаном у заглављу `<cmath>`. Међутим, ако се број динара и пара представи целобројним променљивама може доћи до неочекиваних резултата. Наиме приликом сваке доделе реалне вредности целобројној променљивој врши се

4.2. ВЕЗЕ ИЗМЕЂУ ЦЕЛИХ И РЕАЛНИХ БРОЈЕВА

њена конверзија заокруживањем наниже. Иако делује да вредност $(k - \lfloor k \rfloor) \cdot 100$ мора бити целобројна, јер број k по претпоставци има само две децимале, због непрецизности рада са бројевима у покретном зарезу могуће је да се деси да та вредност понекад буде мало мања него што је то очекивано (такав је случај управо са примером из поставке задатка, за износ $k = 2.30$ добија се $p = (k - \lfloor k \rfloor) \cdot 100 = 29.999999999999982$). У том случају ће се приликом доделе целобројној променљивој добити вредност која је за један мања него што треба да буде. Да се то не би догађало, довољно је пре доделе целобројној променљивој извршити заокруживање на најближу целу вредност. У језику C++ то се може урадити библиотечком функцијом `round`, декларисаном у заглављу `<cmath>`.

```
double iznos;
cin >> iznos;
int dinari = (int)floor(iznos);
int pare = (int)round((iznos - floor(iznos)) * 100.0);
cout << dinari << endl << pare << endl;
```

Задатак: Лимунада

Одредити колико флаша дате запремине је могуће напунити датом количином лимунаде.

Улаз: Са стандардног улаза учитавају се два позитивна реална броја. У првом реду дата је запремина флаше у литрима као позитиван реалан број на до три децимале, а у другом је дата расположива количина лимунаде у литрима, такође на до три децимале. Запремина флаше није већа од 5 литара, а колична лимунаде није већа од 50 литара.

Изназ: На стандардни излаз исписати цео број који представља број флаша које је могуће напунити.

Пример

Улаз	Изназ
0.2	3
0.6	

Решење

Основна формула се добија као у задатку [Пергла](#). Нека је запремина флаше F , а лимунаде L . Број флаша N је највећи цео број N такав да је $N \cdot F \leq L$. То је број $N = \lfloor \frac{L}{F} \rfloor$, где заграде $\lfloor \dots \rfloor$ означавају заокруживање наниже тј. највећи цео број који није већи од датог броја. Дакле, број флаша се одређује као цео део количника количине лимунаде и запремине флаше.

Користећи изведену формулу директно, на примеру датом у поставци задатка добијамо резултат 2 уместо 3. То је зато што је резултат дељења $0.6 / 0.2$ једнак 2.9999999999999996 , који кад се заокружи наниже даје 2.

```
double vFlase, vLimunade;
cin >> vFlase >> vLimunade;
int brFlasa = floor(vLimunade / vFlase);
cout << brFlasa << endl;
```

Овај проблем ћемо избећи ако количнику $\frac{L}{F}$ додамо мали позитиван број пре заокруживања. Остаје да видимо колики тај број треба да буде.

Најмања вредност количника $\frac{L}{F}$ коју уопште можемо добити је количник најмање количине лимунаде и највеће запремине флаше, дакле $0.001 / 5 = 0.0002$. То значи да се флаше пуне у корацима (“квантима”) од по најмање 0.0002 флаше. Према томе, додавањем броја eps довољно мањег од 0.0002 нашем количнику (на пример $eps = 0.000001 = 1e - 6$), нећемо пребацити следећи цео број, то јест такво додавање неће покварити резултат када је он тачан. Са друге стране, ако је резултат требало да буде цео број, али је због ограничене тачности реалних бројева мањи од целог за рецимо $1e - 14$ (или мање), додавањем eps ћемо управо пребацити следећи цео број и тиме исправити грешку.

```
double vFlase, vLimunade;
cin >> vFlase >> vLimunade;
int brFlasa = floor(1e-6 + vLimunade / vFlase);
cout << brFlasa << endl;
```

4.2.3 Заокруживање количника целобројним дељењем

Ако су n и k природни бројеви, тада се вредности $\lfloor \frac{n}{k} \rfloor$ и $\lceil \frac{n}{k} \rceil$ могу израчунати и само помоћу операција над целим бројевима. Тиме избегавамо употребу реалних бројева и све опасности до којих може доћи услед немогућности потпуно прецизног записа реалних бројева.

Доказаћемо да је број $\lfloor \frac{n}{k} \rfloor$ једнак целобројном количнику бројева n и k који ћемо означавати са $n \operatorname{div} k$. Рачунањем целобројног количника избегавамо употребу реалног дељења и функција за рад са реалним бројевима. Нека је $n = q \cdot k + r$, тако да је $0 \leq r < k$. Тада је $q \cdot k + 0 \leq q \cdot k + r < q \cdot k + k$ тј. $q \cdot k \leq n < (q + 1) \cdot k$, тј. $q \leq \frac{n}{k} < q + 1$. Зато је $q = \lfloor \frac{n}{k} \rfloor$. Међутим, на основу основне дефиниције целобројног количника, коју смо навели у задатку **Разломак у мешовит број**, важи да је $q = n \operatorname{div} k$.

Програмски језици по правилу дају могућност да се директно, целобројним дељењем, израчуна $\lfloor \frac{n}{k} \rfloor$ (то је, како смо већ доказали, целобројни количник бројева n и k тј. број $n \operatorname{div} k$), али не и да се директно израчуна $\lceil \frac{n}{k} \rceil$. Постоји неколико начина да се то уради посредним путем.

Први начин се заснива на томе да је $\lceil \frac{n}{k} \rceil$ једнако $\frac{n}{k}$ када је n дељиво са k (без остатка), а да је једнако $\lfloor \frac{n}{k} \rfloor + 1$ тј. $n \operatorname{div} k + 1$ када постоји остатак. На пример, ако је $n = 18$ и $k = 3$, тада је $\frac{n}{k} = 6$ и то је управо најмањи цео број већи или једнак (а уједно и мањи или једнак) вредности 6, тј. $\lceil \frac{n}{k} \rceil = \lfloor \frac{n}{k} \rfloor = \frac{n}{k} = 6$. Ако је $n = 18$, а $k = 4$, тада је $\frac{n}{k} = 4.5$, па је $\lfloor \frac{n}{k} \rfloor = 4$, $\lceil \frac{n}{k} \rceil = 5$ и важи да је $\lceil \frac{n}{k} \rceil = \lfloor \frac{n}{k} \rfloor + 1$.

Докажимо претходно тврђење и формално. Ако постоје q и r такви да је $n = q \cdot k + r$, где је $0 \leq r < k$, тада важи да је $q = n \operatorname{div} k$, $r = n \operatorname{mod} k$ и $\frac{n}{k} = \frac{q \cdot k + r}{k} = q + \frac{r}{k}$. Пошто је $0 \leq \frac{r}{k} < 1$, важи да је $q \leq \frac{n}{k} < q + 1$. Зато је и $\lfloor \frac{n}{k} \rfloor = q$, тј. $\lfloor \frac{n}{k} \rfloor$ заправо представља вредност целобројног количника n и k тј. вредност $n \operatorname{div} k$, што смо и раније доказали. Ако је $r = 0$, тада важи да је $\frac{n}{k} = q$ и q је најмањи број такав да већи или једнак од $\frac{n}{k}$, тј. $\lceil \frac{n}{k} \rceil = q = n \operatorname{div} k = \frac{n}{k}$. Ако је $r > 0$, тада је $\frac{r}{k} > 0$, па је $q < \frac{n}{k}$. Зато је $q + 1$ најмањи број већи или једнак (у овом случају је већи) од $\frac{n}{k}$, тј. $\lceil \frac{n}{k} \rceil = q + 1 = n \operatorname{div} k + 1 = \lfloor \frac{n}{k} \rfloor + 1$.

Други начин је да се примети да важи да је

$$\lceil \frac{n}{k} \rceil = \lfloor \frac{n+k-1}{k} \rfloor = (n + k - 1) \operatorname{div} k.$$

Размотримо вредности $\lceil \frac{n}{k} \rceil$ и $\lfloor \frac{n}{k} \rfloor$ за $k = 4$ и разне вредности n .

n	...	8	9	10	11	12	13	14	15	16	...
$\lfloor \frac{n}{4} \rfloor$...	2	2	2	2	3	3	3	3	4	...
$\lceil \frac{n}{4} \rceil$...	2	3	3	3	3	4	4	4	4	...

Приметимо да за вредности $n = 9$, $n = 10$, $n = 11$ и $n = 12$ важи да је $\lceil \frac{n}{4} \rceil = 3$. Са друге стране за вредности $n = 12$, $n = 13$, $n = 14$ и $n = 15$ важи да је $\lfloor \frac{n}{4} \rfloor = 3$. Додавањем вредности $k - 1 = 3$ на полазни број n тј. на бројеве 9, 10, 11 или 12 добијају се редом бројеви 12, 13, 14 и 15 и израчунавањем целобројног количника добија се вредност 3, што је и било потребно.

Да бисмо формално доказали да важи $\lceil \frac{n}{k} \rceil = \lfloor \frac{n+k-1}{k} \rfloor$, опет претпоставимо да је $n = q \cdot k + r$, где је $0 \leq r < k$. Тада је $n + k - 1 = q \cdot k + r + k - 1$. Зато је $\frac{n+k-1}{k} = q + \frac{r+k-1}{k}$. Пошто је $0 \leq r < k$, важи и да је $0 \leq \frac{0+k-1}{k} \leq \frac{r+k-1}{k} < \frac{k+k-1}{k} < 2$, па је $q \leq \frac{n+k-1}{k} < q + 2$. Ако је $r = 0$, тада је $\frac{n+k-1}{k} = q + \frac{k-1}{k}$ и пошто је $\frac{k-1}{k} < 1$, тада је $q \leq \frac{n+k-1}{k} < q + 1$, па важи $\lfloor \frac{n+k-1}{k} \rfloor = q$, а већ раније смо показали да је у случају $r = 0$ важи да је $q = \frac{n}{k} = \lceil \frac{n}{k} \rceil$. Ако је $r > 0$ тада је $\frac{r+k-1}{k} \geq 1$, па је $\frac{n+k-1}{k} \geq q + 1$. Пошто већ знамо да је $\frac{n+k-1}{k} < q + 2$, важи да је $\lfloor \frac{n+k-1}{k} \rfloor = q + 1$ за који смо већ показали да је у случају $r > 0$ једнак броју $\lfloor \frac{n}{k} \rfloor + 1$ тј. броју $\lceil \frac{n}{k} \rceil$.

Напоменимо да мали проблем у имплементацији ове технике представља могућност да збир $n + k - 1$ прекорачи опсег целобројног типа и ову технику није пожељно примењивати када n и k могу да буду релативно велики бројеви (колико велики, зависи од њиховог целобројног типа).

Задатак: Поклони

Сваки поклон садржи тачно k чоколадица. Ако на располагању имамо укупно n чоколадица, колико поклона је могуће запаковати?

4.2. ВЕЗЕ ИЗМЕЂУ ЦЕЛИХ И РЕАЛНИХ БРОЈЕВА

Улаз: Са стандардног улаза се читавају два цела броја (сваки у посебном реду):

- k ($1 \leq k \leq 20$) - број чоколадица у једном поклону
- n ($0 \leq n \leq 1000$) - укупан број чоколадица

Излаз: На стандардни излаз исписати један цео број - највећи број поклона које је могуће направити.

Пример

Улаз	Излаз
4	4
19	

Решење

Ако је број поклона x , број чоколадица у једном поклону k , а укупан број чоколадица n , тада треба пронаћи највеће x тако да важи да је $x \cdot k \leq n$. На пример, ако има $n = 29$ чоколадица и у сваки поклон стаје $k = 4$ чоколадица тада је могуће направити 7 поклона (тако се употреби $4 \cdot 7 = 28$ чоколадица). За 8 поклона потребно је $4 \cdot 8 = 32$ чоколадице, а толико немамо на располагању. У задатку **Пертла** доказали смо да је $x = \lfloor \frac{n}{k} \rfloor$, где заграде $\lfloor \dots \rfloor$ означавају заокруживање наниже тј. највећи цео број мањи од датог броја. Пошто су у задатку дати цели бројеви n и k најлакше и најбоље решење је израчунати њихов целобројни количник.

```
int k, n;  
cin >> k >> n;  
int brojPoklona = n / k;  
cout << brojPoklona << endl;
```

Да су бројеви реални, могуће би било израчунати њихов реални количник и заокружити га наниже библиотечком функцијом `floor`.

```
double k, n;  
cin >> k >> n;  
double brojPoklona = floor(n/k);  
cout << brojPoklona << endl;
```

Задатак: Лифт

У једном хотелу n људи чека испред лифта. У лифт може да стане њих k . Колико је најмање возњи лифтом потребно да се сви попну у своје собе?

Улаз: Са стандардног улаза читавају се два цела броја, сваки у посебном реду:

- укупан број људи испред лифта n ($0 \leq n \leq 200$)
- број људи који могу да одједном стану у лифт k ($1 \leq k \leq 10$)

Излаз: На стандардни излаз исписати један цео број - најмањи потребан број возњи лифтом.

Пример 1

Улаз	Излаз
18	6
3	

Пример 2

Улаз	Излаз
18	5
4	

Решење

Означимо број људи који се могу превести у једној возњи са k , укупан број људи са n , а број возњи са x . Потребно је пронаћи најмање x такво да важи $x \cdot k \geq n$. Пошто су n и k позитивни цели бројеви, потребно је пронаћи најмање x тако да је $x \geq \frac{n}{k}$. У задатку **Точак** доказали смо да је то вредност $x = \lceil \frac{n}{k} \rceil$, где заградама $\lceil \dots \rceil$ означавамо заокруживање навихше, на најмањи цео број већи или једнак датом броју.

Испитивање дељивости

Један начин да целобројни количник заокружимо навихше је испитивање дељивости, које се може урадити на неколико начина. Може се употребити тернарни условни оператор.

```
int n, k;  
cin >> n >> k;  
int brojVoznji = (n % k == 0) ? (n / k) : (n / k + 1);  
cout << brojVoznji << endl;
```

Може се употребити наредба гранања `if` и ако је n дељиво са k накнадно увећати број возњи добијен као $\lfloor \frac{n}{k} \rfloor$ тј. $n \text{ div } k$.

```
int n, k;
cin >> n >> k;
int brojVoznji = n / k;
if (n % k != 0)
    brojVoznji++;
cout << brojVoznji << endl;
```

Може се употребити и чињеница да се логичка вредност `true` може имплицитно конвертовати у целобројну вредност 1, а `false` у 0 и израз $(n / k) + (n \% k \neq 0)$ (на целобројни количник додаје се 1 ако постоји остатак и 0 ако остатка нема) даје тражени резултат.

```
int n, k;
cin >> n >> k;
int brojVoznji = (n / k) + (n % k != 0);
cout << brojVoznji << endl;
```

Померање

До решења можемо доћи на основу везе $\lceil \frac{n}{k} \rceil = \lfloor \frac{n+k-1}{k} \rfloor$.

```
int n, k;
cin >> n >> k;
int brojVoznji = (n + k - 1) / k;
cout << brojVoznji << endl;
```

Коришћење заокруживања реалних бројева

Један начин да се количник заокружи навише је да се користи аритметика са реалним бројевима и библиотечка функција `ceil` декларисана у заглављу `<cmath>`, која врши заокруживање реалних бројева навише, уз све опасности које рад са реалним бројевима носи.

```
int n, k;
cin >> n >> k;
int brojVoznji = ceil((double)n / (double)k);
cout << brojVoznji << endl;
```

Задатак: Оцена на испиту

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види шексџи задатак.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Аритметика

Једно могуће решење се заснива на посебној провери да ли је студент пао испит (добио оцену 5) и затим на коришћењу аритметике да би се одредила оцена ако је положио. Најмањи број поена потребан за оцену 6 је 51, за оцену 7 је 61 итд. Дакле, најмањи број поена потребан да би се добила оцена o је $10 \cdot (o - 1) + 1$. Ако је познат број поена p , одговарајућа оцена је највећа оцена o таква да је $10 \cdot (o - 1) + 1 \leq p$ тј. да је $10 \cdot (o - 1) \leq p - 1$. На основу задатка **Поклони** знамо да је највећи број x такав да је $10 \cdot x \leq p - 1$ број $\lfloor \frac{p-1}{10} \rfloor$, па је зато $o = \lfloor \frac{p-1}{10} \rfloor + 1$. Заиста, целобројним дељењем бројем 10, интервал $[50, 59]$ се пресликава у број 5, интервал $[60, 69]$ у број 6 и слично. Зато, ако се од од бројева из интервала $[51, 60]$ одузме 1, одреди целобројни количник са 10 и на резултат дода 1, добија се тражена оцена 6. Слично важи и за све наредне интервале, тако да се оцена може израчунати тако што се од броја поена одузме 1, израчуна целобројни количник са 10 и на то дода 1.

```
// poeni osvojeni na ispitu
int poeni;
cin >> poeni;
// ocena na ispitu
int ocena = poeni < 51 ? 5 : (poeni - 1) / 10 + 1;
```

```
// prikaz rezultata
cout << ocena << endl;
```

Задатак: Крушке

Учитељица је купила k крушака, које жели да подели ученицима из свог одељења. Ако у одељењу има u ученика, колико још најмање крушака учитељица треба да купи да би била сигурна да ће сваком детету моћи да да исти број крушака.

Улаз: Са стандардног улаза уносе се два природна броја (сваки у посебном реду):

- k ($0 \leq k \leq 300$) број крушака које учитељица тренутно има.
- u број ученика у одељењу ($15 \leq u \leq 30$).

Излаз: На стандардни излаз исписати један цео број који представља најмањи број крушака које учитељица треба да купи.

Пример 1

Улаз Излаз
73 19
23

Пример 2

Улаз Излаз
100 0
20

Решење**Гранање**

Учитељица деци може да подели све крушке које има тако да сви добију исто. Након поделе остаће јој $k \bmod u$ крушака. Ако се испостави да јој није преостала ни једна крушка (тј. ако је k дељиво са u), тада не мора ништа да докупљује. Ако јој остане одређени број крушака, тада је потребно да докупи толико још крушака да има тачно u крушака да би сваком детету могла да да још једну крушку, тј. потребно је да докупи $u - k \bmod u$ крушака. На основу овога задатак се може једноставно решити коришћењем гранања.

```
int krusaka, ucenika;
cin >> krusaka >> ucenika;
int nedostaje;
if (krusaka % ucenika == 0)
    nedostaje = 0;
else
    nedostaje = ucenika - krusaka % ucenika;
cout << nedostaje << endl;
```

Најмањи број већи или једнак k који је дељив бројем u

Приметимо да се у овом задатку суштински тражи да се одреди најмањи могући број већи или једнак од броја k који је дељив бројем u . То је број $u \cdot \lceil \frac{k}{u} \rceil$. Заиста, слично као у задатку **Лифт** може се приметити да ће након куповине свако дете добити $\lceil \frac{k}{u} \rceil$ крушака. Онда је потребно да након куповине учитељица има $u \cdot \lceil \frac{k}{u} \rceil$ крушака, а потребно је докупити $u \cdot \lceil \frac{k}{u} \rceil - k$ крушака, што се може директно израчунати коришћењем било ког начина за заокруживање количника навише који су приказани у задатку **Лифт**.

Овај израз је могуће упростити. Претпоставимо да се при дељењу броја n бројем k добија целобројни количник $q = k \operatorname{div} u$ и остатак $r = k \bmod u$ тј. да важи $k = q \cdot u + r$, за $0 \leq r < u$. Ако је $r = 0$, тада учитељица не мора ништа да купује (заиста, важи да је $u \cdot q = u \cdot \lceil \frac{k}{u} \rceil = u \cdot \frac{k}{u} = k$). Ако је $r \neq 0$, тада је следећи садржалац броја ученика једнак $u \cdot (q + 1)$. Потребан број крушака до њега је $u \cdot (q + 1) - k = u \cdot q + u - k = u - r$, тј. $u - k \bmod u$. Решење је у оба случаја једнако $(u - k \bmod u) \bmod u$.

```
int krusaka_po_uceniku = (krusaka + ucenika - 1) / ucenika;
int ukupno_krusaka = krusaka_po_uceniku * ucenika;
int nedostaje = ukupno_krusaka - krusaka;
```

Задатак: Планинари

Планинари се пењу на врх планине чија је висина задата у метрима. Дању се могу попети према горе извештан број метара, али током ноћи спавају, па се морају спустити за одређен број метара. Напишите програм који ће одредити колико дана треба планинарима да се попну до врха планине.

Улаз: У прва три реда стандардног улаза налазе се природни бројеви G, D, H ($1 \leq D < G \leq H \leq 10^9$) који редом означавају: - G - висину метара коју планинари пређу дању пењући се навише, - D - висину метара коју се планинари спусте ноћу, - H - висина планине.

Излаз: У први и једини ред стандардног излаза испишите број дана који ће проћи док планинари не стигну до врха.

Пример

Улаз	Излаз
5	2
1	
6	

Решење

Решавање неједначине

Ако се планинари током дана попну за G метара, и ноћу спусте за D метара, онда се након једног дана и једне ноћи укупно попну за $G-D$ метара. Пењање тече тако што се током X дана и ноћи они пењу и силазе, да би на крају, следећег дана достигли врх планине. Дакле, потребно је одредити најмањи цео број X такав да важи да је $X \cdot (G - D) + G \geq H$, тј. $X \cdot (G - D) \geq H - G$. На основу задатка **Лифт** важи да је $X = \left\lceil \frac{H-G}{G-D} \right\rceil$. На основу резултата који је доказан у том задатку важи да је

$$\left\lceil \frac{H-G}{G-D} \right\rceil = \left\lfloor \frac{(H-G)+(G-D-1)}{G-D} \right\rfloor = \left\lfloor \frac{H-D-1}{G-D} \right\rfloor = (H - D - 1) \operatorname{div} (G - D).$$

Након што се израчуна X , коначан резултат је $X + 1$, јер планинари долазе на врх планине током $X + 1$ дана.

```
#include <iostream>
using namespace std;
```

```
int main() {
    int g, d, h;
    cin >> g >> d >> h;
    int brojDana = (h - d - 1) / (g - d) + 1;
    cout << brojDana << endl;
    return 0;
}
```

Линеарна претрага

Један интуитиван, али неефикасан начин да се реши задатак је да се примени алгоритам линеарне претраге и да се редом у петљи за сваки дан, рачунају висине на којима се планинари налазе све док планинари не достигну висину врха планине.

```
#include <iostream>
using namespace std;
```

```
int main() {
    int g, d, h;
    cin >> g >> d >> h;

    // broj trenutnog dana
    int brojDana = 0;
    // visina na kojoj se planinari nalaze
    int visina = 0;
    while (true) {
        // naredni dan
        brojDana++;
        // planinari se penju
        visina += g;
        // proveravamo da li su dostigli vrh
        if (visina >= h)
            break;
    }
}
```

4.2. ВЕЗЕ ИЗМЕЂУ ЦЕЛИХ И РЕАЛНИХ БРОЈЕВА

```
// planinari se spustaju
visina -= d;
}

cout << brojDana << endl;
}
```

Задатак: Сечење плочица

Правоугаону терасу димензија $d \times s$ центиметара квадратних треба поплочати коришћењем плочица квадратног облика стране p центиметара, које се постављају тако да су им стране паралелне странама терасе. Написати програм којим се одређује колико се плочица мора сећи ради поплочавања, као и површину дела терасе који заузимају сечене плочице. Од сваке сечене плочице користи се један део, а други одбацује.

Улаз:

- p - страница плочице у см ($10 \leq p \leq 50$)
- d - дужина просторије у см ($200 \leq d \leq 10000$)
- s - ширина просторије у см ($200 \leq s \leq 10000$)

Издаз: Број плочица које се морају сећи и површина коју покривају сечене плочице.

Пример

Улаз	Издаз
20	29
310	5700
270	

Решење

Нека је:

- p - дужина стране плочице у см,
- d - дужина просторије у см,
- s - ширина просторије у см.

Тражени број плочица које се морају сећи добија као разлика минималног броја плочица које покривају правоугаону област: $\left\lceil \frac{d}{p} \right\rceil \cdot \left\lceil \frac{s}{p} \right\rceil$ и максималног броја плочица које су садржане у правоугаоној области: $\left\lfloor \frac{d}{p} \right\rfloor \cdot \left\lfloor \frac{s}{p} \right\rfloor$. Заиста максимални број плочица који се без сечења може сложити по дужини терасе је највећи број n за који важи да је $n \cdot p \leq d$, а на основу задатка **Поклони** то је управо $\left\lfloor \frac{d}{p} \right\rfloor$. Слично, минимални број плочица које се морају сложити по дужини терасе да би покриле целу дужину терасе (уз евентуални вишак) је најмањи број n такав да је $n \cdot p \geq d$, а на основу задатка **Лифт** то је управо $\left\lceil \frac{d}{p} \right\rceil$. За ширину важе потпуно аналогни закључци.

Површина коју покривају сечене плочице добија се као разлика површине терасе: $d \cdot s$ и површине коју покривају плочице које нису сечене: $\left\lfloor \frac{d}{p} \right\rfloor \cdot p \cdot \left\lfloor \frac{s}{p} \right\rfloor \cdot p$.

Заокруживање целобројног количника се може урадити на разне начине (на пример, коришћењем чињенице да је $\left\lfloor \frac{d}{p} \right\rfloor = d \operatorname{div} p$ и да је $\left\lceil \frac{d}{p} \right\rceil = \left\lfloor \frac{d+p-1}{p} \right\rfloor = (d+p-1) \operatorname{div} p$, како је детаљно објашњено у задацима **Поклони** и **Лифт**.

```
// dimenzija plocice i duzina i sirina prostorije
int p, d, s;
cin >> p >> d >> s;

int dfloor = d / p, dceil = (d + p - 1) / p;
int sfloor = s / p, sceil = (s + p - 1) / p;
// dceil * sceil - minimalni broj plocica koje pokrivaju oblast
// dfloor * sfloor - maksimalni broj plocica sadrzanih u oblasti
int brojSecenihPlocica = (dceil * sceil) - (dfloor * sfloor);
int površinaPokrivenaSecenimPlocicama = (d*s) - (dfloor*p*sfloor*p);
```



```
cout << brojSecenihPlocica << endl;
cout << površinaPokrivenaSecenimPlocicama << endl;
```

Заокруживање је могуће извршити и коришћењем реалне аритметике и библиотечких функција `floor` и `ceil` декларисаним у заглављу `<cmath>`.

```
int dfloor = (int)floor((double)d / (double)p),
    dceil = (int)ceil ((double)d / (double)p);
int sfloor = (int)floor((double)s / (double)p),
    sceil = (int)ceil ((double)s / (double)p);
// dceil * sceil - minimalni broj plocica koje pokrivaju oblast
// dfloor * sfloor - maksimalni broj plocica sadrzanih u oblasti
int brojSecenihPlocica = (dceil * sceil) - (dfloor * sfloor);
int površinaPokrivenaSecenimPlocicama = (d*s) - (dfloor*p*sfloor*p);
```

Један начин да се задатак реши је да се провери колики је вишак дужине v_d , вишак ширине v_s и да се на основу тога израчунају тражени подаци. Вишак ширине једнак је $v_s = s \bmod p$, а вишак дужине $v_d = d \bmod p$. Ако постоји вишак ширине, онда се у део који је покривен сеченим плочица сигурно уграђује $d \div p$ плочица чија је димензија $p \times v_s$. Ако постоји вишак дужине, онда се уграђује и $s \div p$ плочица чија је димензија $p \times v_d$. На крају, ако постоји и један и други вишак, потребна је и једна мала плочица димензије $v_d \times v_s$ у ћошку.

```
int brojSecenihPlocica = 0;
int površinaPokrivenaSecenimPlocicama = 0;

// visak po duzini i visak po sirini
int visakD = d % p, visakS = s % p;

if (visakD != 0) {
    // dodajemo s / p plocica cija je dimenzija visakD x p
    brojSecenihPlocica += s / p;
    površinaPokrivenaSecenimPlocicama += visakD * p * (s / p);
}

if (visakS != 0) {
    // dodajemo d / p plocica cija je dimenzija visakS x p
    brojSecenihPlocica += d / p;
    visakS = s % p;
    površinaPokrivenaSecenimPlocicama += visakS * p * (d / p);
}

if (visakD != 0 && visakS != 0) {
    // dodajemo jednu plocicu cija je dimenzija visakD x visakS
    brojSecenihPlocica += 1;
    površinaPokrivenaSecenimPlocicama += visakD * visakS;
}
```

Задатак: Папир

Написати програм који израчунава масу одређеног броја одштампаних књига, да би издавачка кућа знала коју носивост камиона треба да наручи.

Улаз:

- Природан број *brojKnjiga* ($1 \leq brojKnjiga \leq 1000$) број одштампаних примерака књиге.
 - Природан број *brojStranica* ($30 \leq brojStranica \leq 500$), број страна једне књиге;
 - Природан број *duzinaStranice* ($50 \leq duzinaStranice \leq 297$), дужина листа тј. стране задата у милиметрима;
 - Природан број *sirinaStranice* ($50 \leq sirinaStranice \leq 210$), ширина листа тј. стране задата у милиметрима;
 - Реалан број *masaPapira* ($50 \leq masaPapira \leq 100$), маса папира у грамима по квадратном метру.

4.2. ВЕЗЕ ИЗМЕЂУ ЦЕЛИХ И РЕАЛНИХ БРОЈЕВА

– Реалан број $masaKorica$ ($10 \leq masaKorica \leq 100$), маса корица једне књиге у граммима;

Изназ:

- Реалан број који представља укупну масу књига у килограмима, заокружен на 3 децимале.

Пример

Улаз	Изназ
850	364.242
193	
260	
200	
80.0	
25.0	

Објашњење

Потребно је одштампати 850 примерака књиге која има 193 стране. Један лист папира има две стране, тако да се књига састоји од 97 листова (на последњем листу одштампана је само једна страна) и корица. Маса једног листа папира је 4.16 грама. Тежина једне књиге је зато једнака збиру масе листова која је једнака 403.52 грама и масе корица која је 25 грама, тј. једна књига је тешка 428.52 грама. Зато је укупна маса једнака 364.242 килограма.

Задатак: Кофа

У дечијем базену постоји гусарски брод на чијем се врху налази кофа која се празни на сваких k минута. Ако се зна број минута протеклих од неког ранијег пражњења кофе напиши програм који одређује број минута преосталих до првог следећег минута у коме ће се кофа сигурно испразнити (0 ако ће се кофа испразнити у тренутном минуту).

Улаз: У првом реду стандардног улаза налази се цео позитиван број k ($0 < k \leq 60$) који представља број минута између свака два узастопна пражњења кофе, а у другом број n ($0 \leq n \leq 240$) који представља број минута од неког претходног пражњења кофе.

Изназ: На стандардни излаз исписати тражени број.

Пример 1		Пример 2	
Улаз	Изназ	Улаз	Изназ
5	2	5	0
23		25	

Задатак: Воћни пакети

Животиње у зоолошком врту воле да једу воће. У врт се допремају пакети воћа у којима се налази a килограма ананаса, b килограма банана и j килограма јабука. Волонтери треба да припреме дневни оброк за који знају да треба да садржи бар A килограма ананаса, B килограма банана и J килограма јабука. Напиши програм који израчунава који је најмањи број пакета воћа које волонтери треба да наруче да би могли да припреме дневни оброк.

Улаз: Са стандардног улаза учитава се 6 целих бројева, сваки у посебном реду: a, b, j , из интервала $[1, 5]$ и A, B, J из интервала $[5, 100]$.

Изназ: На стандардни излаз исписати један цео број који представља број пакета које треба наручити.

Пример

Улаз	Изназ
3	5
5	
4	
12	
14	
17	

Задатак: Дељиви око броја

Дати су цели бројеви n и k . Напиши програм који одређује највећи цео број $l \leq n$ дељив бројем k и најмањи цео број $d \geq n$ дељив бројем k .

Улаз: Са стандардног улаза уносе се два цела броја (сваки у посебном реду).

- n ($0 \leq n \leq 100000$)
- k ($1 \leq k \leq 10000$)

Излаз: На стандардни излаз исписати два тражена цела броја, сваки у посебном реду.

Пример 1

<i>Улаз</i>	<i>Излаз</i>
23	20
5	25

Пример 2

<i>Улаз</i>	<i>Излаз</i>
49	49
7	49

Задатак: Број дељивих у интервалу

Напиши програм који одређује колико у интервалу $[a, b]$ постоји бројева дељивих бројем k .

Улаз: Са стандардног улаза уносе се три цела броја, сваки у посебном реду.

- a ($0 \leq a \leq 10^9$)
- b ($a \leq b \leq 10^9$)
- k ($1 \leq k \leq 10^9$)

Излаз: На стандардни излаз исписати тражени цео број.

Пример

<i>Улаз</i>	<i>Излаз</i>
30	5
53	
5	

Објашњење

Бројеви су 30, 35, 40, 45 и 50.

Задатак: Судоку

Мирко је мали програмер који покушава да испрограмира игрицу судоку. Близу је да заврши, али му је потребна мала помоћ. Смислио је да корисник мишем бира квадрат у који ће уписати број. Поље се састоји од 81 квадратића распоређеног у 9 врста и 9 колона и организованог у 9 већих квадрата (као на слици). Сваки квадратић је димензије 30 пута 30 пиксела (укупно поље је димензије 270 пута 270 пиксела). Познат је положај пиксела на који је кликнуто мишем. Положај је одређен редним бројевима (координатама) тог пиксела по хоризонтали и по вертикали, рачунајући од доњег левог угла поља (пиксели се и по хоризонтали и по вертикали броје од 1 до 270). Потребно је исписати редни број врсте, колоне и већег квадрата у којем се налази пиксел на који је кликнуто (врсте се броје од 1 до 9 одоздо навише, колоне од 1 до 9 слева надесно, а квадрати по врстама од доњег левог угла, како је обележено на слици).

4.2. ВЕЗЕ ИЗМЕЂУ ЦЕЛИХ И РЕАЛНИХ БРОЈЕВА

9									
8	7			8			9		
7									
6									
5	4			5			6		
4									
3									
2	1			2			3		
1									
	1	2	3	4	5	6	7	8	9

Слика 4.1: Судоку

Улаз: Са улаза се читавају два цела броја x и y ($1 \leq x, y \leq 270$) које представљају координате пиксела на који је кликнуто.

Излаз: На стандардни излаз исписати три броја, сваки посебном реду: редни број врсте (од 1 до 9), редни број колоне (од 1 до 9) и редни број квадрата (од 1 до 9).

Пример 1

Улаз	Излаз
128	8
230	5
	8

Пример 2

Улаз	Излаз
180	6
180	6
	5

Пример 3

Улаз	Излаз
181	7
181	7
	9

Задатак: Магаџин сокова

У магаџин се на n полица слажу гаџице са соковима. У сваку гаџицу може да стане k флаша сокова. Одредити најмањи укупан број гаџица потребан да се спакују све флаше из магаџина, ако је познат број флаша сокова на свакој полици, при чему се у сваку гаџицу могу паковати искључиво флаше са једне полице.

Улаз: У првој линији стандардног улаза налази се природан број n ($1 \leq n \leq 100$) који представља број полица у магаџину. У другој је природан број k ($5 \leq k \leq 30$) који представља капацитет гаџице. У свакој од наредних n линија налази се природан број, из интервала $[5, 500]$, који представља број флаша на свакој полици.

Излаз: У једној линији стандардног излаза исписује се најмањи број употребљених гаџица.

Пример

Улаз	Излаз
3	10
8	
31	
25	
16	

Објашњење

За флаше са прве полице потребно је 4 гаџице, са друге исто 4, а са треће 2.

Види груџачија решења овој задатку.

Задатак: H₂SO₄

Овај задатку је ионовљен у циљу увежбавања различитих техника решавања. *Види текст задатку.*

Покушај да задатку урадиш коришћењем техника које се излажу у овом поглављу.

4.3 Карактерски тип података

У језику C++ постоји и подршка за представљање појединачног ASCII карактера коришћењем типа `char` који заузима 1 бајт.

Појединачни карактер можемо учитати позивом `cin >> c` или позивом `cin.get(c)` и у оба случаја враћа се истинитосна вредност која указује да ли је карактер успешно учитан (враћа се вредност `false` уколико није било могуће учитати карактер, тј. ако се дошло до краја улаза). Уколико је позив успешно завршен, у променљиву `c` се смешта вредност која одговара ASCII кôду учитаног карактера. Појединачни карактер `c` типа `char` можемо на екран исписати позивом `cout << c`, а ASCII вредност овог карактера тако што најпре извршимо његову конверзију у тип `int` а затим ту вредност испишемо на стандардни излаз: `cout << (int)c`.

Често је потребно за учитани карактер утврдити којој класи припада (цифра, мало или велико слово, знак интерпункције, белина и слично). Ово је могуће утврдити коришћењем функција декларисаних у заглављу `<ctype>`, као што су:

- `isdigit` – проверава да ли је задати карактер цифра
- `islower` – проверава да ли је задати карактер мало слово
- `isupper` – проверава да ли је задати карактер велико слово

Све ове функције као аргумент примају карактер за који испитујемо одговарајући услов, а враћају вредност различиту од 0 ако је провера успела, односно вредност 0 ако није.

Уместо коришћења библиотечких функција могуће је имплементирати решење које се ослања на чињеницу да се карактери у језику C++ интерно репрезентују својим ASCII кодовима, па се карактери могу поредити стандардним релацијским операторима (`<=`, `>=`, `<`, `>`, `==` и `!=`). Поређење карактера је заправо поређење њихових кодова. Даље, ASCII кодови имају особину да су сва мала слова и сва велика слова енглеске абетеце поређана једно иза другог по абecedном редоследу, док су цифре поређане једна иза друге по нумеричкој вредности. То омогућава да се провера да ли је карактер мало слово енглеске абетеце изврши тако што се провери да ли је његов кôд између кодова карактера 'a' и 'z' тј. да ли припада интервалу од 'a' до 'z' (ако је учитани карактер `c`, провера се врши условом 'a' <= c && c <= 'z'). Слично важи и за велика слова и за цифре (провера да ли је карактер `c` цифра може се формулисати условом '0' <= c && c <= '9').

Поред наведених функција, у заглављу `<ctype>` декларисане су и функције које конвертују мала у велика слова и обратно. Прецизније, постоје наредне функције које раде над појединачним карактерима:

- `tolower` – мало слово трансформише у велико, док остале карактере не мења
- `toupper` – велико слово трансформише у мало, док остале карактере не мења

Мало слово се може трансформисати у велико слово и тако што се прво одреди које је то по реду слово у абетеди (најбоље је да бројање креће од нуле), тако што се од тог карактера тј. његовог кода одузме карактер 'a' тј. његов кôд, а затим се нађе велико слово које има тај редни број у абecedном реду свих великих слова, тако што се на карактер 'A' тј. његов кôд дода редни број карактера. Дакле, ако је променљива `c` садржи мало слово, његову трансформацију у велико слово можемо извршити изразом 'A' + (c - 'a') тј. c - 'a' + 'A'. Трансформација великих у мала слова вршила би се потпуно аналогно, изразом c - 'A' + 'a'.

Задатак: Класификација карактера

Написати програм који за учитани ASCII карактер испитује да ли је мало слово, велико слово, цифра или нешто четврто

Улаз: У једној линији стандардног улаза налази се један карактер.

Излаз: У једној линији стандардног излаза приказати једну од следћих информација: MALO SLOVO, VELIKO SLOVO, CIFRA, OSTALO.

Пример 1

Улаз Излаз
a MALO SLOVO

Пример 2

Улаз Излаз
B VELIKO SLOVO

Пример 3

Улаз Излаз
7 CIFRA

Пример 4

Улаз Излаз
: OSTALO

Решење

Прво питање је како учитати појединачни карактер. У језику C++ то је могуће урадити или са `cin >> c` или са `cin.get(c)` и у оба случаја добијамо резултујућу истинитосну вредност која казује да ли је карактер успешно учитан.

4.3. КАРАКТЕРСКИ ТИП ПОДАКА

Испитивање да ли карактер припада некој од наведених класа најбоље је вршити коришћењем библиотечких функција. У програмском језику C++, можемо користити функције `isdigit`, `islower` и `isupper` из библиотеке `<ctype>` којима се проверава да ли је задати карактер цифра, мало или велико слово.

Пошто су сви услови међусобно искључиви, проверу услова можемо вршити угнежђеним наредбама гранања тј. конструкцијом `else-if` коју смо дискутовали у задатку **Агрегатно стање**. На пример, прво проверавамо да ли је карактер мало слово, ако није, онда проверавамо да ли је велико слово, ако није, онда проверавамо да ли је цифра, и на крају, ако није, онда пријављујемо да припада класи осталих карактера.

```
char c;
cin >> c;
if (islower(c))
    cout << "MALO SLOVO" << endl;
else if (isupper(c))
    cout << "VELIKO SLOVO" << endl;
else if (isdigit(c))
    cout << "CIFRA" << endl;
else
    cout << "OSTALO" << endl;
```

Уместо коришћења библиотечких функција могуће је имплементирати решење које се ослања на чињеницу да се карактери интерно репрезентују својим ASCII кодовима у језику C++, па се карактери могу поредити стандардним релацијским операторима ('<=', '>=', '<', '>', '==' и '!='). Поређење карактера у језику C++ је заправо поређење њихових кодова. Даље, ASCII кодови карактера имају особину да су сва мала слова и сва велика слова ASCII скупа (енглеска абечеда) поређана једно иза другог по абecedном редоследу, док су цифре поређане једна иза друге по нумеричкој вредности. То омогућава да се провера да ли је карактер мало слово изврши тако што се провери да ли је његов код између кодова карактера 'a' и 'z' тј. да ли припада интервалу од 'a' до 'z' (ако је учитани карактер `c` провера се врши условом `'a' <= c && c <= 'z'`). Слично важи и за велика слова и за цифре.

Нагласимо да се не може користити запис `'a' <= c <= 'z'`. У језику C++ он не производи грешку, али не израчунава оно што би нам овде требало (он заправо пореди вредност 0 или 1 која се добије тако што се израчуна вредност израза `'a' <= c` са кодом карактера 'z' и увек има вредност тачно).

Библиотечке функције ове провере могу извршити на ефикаснији начин (уместо поређења релацијским операторима, оне често примењују операције над бинарним записом ASCII кодова и могу да класификују карактере само на основу стања неколико битова) тако да се препоручује њихово коришћење, чиме се добија поузданији и преносивији код.

```
char c;
cin >> c;
if ('a' <= c && c <= 'z')
    cout << "MALO SLOVO" << endl;
else if ('A' <= c && c <= 'Z')
    cout << "VELIKO SLOVO" << endl;
else if ('0' <= c && c <= '9')
    cout << "CIFRA" << endl;
else
    cout << "OSTALO" << endl;
```

Задатак: Трансформација карактера

Написати програм којим се трансформише учитани карактер тако што се мало слово пребацује у велико, велико слово у мало, а остали карактери се не мењају.

Улаз: У првој линији стандардног улаза налази се један карактер.

Израз: На стандардном излазу приказати трансформисан карактер.

Пример

Улаз	Израз
t	T

Решење

Први корак у решењу задатка је класификација карактера (провера да ли је у питању мало слово, велико слово или нешто треће) и она се ради на потпуно исти начин као у задатку **Класификација карактера**, најбоље коришћењем библиотечких функција. У језику C++ се могу користити функције `islower` и `isupper`. Када се утврди да је карактер мало слово или да је велико слово треба га трансформисати.

Најбољи начин да се изврши трансформација је употреба библиотечких функција. У језику C++ и његовом заглављу `ctype` на располагању имамо функције `tolower` и `toupper`. Обе функције као једини параметар примају карактер (податак типа `char` који је заправо нумеричка променљива која чува ASCII код карактера) и враћају карактер. Ако је параметар функције `tolower` велико слово, функција враћа одговарајуће мало слово, док у супротном враћа исти карактер који јој је прослеђен. Потпуно аналогно функција `toupper` мала слова трансформише у велика, док остале карактере не мења.

```
char ch;
cin >> ch;
if (islower(ch))
    ch = toupper(ch);
else if (isupper(ch))
    ch = tolower(ch);
cout << ch << endl;
```

И у овом задатку бисмо могли имплементирати решење које не користи библиотечке функције (мада је оно мало теже и лошије). Ако знамо да се за представљање слова енглеске абеледе у језику C++ користе ASCII кодови чији је распоред направљен тако да су им вредности узастопни цели бројеви сортирани по абеледи, трансформација малог у велико слово се може извршити тако што се прво одреди које је то по реду слово у абеледи (најбоље је да бројање креће од нуле), тако што се од тог карактера тј. његовог кода одузме карактер 'a' тј. његов код, а затим се нађе велико слово које има тај редни број у абеледном реду свих великих слова, тако што се на карактер 'A' тј. његов код дода редни број карактера. Дакле, ако је променљива `c` садржи мало слово, његову трансформацију у велико слово можемо извршити изразом `'A' + (c - 'a')` тј. `c - 'a' + 'A'`. На пример, ако променљива `c` чува карактер 'd' тада је вредност `c - 'a'` једнака 4 (јер се од кода карактера 'd' који је једнак 101 одузима код карактера 'a' који је једнак 97), док је вредност израза `'A' + (c - 'a')` једнака 'D' (јер се на код карактера 'A' који је једнак 65 додаје вредност 4 и тако добија код 69 који је код карактера 'D'). Наведене бројевне вредности кодова никада не треба наводити у програмима већ је уместо тога увек боље користити карактерске константе које су потпуно еквивалентне бројевним кодовима, а чине програм много разумљивијим. Трансформација великих у мала слова вршила би се потпуно аналогно, изразом `c - 'A' + 'a'`.

Нагласимо још да је резултат примене било које аритметичке операције на два податка типа `char` податак типа `int` (ово се назива целобројна промоција и уведена је због тога што је распон типа `char` мали и постоји велика вероватноћа да дође до прекорачења опсега типа `char` приликом извођења операција). У језику C++ се доделом податка типа `int` променљивој типа `char` врши његова имплицитна конверзија.

```
char ch;
cin >> ch;
if (ch >= 'a' && ch <= 'z')
    ch = ch - 'a' + 'A';
else if (ch >= 'A' && ch <= 'Z')
    ch = ch - 'A' + 'a';
cout << ch << endl;
```

Задатак: Абецедно огледало

Катарина је одлучила да својој другарици пошаље шифровану поруку, која садржи само слова енглеске абеледе, цифре и интерпункцијске знаке. Свако слово ће шифровати посебно на основу наредних правила. Мала слова се шифрују великим словима тако што се слово `a` шифрује словом `Z`, слово `b` шифрује словом `Y`, словом `x` итд., све до слова `u` које се шифрује словом `v` и `z` које се шифрује словом `A`. Велика слова се шифрују потпуно аналогно - од `A` које се шифрује са `z` до `Z` које се шифрује са `a`. Остали карактери се не мењају.

Улаз: Са стандардног улаза уноси се једна линија текста, завршена карактером тачка (карактером `.`).

Израз: На стандардни излаз исписати шифровани текст (без карактера тачка).

Пример

```
Улаз           Излаз
Zdravo svima.  aWIZEL HERNZ
```

Решење

Први задатак је да читамо карактере један по један, док не дођемо до карактера '.'. Једна могућност била да учитамо одједном целу линију и да након тога обрађујемо један по један њен карактер, слично као у задатку [Цезаров код](#). У језику C++ се то може урадити помоћу функције помоћу `getline` и петље која пролази кроз све карактере ниске. Међутим, за тим нема потребе, јер карактере можемо читати један по један, у петљи и одмах их шифровати. Један прилично популаран начин да се таква петља организује је да се у самом услову петље и прочита карактер и провери да ли је прочитани карактер ознака краја реда. У језику C++ један начин је да се направи петља облика

```
char c;
while (cin.get(c) && c != '.')
    ...
```

У њој позивамо функцију `cin.get(c)` која учитава карактер или враћа `false` ако учитавање карактера није било могуће, тј. ако се дошло до краја улаза (види задатак [Класификација карактера](#)). Петља се завршава или када се дође до краја улаза или када је успешно учитан карактер ..

У телу петље потребно је шифровати сваки карактер. Прво вршимо класификацију на мала слова, велика слова и остале карактере, на начин који је приказан у задатку [Класификација карактера](#). Ако је карактер класификован као мало слово, вршимо његово шифровање и то тако што приметимо да ће растојање између кода карактера који се шифрује од кода карактера 'a' бити једнако растојању између карактера 'Z' и резултата. На пример, ако је у питању карактер 'c' његово растојање од карактера 'a' је два, па је резултат карактер чији се код добија када се број два одузме од кода карактера 'Z' тј. добија се карактер 'X'. Дакле, ако је с мало слово, његова шифра је 'Z' - (c - 'a'), а ако је с велико слово, његова шифра је 'z' - (c - 'A').

```
char c;
while (cin.get(c) && c != '.') {
    if (islower(c))
        cout.put('Z' - (c - 'a'));
    else if (isupper(c))
        cout.put('z' - (c - 'A'));
    else
        cout.put(c);
}
```

4.4 Набројиви тип

Поред већ наведених уграђених типова података, могуће је дефинисати и нове типове података. У језику C++ можемо дефинисати набројиве типове података, са коначно много вредности. За то се користи кључна реч `enum`. На пример, наредном конструкцијом бисмо дефинисали тип који има вредности дана у недељи.

```
enum DanUNedelji {PON, UTO, SRE, CET, PET, SUB, NED};
```

Променљиве овог типа даље декларишемо на уобичајени начин (нпр. `DanUNedelji dan;`). На вредности набројивог типа реферишемо тако што наведемо саму вредност тог типа. На пример, `SRE`. Исти ефекат би се могао постићи и дефинисањем нумеричких константи (нпр. 1 за понедељак, 2 за уторак итд.). Ако би се у програму користиле константне вредности, програм би био мање разумљив (јер се лакше разуме шта значи вредност `SRE`, него константна вредност 3). То се може решити именованим константама, међутим, остаје проблем да се променљивима типа `int` могу доделити и вредности које нису исправне (нпр. вредност 8 не означава ни један дан), док код набројивог типа тако нешто није могуће.

Задатак: Врста троугла на основу углова

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види тексты задатка.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Ако проверу углова желимо да користимо и у другим ситуацијама ван овог задатка, можемо дефинисати функцију која одређује тип троугла на основу углова. Та функција треба да врати податак о врсти троугла и то је могуће урадити било помоћу неког целобројног кода, било помоћу ниске карактера, међутим, дефинитивно најбоље решење је да се дефинише посебни набројиви тип са три вредности које представљају три врсте троуглова. У језику С++ за то се користи конструкција `enum`.

```
// pretvara ugao dat u stepenima(s) i minutima (m) u
// ugao samo u minutima
int uMinute(int s, int m) {
    return s * 60 + m;
}

enum VrstaTrougla {TUPOUGLI, PRAVOUGLI, OSTROUGLI};

VrstaTrougla vrstaTrougla(int ugao1, int ugao2, int ugao3) {
    // prav ugao u minutima
    int pravUgao = uMinute(90, 0);
    // najveci od tri ugla
    int maxUgao = max({ugao1, ugao2, ugao3});
    // ako je najveci ugao tup, trougao je tupougli
    if (maxUgao > pravUgao)
        return TUPOUGLI;
    // ako je najveci ugao prav, trougao je pravougli
    else if (maxUgao == pravUgao)
        return PRAVOUGLI;
    // u suprotnom je najveci ugao ostar, pa su svi uglovi ostri
    else
        return OSTROUGLI;
}

string VrstaTrouglaToStr(VrstaTrougla v) {
    switch(v) {
        case TUPOUGLI: return "tupougli";
        case PRAVOUGLI: return "pravougli";
        case OSTROUGLI: return "ostrougli";
    }
    return "";
}

bool ugloviTrougla(int ugao1, int ugao2, int ugao3) {
    // uglovi trougla ne smeju biti 0 i zbir im mora biti 180 stepeni
    return ugao1 != uMinute(0, 0) && ugao2 != uMinute(0, 0) &&
           ugao3 != uMinute(0, 0) &&
           ugao1 + ugao2 + ugao3 == uMinute(180, 0);
}

int main() {
    // učitavamo uglove u stepenima i minutima
    int ugao1_s, ugao1_m, ugao2_s, ugao2_m, ugao3_s, ugao3_m;
    // ...
    // pretvaramo ih u uglove date samo u minutima
    int ugao1 = uMinute(ugao1_s, ugao1_m);
    int ugao2 = uMinute(ugao2_s, ugao2_m);
    int ugao3 = uMinute(ugao3_s, ugao3_m);

    // proveravamo da li mogu biti uglovi trougla
    // i odredjujemo vrstu trougla
}
```

4.4. НАБРОЈИВИ ТИП

```
if (!ugloviTrougla(ugao1, ugao2, ugao3))
    cout << "ne" << endl;
else
    cout << VrstaTrouglaToStr(vrstaTrougla(ugao1, ugao2, ugao3))
        << endl;
return 0;
}
```

Задатак: Линеарна једначина

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текст задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Пошто решавање једначине може бити употребљиво у разним контекстима има смисла дефинисати функцију која одређује број решења и решава једначину када је решење јединствено (наравно, линеарна једначина је веома једноставна и лако је решити, међутим има смисла приказати како би се оваква функција направила, јер би се слична техника могла употребити и за дефинисање функције која решава квадратне и кубне једначине или системе линеарних једначина). Функција враћа број решења путем повратне вредности набројивог типа, а решење x преко повратног параметра. У случају да не постоји јединствено решење, променљивој x можемо доделити недефинисану вредност. У језику C++ можемо употребити константу `numeric_limits<double>::quiet_NaN()` (енгл. Not-a-Number) дефинисану у заглављу `<limits>`.

```
enum BrojResenjaLinearneJednacine {NULA, JEDAN, BESKONACNO};
const double NEDEFINISANO = numeric_limits<double>::quiet_NaN();

// funkcija koja odredjuje koliko resenja ima jednacina ax + b = 0
// postavljajuci vrednost x ako postoji jedinstveno resenje
BrojResenjaLinearneJednacine linearna_jednacina(double a, double b,
                                                double &x) {

    if (a != 0.0) {
        // a != 0: jednacina ima jedinstveno resenje
        x = -b / a;
        return JEDAN;
    } else {
        x = NEDEFINISANO;
        // jednacina je oblika 0x+b=0:
        // ima beskonacno mnogo resenja ako je b = 0,
        // nema resenja ako je b != 0
        if (b == 0.0)
            return BESKONACNO;
        else
            return NULA;
    }
}
```

```
int main() {
    double a, b; // koeficijenti jednacine
    cin >> a >> b;

    double x; // resenje jednacine
    switch (linearna_jednacina(a, b, x)) {
    case NULA:
        cout << "NEMA RESENJA" << endl;
        break;
    case JEDAN:
        cout << fixed << showpoint << setprecision(2) << x << endl;
        break;
    case BESKONACNO:
```

```
cout << "BESKONACNO RESENJA" << endl;
break;
}
```

Задатак: Школарина

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шексџи задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Положај две праве

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шексџи задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Берза

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шексџи задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Види групаџија решења овог задатака.

4.5 Структурни тип

Када желимо да неколико података третирамо као целину, након дефинисања неколико појединачних променљивих, податке можемо да групишемо у торку и доделимо једној променљивој.

На пример, опис једног такмичара чине подаци о његовом имену, разреду, школи из које долази и броју освојених поена.

Задатак: Разломци

Напиши програм који сабира и множи разломке.

Улаз: Са стандардног улаза се уносе два разломка, сваки у посебном реду. Сваки разломак је записан тако што је су записани његов бројилац и именилац, раздвојени карактером /.

Излаз: На стандардни излаз исписати у првом реду збир, а у другом производ разломака. Разломци се исписују у истом формату у ком су учитани, максимално скраћени (бројилац и именилац су узајамно прости).

Пример

Улаз	Излаз
2/3	11/12
1/4	1/6

Решење

Разломак можемо представити структуром која чува бројилац и именилац.

Дефинишемо функцију која скраћује дати разломак. Пошто се разломак мења, а у питању је структура која се подразумевано у функцију преноси по вредности, функција која скраћује разломак тај разломак прима по референци. Одређујемо НЗД бројилоца и имениоца применом Еуклидовога алгоритма и оба их скраћујемо тим бројем.

Разломке $\frac{a_1}{b_1}$ и $\frac{a_2}{b_2}$ сабирамо тако што прво формирамо разломак $\frac{a_1 b_2 + a_2 b_1}{b_1 b_2}$ и затим га скратимо. Исте разломке množимо тако што прво формирамо разломак $\frac{a_1 a_2}{b_1 b_2}$ и затим га скратимо.

```
struct Razlomak {
    int brojilac, imenilac;
};
```

```
int NZD(int a, int b) {
    while (b != 0) {
        int tmp = a % b;
```

4.5. СТРУКТУРНИ ТИП

```
    a = b;
    b = tmp;
}
return a;
}

void skrati(Razlomak &r) {
    int nzd = NZD(r.brojilac, r.imenilac);
    r.brojilac /= nzd;
    r.imenilac /= nzd;
}

Razlomak saberi(const Razlomak &a, const Razlomak &b) {
    Razlomak r;
    r.brojilac = a.brojilac*b.imenilac + a.imenilac*b.brojilac;
    r.imenilac = a.imenilac * b.imenilac;
    skrati(r);
    return r;
}

Razlomak mnozi(const Razlomak &a, const Razlomak &b) {
    Razlomak r;
    r.brojilac = a.brojilac * b.brojilac;
    r.imenilac = a.imenilac * b.imenilac;
    skrati(r);
    return r;
}

Razlomak ucitaj() {
    Razlomak r;
    char c;
    cin >> r.brojilac >> c >> r.imenilac >> ws;
    skrati(r);
    return r;
}

void ispisi(const Razlomak &r) {
    cout << r.brojilac << "/" << r.imenilac << endl;
}

int main() {
    Razlomak a = ucitaj();
    Razlomak b = ucitaj();
    ispisi(saberi(a, b));
    ispisi(mnozi(a, b));
    return 0;
}
```

Задатак: Бољи у две дисциплине

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Поређење се може реализовати помоћу библиотеке подршке за торке и њихово лексикографско поређење (које је описано у задатку [Пунолетство](#)).

Могуће је дефинисати структуру (или класу) за дефинисање поена једног такмичара и на тој структури дефинисати метод којим се врши лексикографско поређење.

```

struct Takmicar {
    // poeni takmicara iz matematike i programiranja
    int mat, prog;

    // ucitavanje podataka o takmicaru sa standardnog ulaza
    void ucitaj() {
        cin >> mat >> prog;
    }

    // ukupan broj poena takmicara
    int ukupno() const {
        return mat + prog;
    }

    // proverava da li je ovaj takmicar bolji od datog takmicara t
    bool bolji(const Takmicar& t) {
        return (ukupno() > t.ukupno()) ||
            (ukupno() == t.ukupno() && prog > t.prog);
    }
};

int main() {
    Takmicar t1;
    t1.ucitaj();
    Takmicar t2;
    t2.ucitaj();

    // redni broj pobednika
    int pobednik;
    // drugi je pobednik ako i samo ako je bolji od prvog
    if (t2.bolji(t1))
        pobednik = 2;
    else
        pobednik = 1;

    cout << pobednik << endl;
    return 0;
}

```

Види груписања решења овој задатку.

4.6 Торке

У математици се често помињу уређени парови и уређене n-торке.

Ако желимо да објединимо неколико података истог типа, тада уместо дефинисања новог структурног типа можемо употребити библиотечки тип *торке*.

У језику C++ n-торке се могу представити типом `tuple` дефинисаним у заглављу `<tuple>`. На пример, тип `tuple<int, int, int>` представља тројку целих бројева. Торке се могу иницијализовати помоћу витичастих заграда или креирати функцијом `make_tuple`. На пример, датум можемо представити торком тако да прво ставимо годину, па затим месец и на крају дан.

```

tuple<int, int, int> datum1 {2000, 1, 1};
tuple<int, int, int> datum2 = make_tuple(1999, 12, 31);

```

Поља торке се једноставно могу прочитати функцијама `get<i>` где је са `i` означен редни број компоненте торке који желимо да прочитамо. На пример, `get<0>(datum1)` враћа вредност 2000, а `get<2>(datum1)` вредност 1. Постоји и начин да се комплетна торка распакује на појединачне вредности, на пример позив: `tie(godina, mesec, dan) = datum1` би у променљиве `godina`, `mesec` и `dan` сместио вредности одговарајућих

поља торке `datum1`.

Основна предност торки у односу на структуре је то што се могу једноставно лексикографски поредити коришћењем уобичајених релацијских оператора `<`, `<=`, `>` и `>=`.

Задатак: Пунолетство

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види тексты задатка.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

У задатку се захтева поређење два датума - датума који је тачно 18 година након године рођења и задатог датума у којем се испитује пунолетство. Ако се датуми представе уређеним тројкама облика (g, m, d) тада се поређење датума своди на лексикографско поређење ових уређених тројки. Постоје различити начини да се то уради.

Торке и библиотечко лексикографско поређење

Могуће је искористити библиотечку подршку за рад са n -торкама и њихово поређење које се по правилу врши лексикографски. Ако датуме представимо тројкама бројева у којима је на првом месту година, на другом месец, а на трећем дан, тада лексикографски поредак ових тројки одговара уобичајеном поретку датума. У језику C++ торке се представљају типом `tuple` и лексикографски се могу поредити коришћењем релацијских оператора `<`, `>`, `<=`, `>=`, `==` и `!=`.

```
// ...
if (make_tuple(g2, m2, d2) >= make_tuple(g1 + 18, m1, d1))
    cout << "DA" << endl;
else
    cout << "NE" << endl;

return 0;
}
```

Задатак: Бољи у две дисциплине

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види тексты задатка.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Сваки такмичар је представљен уређеним паром бројева (поенима из математике и програмирања) и потребно је одредити већи од два уређена пара, при чему је релација поретка међу тим паровима одређена условима задатка (пореди се прво укупан број поена, па онда поени из програмирања). Ово је класичан пример релације лексикографског поређења.

Вежбе ради, ту релацију можемо имплементирати и сами, у засебној функцији која прихвата податке за два такмичара (четири броја) и враћа истинитосну (буловску) вредност `true` ако је први бољи од другог у овој релацији. Различити могући начини имплементације лексикографског поређења описани су у задатку [Пунолетство](#).

Један начин је да поређење реализујемо у функцији која проверава један по један услов и ако открије да је на основу неког услова могуће вратити резултат, помоћу `return` тај резултат враћа.

```
// Provera da li je takmicar A bolji od takmicara B
bool bolji(int matA, int progA, int matB, int progB) {
    int ukupnoA = matA + progA;
    int ukupnoB = matB + progB;
    if (ukupnoA > ukupnoB)
        return true;
    if (ukupnoA < ukupnoB)
        return false;
    return progA > progB;
}
```

```

}

int main() {
    // broj poena prvog takmicara iz matematike i programiranja
    int mat1, prog1;
    cin >> mat1 >> prog1;

    // broj poena drugog takmicara
    int mat2, prog2;
    cin >> mat2 >> prog2;

    // Redni broj pobednika
    int pobednik;

    // drugi je pobednik ako i samo ako je bolji od prvog
    if (bolji(mat2, prog2, mat1, prog1))
        pobednik = 2;
    else
        pobednik = 1;

    cout << pobednik << endl;
    return 0;
}

```

Поређење се може реализовати и помоћу сложеног логичког израза.

```

// Provera da li je takmicar A bolji od takmicara B
bool bolji(int matA, int progA, int matB, int progB) {
    int ukupnoA = matA + progA;
    int ukupnoB = matB + progB;
    return (ukupnoA > ukupnoB) ||
           (ukupnoA == ukupnoB && progA > progB);
}

```

Задатак: Најбољи у две дисциплине

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Поређани датуми

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Види груичија решења овог задатка.

Глава 5

Низови, ниске, матрице

У многим задацима потребно истовремено у меморији запамтити целу серију података, да би се затим на тим подацима могле вршити различите обраде. За то се користе различите врсте *низова* (класични низови, вектори, листе, матрице итд.).

5.1 Једнодимензионалне колекције података

5.1.1 Елементи програмског језика

Најједноставнија колекција која се може користити за истовремено складиштење целе серије података је *класичан низ*.

5.1.1.1 Статички алоцирани низови

У језику С++ величина класичних статичких низова мора бити позната у фази превођења програма и зато је потребно унапред знати њихов максимални број елемената. У такмичарским задацима ово је обично саставни део формулације задатка, међутим, у реалној програмерској пракси оваква ограничења често или нису позната или су доста већа од броја елемената које програм обично користи тако да се меморија непотребно заузима. Статички низови у језику С++ се обично користе на следећи начин.

```
const int MAX = 50000;
int a[MAX];
int n;
cin >> n;
for (int i = 0; i < n; i++)
    cin >> a[i];
...
```

Приметимо да је MAX константа која је позната у време превођења програма.

Постоји значајна разлика између статички алоцираних низова који су декларисани унутар функција (локално) и оних декларисаних ван свих функција (глобално). Глобални низови обично могу бити знатно већи и вредности свих елемената су иницијално постављене на нулу (код локалних низова иницијалне вредности елемената нису дефинисане).

Димензија статички алоцираног низа (максимални број елемената) често је већа од стварног броја елемената уписаних у низ. Стога је увек потребно у посебној променљивој памтити колико елемената низ стварно садржи (у наведеном примеру то је била променљива n). Приликом преноса низа у функцију, потребно је уз низ проследити и ту променљиву.

Функције којима су низови параметри примају само меморијску адресу почетка низа. Све модификације низа унутар функције врше се над оригиналним низом (оним који је наведен као аргумент, приликом позива). Стога су низови увек улазно-излазни параметри функција (за разлику од, на пример, података типа int или структура који су улазни параметри, осим ако се не проследи по референци). Статички низови који су дефинисани у функцијама се ослобађају по завршетку извршавања тела функције и није их могуће вратити као

резултујућу вредност функције (ако функција треба да врати статички низ, њега је неопходно проследити као параметар функције и функција може само да попуни његове елементе).

5.1.1.2 VLA

Неки компилатори подржавају облик низова који се назива VLA (engl. variable length array), где се као димензија низа наводи променљива чија је вредност позната тек у фази извршавања програма.

```
int n;
cin >> n;
int a[n];
for (int i = 0; i < n; i++)
    cin >> a[i];
...
```

VLA никада нису били део стандарда језика C++ (једно време били су део стандарда језика C, па су након тога избачени) и стога их никако није препоручљиво користити.

5.1.1.3 Динамички алоцирани низови

Алокација низова у језику C++ у тренутку извршавања програма (тзв. динамичка алокација) је могућа (оператором `new`), али захтева рад са показивачима и ручно ослобађање меморије (оператором `delete []`) и сматра се прилично напредном темом, тако да ту врсту низова нећемо користити.

```
int n;
cin >> n;
int *a = new int[n];
for (int i = 0; i < n; i++)
    cin >> a[i];
...
delete[] a;
```

У језику C++ за динамичку алокацију низова могуће је користити и библиотеку за рад са меморијом језика C и функције `malloc` и `free`, међутим и тај облик рада са низовима је непримерен језику C++ и препоручујемо да се у њему не користи.

```
int n;
cin >> n;
int *a = (int*)malloc(n * sizeof(int));
for (int i = 0; i < n; i++)
    cin >> a[i];
...
free(a);
```

5.1.1.4 Библиотечке колекције (`vector`, `array`)

Најбоље решење долази у форми динамички алоцираних библиотечких колекција тј. колекција за чије се складиштење меморија заузима тек током извршавања (а не током превођења) програма и које програмеру пружају веома удобан интерфејс за рад са динамички алоцираном меморијом. У језику C++ најчешће коришћена динамичка колекција, која одмењује класичне, статички алоциране низове је *вектор* (`vector` из заглавља `<vector>`).

- Вектор је параметризован типом елемената које чува тако да се користе `vector<int>`, `vector<double>`, `vector<string>`, али и, на пример, `vector<vector<int>>`.
- Ако је број елемената вектора `n` познат пре његове декларације (а у току извршавања програма), тада је вектор најбоље декларисати уз навођење димензије у склопу декларације (на пример, `vector<double> a(n)`). Могуће је навести и почетну вредност свих елемената (на пример, декларацијом `vector<double> a(n, 1.0)` гради се вектор од `n` елемената постављених на вредност `1.0`).
- Након декларације у којој је наведен број елемената, вектор се користи на потпуно исти начин као и низ (елементу на позицији `i` приступа се са `a[i]`). Димензију (број елемената) вектора `a` можемо одредити помоћу методе `a.size()` (*метода* је посебан облик функције која се позива над неким објектом тако

што се име тог објекта наводи испред имена функције и тачкице и која нам даје неке информације о том објекту или проузрокује одређене измене тог објекта).

- Једна од најзначајнијих предности вектора у односу на класичне низове је то што им се димензија може мењати током извршавања програма. Проширивање вектора `a` додавањем елемента `x` на његов крај могуће је уз помоћ позива методе `a.push_back(x)` (тима се увећава број елемената за 1).
- Промена димензије вектора се може урадити методом `resize` - након позива `a.resize(n)` вектор `a` има димензију `n` (под условом да је било довољно меморије).
- Простор за смештање елемената у вектор можемо резервисати и без промене његове димензије коришћењем методе `reserve`. Након позива `a.reserve(n)` резервише се простор за смештање `n` елемената, али величина вектора остаје непромењена (у почетку 0), све док се елементи не додају помоћу `push_back`.

Најчешћи облик коришћења вектора у овој збирци је следећи.

```
int n;
cin >> n;
vector<int> a(n);
for (int i = 0; i < n; i++)
    cin >> a[i];
...
```

Ако број елемената није задат у посебној променљивој, могуће је прочитати га методом `size`.

```
for (int i = 0; i < a.size(); i++)
    // obradjuje se a[i]
```

Савремени С++ (почевши од стандарда С++-11) допушта и краћи облик петље којом се обрађује један по један елемент вектора (али и низа и других библиотечких колекција). У многим језицима оваква петља назива се петља `foreach`.

```
for (int x : a)
    ...
```

Вектори се у функције преносе по вредности (осим ако се експлицитно не наведе да се преносе по референци), што значи да се у функцију преноси копија вектора који је наведен као аргумент у позиву. Ако желимо да функција модификује садржај вектора, неопходно га је пренети по референци (навођењем симбола `&`). Тада се у функцију шаље само меморијска адреса тог вектора и све операције се спроводе над оригиналним вектором (не врши се копирање). Копирањем се непотребно троши меморија и време, па има смисла векторе проследити по референци и функцијама које само анализирају њихов садржај и не мењају их. Тада се обично наводи кључна реч `const` чиме се обезбеђује да се вектор не може променити у функцији иако је пренет по референци. На пример,

```
int zbir(const vector<int>& a) {
    int z = 0;
    for (int x : a)
        z += a;
    return z;
}
```

Функција може да врати вектор као своју резултујућу вредност. Захваљујући тзв. семантици премештања (енгл. *move semantics*) уведеној у савременим верзијама језика С++ не врши се никакво копирање садржаја, па се овим не умањују перформансе програма.

Рецимо и да у језику С++, од стандарда С++-11 постоји колекција `аггау` која је веома слична класичном статичком низу (фиксне је величине и та величина се наводи у оквиру типа), међутим, нуди одређене методе и глобалне функције које омогућавају да се добије колекција која има ефикасност класичних низова, а удобност коришћења вектора (на пример, величину увек можемо сазнати методом `size`, док код обичних низова пренетих у функцију немамо могућност да сазнамо величину, већ је морамо прослеђивати као параметар функције). Ипак, у овој збирци `аггау` нећемо користити.

```
const int MAX = 100;
array<int, MAX> a;
for (int i = 0; i < n; i++)
```

```
cin >> a[i];
...
```

5.1.2 Елементарно коришћење низова

Задатак: Испис у обратном редоследу

Написати програм којим се уноси n целих бројева, а затим се унети бројеви приказују у обратном редоследу од редоследа уношења.

Улаз: У првој линији стандардног улаза налази се природан број n ($1 \leq n \leq 1000$). У следећих n линија налазе се цели бројеви између -1000 и 1000 .

Излаз: На стандардном излазу приказати унете бројеве, сваки у посебној линији, у обратном редоследу од редоследа уношења.

Пример

Улаз	Излаз
5	987
10	14
-123	67
67	-123
14	10
987	

Решење

Да бисмо могли да испишемо учитане елементе у обратном редоследу, потребно је да их након учитавања све складиштмо у неку колекцију, коју ћемо затим исписати у обратном редоследу. Пошто се унапред зна максимални број елемената (у тексту задатка је наведено да неће бити више од 1000 елемената), у језику C++ се може употребити класичан статички алоцирани низ.

Елементе учитавамо у петљи у којој се бројач i креће од 0 па до $n - 1$ (где је n број елемената низа), увећавајући се у сваком кораку за 1. Елементе исписујемо у петљи која се креће уназад тј. у петљи у којој бројач креће од $n - 1$ и у сваком кораку се умањује за 1 док не дође до вредности 0.

```
// maksimalni broj elemenata niza
const int MAX = 1000;

// broj elemenata koji se ucitavaju
int n;
cin >> n;

// ucitavamo elemente u niz
int a[MAX];
for (int i = 0; i < n; i++)
    cin >> a[i];

// ispisujemo elemente niza unazad
for (int i = n - 1; i >= 0; i--)
    cout << a[i] << endl;
```

Задатак: Линије у обратном редоследу

Напиши програм који исписује све линије које се читају са стандардног улаза у обратном редоследу од редоследа учитавања.

Улаз: Са стандардног улаза се читавају линије текста, све до краја улаза.

Излаз: На стандардни излаз исписати учитане линије у обратном редоследу.

Пример

Улаз	Издаз
zdravo	dan
svete	dobar
dobar	svete
dan	zdravo

Решење

Пошто не знамо унапред број линија, за смештање линија морамо употребити колекцију која допушта проширивање додавањем елемената на крај. У језику С++ можемо употребити `vector`, коме ћемо додавати елементе методом `push_back`, а затим их исписати од позиције `size() - 1`, до 0.

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int main() {
    vector<string> a;
    string linija;
    while (cin >> linija)
        a.push_back(linija);
    for (int i = a.size() - 1; i >= 0; i--)
        cout << a[i] << endl;

    return 0;
}
```

Задатак: Најнижа температура

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексат задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Решење уз коришћење низа

Ако претпоставимо да су сви елементи учитани у низ (што, као што смо видели, није потребно за решење овог задатка), можемо једноставно употребити библиотечку функцију за одређивање минимума. Са друге стране, можемо и сами имплементирати алгоритам одређивања минимума (ово решење наводимо само да бисмо илустровали да се алгоритми обраде серија елемената у практично неизмењеном облику могу применити и над елементима низа). Пошто је максимални број елемената унапред познат и веома је мали (највише 50), можемо употребити класични низ (наравно, решење не би било лошије ни да смо употребили `vector` или `array`).

```
// broj gradova
int n;
cin >> n;
// učitavamo temperature u svim gradovima
int temperature[50];
for (int i = 0; i < n; i++)
    cin >> temperature[i];

// algoritam odredjivanja minimuma
int min = temperature[0];
for (int i = 1; i < n; i++)
    if (temperature[i] < min)
        min = temperature[i];
```

```
// ispisujemo minimum
cout << min << endl;
```

Учитавање и одређивање минимума можемо реализовати и у засебним функцијама.

```
void ucitajNiz(int a[], int& n) {
    cin >> n;
    for (int i = 0; i < n; i++)
        cin >> a[i];
}

int minimum(int a[], int n) {
    int min = a[0];
    for (int i = 1; i < n; i++)
        if (a[i] < min)
            min = a[i];
    return min;
}

int main() {
    // broj gradova
    int n;
    // temperature u svim gradovima
    int temperature[50];
    ucitajNiz(temperature, n);
    // ispisujemo minimum
    cout << minimum(temperature, n) << endl;
    return 0;
}
```

Види груписања решења овој задајци.

Задатак: Средине

Овај задајак је поновљен у циљу увежбавања различитих техника решавања. Види текст задајка.

Покушај да задајак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Решење уз коришћење низова

Задатак можемо решити и уз помоћ низова (мада, као што смо видели, то није неопходно). Главна предност таквог решења је то што елементе у низу можемо једноставно обрађивати коришћењем библиотечких функција. Чак иако не користимо библиотечке функције, израчунавање средина можемо имплементирати у засебним функцијама, што олакшава фокусирање на једноставније потпроблеме и чиме програмски код који напишемо постаје употребљив и у другим програмима. Елементе ћемо учитавати слично као у задатку **Читање до краја улаза**. Пошто не знамо унапред колико ће елемената бити на улазу погодно је да елементе један по један додајемо у колекцију. У језику С++ можемо користити вектор (`vector`) у који елементе додајемо методом `push_back`.

Алгоритми којима се средине израчунавају исти су као и у случају обраде појединачних елемената серије без њиховог смештања у низ (на пример, збир елемената серије се израчунава тако што се збир иницијализује на нулу, а затим се у петљи увећава за један по један елемент серије, једино што се сада елементи серије читају из низа уместо са стандардног улаза).

```
double aritmetickaSredina(const vector<double>& a) {
    // zbir elemenata niza
    double zbir = 0.0;
    for (double x : a)
        zbir += x;
    // broj elemenata niza
    int broj = a.size();
```

```

    return zbir / broj;
}

double harmonijskaSredina(const vector<double>& a) {
    // zbir reciprocnih vrednosti elemenata niza
    double zbirReciprocnih = 0.0;
    for (double x : a)
        zbirReciprocnih += 1.0 / x;
    // broj elemenata niza
    int broj = a.size();
    return broj / zbirReciprocnih;
}

int main() {
    // učitavamo elemente niza
    vector<double> a;
    double x;
    while (cin >> x)
        a.push_back(x);

    // ispisujemo njihovu aritmeticku i harmonijsku sredinu
    cout << fixed << showpoint << setprecision(2)
         << aritmetickaSredina(a) << endl
         << harmonijskaSredina(a) << endl;
    return 0;
}

```

Види групација решења овој задатку.

Задатак: Минимално одступање од просека

За дати низ од n реалних бројева одредити вредност најмањег апсолутног одступања од просека вредности унетих бројева (то је најмања апсолутна вредност разлика између елемената и просека низа).

Улаз: У првој линији стандардног улаза уноси се број елемената низа n ($1 \leq n \leq 100$), а затим у следећих n линија елементи низа $-10000 \leq a_i \leq 10000$.

Излаз: Реалан број на две децимале који представља вредност најмањег апсолутног одступања од средње вредности.

Пример

Улаз	Излаз
6	0.78
2.8	
19.3	
-4.2	
7.5	
-11.1	
7.17	

Решење

Просечну вредност учитаних бројева већ смо рачунали у неколико задатака (на пример, у задацима **Средине** и **Просек одличних**). Просечну вредност не можемо одредити док не учитамо све бројеве. Међутим, када израчунамо ову вредност потребно је поново обрадити све бројеве, тј. израчунати њихово растојање до просека. Пошто смо све бројеве учили током израчунавања просека, не можемо их поново читати да бисмо рачунали растојања. Зато је неопходно све учитане бројеве упамтити у програму и неопходно је да употребимо неки облик низа.

Када се бројеви учитају у меморију, тада је једноставно израчунати им просек p (као количник збира који рачунамо класично, као у задатку **Збир n бројева** и броја елемената) и израчунати минимално одступање као

минимум низа $|a_i - p|$ за $i = 0, 1, \dots, n-1$ (користимо алгоритам минимума преликаних вредности на исти начин као у задатку **Најближи датом целом броју**).

Пошто број елемената може варирати, а познат је пре њиховог учитавања употребићемо вектор који ћемо пре учитавања елемената алоцирати на њихов број n (који се учитава пре алоцирања вектора).

```
// broj elemenata niza
int n;
cin >> n;

// ucitavamo sve elemente u niz
vector<double> a(n);
for (int i = 0; i < n; i++)
    cin >> a[i];

// racunamo prosek kao kolicnik zbira i broja elemenata
double zbir = 0;
for (int i = 0; i < n; i++)
    zbir += a[i];
double prosek = zbir / n;

// odredjujemo najmanje rastojanje (najmanju apsolutnu vrednost razlike)
// minimum inicijalizujemo na apsolutnu vrednost razlike prvog elementa
double minRastojanje = abs(a[0] - prosek);
for (int i = 1; i < n; i++)
    // azuriramo minimum analizom i-tog elementa
    minRastojanje = min(minRastojanje, abs(a[i] - prosek));

// ispisujemo konacnu vrednost minimuma
cout << fixed << showpoint << setprecision(2) << minRastojanje << endl;
```

Види групачија решења овог задатка.

Задатак: Ниска палиндром

Написати програм којим се проверава да ли је дата реч *палиндром*, што значи да се једнако чита слево на десно и десно на лево.

Улаз: Прва и једина линија стандардног улаза садржи реч која се састоји само од малих слова енглеске абетеде.

Излаз: На стандардном излазу приказати реч да ако реч представља палиндром, тј. не у супротном.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
madam	da	ganac	ne

Решење

Линеарна претрага за неодговарајућим карактером

Задатак се може решити тако што се пореде парови карактера из дате ниске и то први карактер и последњи, други и претпоследњи и тако даље. Иако се у овом задатку ради са ниском карактера, а не са низом бројева, ниске можемо посматрати као низове (јер елементе на датим позицијама читамо на потпуно исти начин као да су у питању низови).

Један начин обиласка парова је помоћу две променљиве i и j које представљају позиције карактера који се упоређују. Индекс i иницијализујемо на вредност 0 (почетак ниске), а индекс j иницијализујемо на вредност $n-1$ (крај ниске), а затим мењамо индексе тако да се i (индекс првог елемента у пару) увећава за 1 у сваком пролазу, а j (индекс другог елемента) смањује за 1, све док важи да је $i < j$.

Провера се заснива на алгоритму линеарне претраге (приказаном у задатку **Негативан број**) и међу паровима се тражи да ли постоји неки у којем су карактери различити. Претрагу је најједноставније имплементирати у склопу функције чија је основа петља која пролази кроз све парове карактера које треба упоредити. Први пут

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

када наиђемо на различите карактере, провера се прекида и функција може да врати `false`, док вредност `true` враћамо на крају функције, након петље којој су проверени сви парови и у којој је утврђено да не постоји различит пар карактера.

```
bool palindrom(string s) {
    for (int i = 0, j = s.size() - 1; i < j; i++, j--)
        if (s[i] != s[j])
            return false;
    return true;
}
```

Исти избор парова за поређење у оквиру ниске (са почетка и краја ниске) се може извршити и уз помоћ само једног индекса i , с тим што се онда у петљи пореде елементи низа са позиција i и $n - 1 - i$, а i се увећава док год је строго мање од вредности $\lfloor \frac{n}{2} \rfloor$ тј. док је $i < n/2$.

```
bool palindrom(string s) {
    int n = s.size();
    for (int i = 0; i < n/2; i++)
        if (s[i] != s[n-1-i])
            return false;
    return true;
}
```

Види групаџија решења овој задајџка.

Задатак: Погођена комбинаџија

Сеф се отвара тако што се унесе комбинаџија цифара. Напиши програм који проверава да ли је унета комбинаџија тачна и да ли сеф треба да се отвори.

Улаз: Са стандардног улаза се учитава број n ($1 \leq n \leq 100$) а затим и тачна комбинаџија од n цифара (све цифре су задате у једном реду и раздвојене су размаџима). Након тога се уноси комбинаџија од n цифара коју је корисник сефа унео (свака цифра задата је у посебном реду).

Изаз: На стандардни излаз исписати да ако је комбинаџија тачно погођена, тј. не у супротном.

Пример

Улаз	Изаз
3	da
1 2 3	
1	
2	
3	

Решење

Ако не користимо векторе, већ класичне низове, тада поређење није могуће коришћењем оператора `==`.

Ако елементе учитамо у два класична низа, можемо и сами дефинисати функцију која врши проверу њихове једнакости.

Проверу једнакости можемо да реализујемо алгоритмом линеарне претраге (који смо приказали у задатку **Негативан број**) тако што проверавамо да ли постоји позиџија на којој се у два низа налази различит елемент.

```
// provera da li su dva niza jednaka
bool jednaki(int a[], int b[], int n) {
    for (int i = 0; i < n; i++)
        if (a[i] != b[i])
            return false;
    return true;
}
```

Користећи ову идеју, довољно је да чувамо само први низ, а елементе другог обрађујемо док их учитавамо.

```
// ucitavamo jedan po jedan element druge kombinacije i proveravamo
```



```
// da li se razlikuje od odgovarajuceg elementa prve kombinacije
bool jednaki = true;
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    if (a[i] != x) {
        jednaki = false;
        break;
    }
}

// ispisujemo rezultat
cout << (jednaki ? "da" : "ne") << endl;
```

Види груписања решења овој задајци.

Задатак: Парни и непарни елементи

Поштар иде улицом и треба да распореди пошиљке. Одлучио је да прво обиђе једну страну улице (на којој су куће са парним бројевима), а да у повратку обиђе другу страну улице (на којој су куће са непарним бројевима). Напиши програм који за унете бројеве кућа одређује које су на парној, а које на непарној страни улице.

Улаз: Са стандардног улаза се уноси број пошиљки (природан број k мањи од 100), а затим и k природних бројева који представљају бројеве кућа на пошиљкама. Сваки број је задат у посебном реду.

Излаз: На стандардни излаз исписати два реда бројева. У првом реду се налазе сви парни бројеви (исписани у истом редоследу у којем су унети), а у другом сви непарни бројеви (исписани у истом редоследу у којем су унети). Бројеви у сваком реду треба да су раздвојени по једним размаком, а размак може бити наведен и иза последњег броја.

Пример

Улаз	Излаз
5	2 4
1	1 3 5
2	
3	
4	
5	

Решење

Филтрирање у два низа

У програму ћемо чувати две колекције (низа) - једну у којој ћемо чувати парне, а другу у којој ћемо чувати непарне елементе. Учитавамо елемент по елемент, проверавамо да ли је паран или непаран тако што израчунамо остатак при дељењу са 2 (технику провере дељивости израчунавањем остатка видели смо у задатку [Збир година браће и сестре](#)) и у зависности од тога додајемо га у одговарајући низ. Приметимо да се и у овом задатку врши класификација (филтрирање) на основу својстава учитаних бројева (слично као, на пример, у задатку), међутим, пошто је потребно исписати прво парне, па онда непарне бројеве, пре исписа је неопходно бројеве памтити у низу.

Низови могу бити алоцирани на основу максималног броја елемената (величина им може бити 100 елемената, што је број наведен у тексту задатка). Тада је поред низова потребно користити и променљиве које чувају њихов тренутни број попуњених елемената (које се иницијализују на нулу). То је уједно и позиција прве слободне позиције у низу тако да се додавање елемената у низ реализује тако што се елемент упише на ту позицију, а затим се она увећа за 1. Ове две ствари је могуће остварити и у једној наредби (на пример, $a[n++] = x$).

```
// maksimalan broj elemenata dat tekstem zadatka
const int MAX = 100;

int main() {
    // niz parnih i niz neparnih brojeva i njihove trenutne duzine
```

```
int parni[MAX], neparni[MAX];
int p = 0, n = 0;

// učitavamo broj elemenata
int k;
cin >> k;
// učitavamo k elemenata
for (int i = 0; i < k; i++) {
    // učitavamo naredni element
    int x;
    cin >> x;
    if (x % 2 == 0)    // ako je paran
        parni[p++] = x;    // smestamo ga u niz parnih
    else              // u suprotnom
        neparni[n++] = x; // smestamo ga u niz neparnih
}

// ...
```

Дефинисање помоћних функција

Издвајање елемената можемо реализовати и помоћним функцијама.

Када се ради са класичним низовима до функције стиже само адреса њиховог почетка тако да свака измена садржаја представља измену оригиналног низа (не врши се пренос по вредности тј. не прави се копија). Функција која издваја парне елементе прима оригинални низ, број његових попуњених елемената (тај број може бити мањи од димензије низа тј. од резервисаног меморијског простора), низ у који ће сместити парне елементе, а потребно је да врати број попуњених елемената тог низа (број парних елемената оригиналног низа), што може да уради било преко повратне вредности, било преко параметра пренетог по референци.

```
// maksimalan broj elemenata dat tekstem zadatka
const int MAX = 100;

// učitavamo elemente u niz a i vracamo njihov broj kroz promenljivu k
void učitajNiz(int a[], int& k) {
    // učitavamo niz od k elemenata
    cin >> k;
    for (int i = 0; i < k; i++)
        cin >> a[i];
}

// izdvajamo parne elemente iz niza a duzine k
// i smestamo ih u niz parni duzine p
void izdvojParne(int a[], int k, int parni[], int& p) {
    p = 0;
    for (int i = 0; i < k; i++)
        if (a[i] % 2 == 0)
            parni[p++] = a[i];
}

// izdvajamo neparne elemente iz niza a duzine k
// i smestamo ih u niz neparni duzine n
void izdvojNeparne(int a[], int k, int neparni[], int& n) {
    n = 0;
    for (int i = 0; i < k; i++)
        if (a[i] % 2 != 0)
            neparni[n++] = a[i];
}

// ispisujemo elemente niza a duzine k na standardni izlaz
```

```

void ispisiNiz(int a[], int k) {
    for (int i = 0; i < k; i++)
        cout << a[i] << " ";
    cout << endl;
}

int main() {
    // učitavamo brojeve u niz
    int a[MAX], k;
    ucitajNiz(a, k);

    // niz parnih i niz neparnih brojeva i njihove trenutne duzine
    int parni[MAX], neparni[MAX];
    int p, n;
    // izdvajamo parne i neparne elemente
    izdvojParne(a, k, parni, p);
    izdvojNeparne(a, k, neparni, n);

    // ispisujemo niz ucitanih parnih brojeva
    ispisiNiz(parni, p);
    // ispisujemo niz ucitanih neparnih brojeva
    ispisiNiz(neparni, n);
    return 0;
}

```

Коришћење вектора

У језику С++ уместо класичних, статички алоцираних низова могуће је користити и векторе (колекцију `vector` декларисану у заглављу `<vector>`), чија је основна предност то што нови елемент можемо лако додати на крај листе методом `push_back` (при чему се она аутоматски проширује).

Ово решење је боље што се тиче коришћења меморије, јер се не зна унапред колико ће тачно бити парних, а колико непарних елемената, али може бити малчице временски неефикасније (јер се време губи на реалокације и повећање димензије током додавања нових елемената на крај).

Када се дефинишу функције које из вектора издвајају парне тј. непарне елементе, вектор који садржи оригиналне елементе је потребно проследити по референци, да се не би копирао цео његов садржај (пожељно је и да се употреби модификатор `const` да би се нагласило да се приликом филтрирања његов садржај не мења). Резултат филтрирања је могуће вратити као повратну вредност (захваљујући семантици померања – *move semantics*, уведеној у новијим стандардима језика неће бити копирања садржаја вектора приликом његовог враћања из функције преко повратне вредности). Друга могућност је да се резултујући вектор прослеђује као други параметар функције, по референци, али без модификатора `const` (јер се његов садржај може променити). Пошто унапред не знамо колико ће елемената садржати резултујући вектор, употребљавамо методу `push_back` која динамички проширује димензије резултата када је то потребно.

```

// učitavamo vektor celih brojeva sa standardnog ulaza
vector<int> ucitajVektor() {
    int k;
    cin >> k;
    vector<int> a(k);
    for (int i = 0; i < k; i++)
        cin >> a[i];
    return a;
}

// izdvajamo parne elemente vektora u novi vektor
vector<int> izdvojParne(const vector<int>& a) {
    vector<int> parni;
    for (int x : a)
        if (x % 2 == 0)
            parni.push_back(x);
}

```

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

```
    return parni;
}

// ...
int main() {
    // učitavamo brojeve u niz
    vector<int> a = učitajVektor();

    // izdvajamo elemente u niz parnih i niz neparnih brojeva
    vector<int> parni = izdvojParne(a);
    vector<int> neparni = izdvojNeparne(a);

    // ispisujemo niz učitanih parnih brojeva
    ispišiVektor(parni);
    // ispisujemo niz učitanih neparnih brojeva
    ispišiVektor(neparni);

    return 0;
}
```

Види групачија решења овој задатка.

Задатак: Редни број максимума

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тегскај задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Види групачија решења овој задатка.

Задатак: Разлика сума до мах и од мах

Уноси се масе предмета, одредити разлику суме маса предмета до првог појављивања предмета највеће масе и суме маса предмета после првог појављивања предмета највеће масе (предмет највеће масе није укључен ни у једну суму).

Улаз: У првој линији стандардног улаза налази се број предмета n ($1 \leq n \leq 50000$). Свака од наредних n линија садржи по један природан број из интервала $[1, 50000]$, ти бројеви представљају масе сваког од n предмета.

Израз: У првој линији стандардног израза приказати тражену разлику маса.

Пример

Улаз	Израз
5	-14
10	
13	
7	
13	
4	

Објашњење

Предмет највеће масе је 13. Збир маса пре његовог првог појављивања је 10, а после његовог првог појављивања је 24. Зато је тражена разлика -14.

Види групачија решења овој задатка.

Задатак: Просечно одступање од минималног

Дате су цене уређаја у n продавница. Написати програм којим се одређује колико су у просеку цене уређаја скупље од најмање цене уређаја у тим продавницама.

Улаз: У првој линији стандардног улаза налази се природан број n ($1 \leq n \leq 200$). У следећих n линија налазе се позитивни реални бројеви који представљају цене уређаја у продавницама.

Изназ: На стандардном излазу приказати на две децимале заокружено просечно одступање цена од минималне цене.

Пример

Улаз	Изназ
4	7.50
100	
95	
120	
95	

Задатак: Максимална разлика суседних

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види шексџи задатак.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Види групаџија решења овог задатак.

Задатак: Палиндромска реченица

Написати програм којим се проверава да ли је дата реченица палиндром. Реченица је палиндром ако се једнако чита слево на десно и десно на лево, при чему се разматрају само слова и не прави се разлика између великих и малих слова.

Улаз: Прва и једина линија стандардног улаза садржи реченицу (састављену од слова, празнина и интерпункцијских знакова).

Изназ: На стандардном излазу приказати реч да ако реченица представља палиндром иначе приказати реч не.

Пример

Улаз	Изназ
Ana voli Milovana!!!	da

Задатак: Провера монотоности

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види шексџи задатак.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Поређани датуми

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види шексџи задатак.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Прерачунавање миља у километре

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види шексџи задатак.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Види групаџија решења овог задатак.

5.1.3 Трансформације низова

Задатак: Транслација тачака

Дате су координате n тачака у равни. Транслирати тачке тако да им тежиште буде у координатном почетку.

Улаз: У првој линији стандардног улаза налази се природан број n ($1 \leq n \leq 100$). У следећих n линија налазе се по два реална броја, који представљају x и y координате тачака.

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

Издаз: На стандардном излазу приказати координате тачака после транслације, за сваку тачку у једној линији њену x па y координату, координате одвојити једном празнином и приказати их на две децимале.

Пример

Улаз	Издаз
3	-1.00 -1.00
0 0	0.00 -1.00
1 0	1.00 2.00
2 3	

Решење

Тежиште T тачака (x_i, y_i) за i од 1 до n има координате $\left(\frac{\sum_{i=1}^n x_i}{n}, \frac{\sum_{i=1}^n y_i}{n}\right)$. Транслагацијом тачке (x_i, y_i) за вектор (a, b) добија се тачка $(x_i + a, y_i + b)$. Тежиште транслагираних тачака је тачка чије су координате $\left(\frac{\sum_{i=1}^n (x_i + a)}{n}, \frac{\sum_{i=1}^n (y_i + b)}{n}\right)$, што је једнако $\left(\frac{\sum_{i=1}^n x_i}{n} + a, \frac{\sum_{i=1}^n y_i}{n} + b\right)$. Тежиште после транслагације треба да буде координатни почетак $(0, 0)$. Према томе, потребно је да важи $a = -\frac{\sum_{i=1}^n x_i}{n}$ и $b = -\frac{\sum_{i=1}^n y_i}{n}$.

Дакле, одредимо координате тежишта $T(x_T, y_T)$ датих n тачака, као аритметичку средину x односно y координата датих тачака, тако да је $x_T = \frac{\sum_{i=1}^n x_i}{n}$, $y_T = \frac{\sum_{i=1}^n y_i}{n}$. Затим дате тачке транслагирамо за вектор $(-x_T, -y_T)$.

Координате тачака можемо сместити у два независна низа исте дужине (низ координата x и низ координата y).

Одређивање аритметичке средине (просека) елемената низа можемо урадити применом алгоритма одређивања аритметичке средине серије бројева (види задатак [Средине](#)), као количник збира и броја елемената серије. Транслагацију можемо такође једноставно урадити трансформацијом оба низа координата у петљи (вршимо пресликавање серије, при чему се резултат смешта на место оригинала).

```
// ...
// izracunavamo teziste, kao aritmeticku sredinu koordinata
double xT = 0, yT = 0;
for (int i = 0; i < n; i++) {
    xT += x[i];
    yT += y[i];
}
xT /= n;
yT /= n;

// transliramo tacke umanjucuci im koordinate za kordinate tezista
for (int i = 0; i < n; i++) {
    x[i] -= xT;
    y[i] -= yT;
}

// ispisujemo rezultat
for (int i = 0; i < n; i++)
    cout << setprecision(2) << showpoint << fixed
         << x[i] << " " << y[i] << endl;
```

Учитавање низова координата, њихов испис, одређивање тежишта и транслагацију можемо издвојити у засебне функције. Ако користимо класичне низове, тада функција учитавања мора да врати број елемената читаног низа, што можемо извести прослеђивањем те димензије по референци (навођењем $\&$). Функција као параметре прима и низове који су статички алоцирани и попуњава њихов садржај читаним елементима (део иза тих елемената ће остати попуњен насумичним вредностима). И функција која рачуна тежиште треба да врати две вредности, па користимо пренос по референци.

```
// ucitavamo tacke u nizove x i y
void ucitajTacke(double x[], double y[], int &n) {
    cin >> n;
```

```

    for (int i = 0; i < n; i++)
        cin >> x[i] >> y[i];
}

// odredjujemo teziste (xT,yT) tacaka zadatih nizovima x i y
void teziste(double x[], double y[], double &xT, double &yT, int n) {
    xT = yT = 0.0;
    for (int i = 0; i < n; i++) {
        xT += x[i];
        yT += y[i];
    }
    xT /= n; yT /= n;
}

// transliramo tacke zadate nizovima x i y za vektor (vx, vy)
void transliraj(double x[], double y[], double vx, double vy, int n){
    for (int i=0; i<n; i++) {
        x[i] += vx;
        y[i] += vy;
    }
}

// ispisujemo tacke zadate nizovima x i y
void ispisiTacke(double x[], double y[], int n) {
    for (int i = 0; i < n; i++)
        cout << setprecision(2) << showpoint << fixed
            << x[i] << " " << y[i] << endl;
}

int main() {
    double x[100], y[100];
    int n;
    ucitajTacke(x, y, n);
    double xT, yT;
    teziste(x, y, xT, yT, n);
    transliraj(x, y, -xT, -yT, n);
    ispisiTacke(x, y, n);
    return 0;
}

```

Види груписања решења овој задајци.

Задатак: Избацивање елемената

Број је непожељан у низу целих бројева ако дели укупан број елемената (нпр. у низу дужине 10 су непожељни елементи који деле број 10, а то су 1, 2, 5 и 10). Потребно је пронаћи све непожељне елементе у низу и уклонити их. Након тога се број елемената може променити и неки други елементи могу постати непожељни. Поступак се понавља док се не добије низ без непожељних елемената. Напиши програм који за дати низ одређује збир преосталих елемената, након уклањања непожељних.

Улаз: Са стандардног улаза се уноси број n ($1 \leq n \leq 50000$), а затим и n елемената низа из распона од 1 до 100.

Излаз: На стандардни излаз исписати један цео број који представља збир преосталих елемената у низу, након узадног уклањања свих непожељних елемената.

Пример

Улаз	Израз
7	9
1	
2	
3	
4	
5	
6	
7	

Решење

Основу задатака чини филтрирање низа, тј. избацивање елемената из низа који задовољавају дато својство (о филтрирању серија елемената већ је било речи у задацима **Јутарње температуре** и). Ако би се резултат памтио у другом низу, филтрирање бисмо могли извршити тако што бисмо тако што бисмо пролазили кроз све елементе датог низа и сваки који не задовољава својство избацивања (у нашем случају сваки елемент који не дели димензију низа) додавали на крај резултата (слично смо радили у задатку **Парни и непарни елементи**). Приликом грађења резултујућег низа можемо да одржавамо број елемената тренутно смештених у резултујући низ. Поступак се заправо не разликује много ни ако се филтрирање врши у месту (у истом низу). Наиме, резултат се уместо у посебан низ може смештати на почетак самог низа који се филтрира. Постављањем елемената низа у његов почетни део не постоји опасност да се пребрише неки елемент који још није обрађен.

Ако се користи класичан, статички алоцирани низ, његова стварна димензија се не може променити након избацивања неких елемената, међутим, уз низ се одржава променљива која чува број његових попуњених елемената и скраћивање низа се постиже смањивањем вредности те променљиве.

Филтрирање елемената се понавља све док се његовом применом мења димензија низа. Кôд организујемо кроз бесконачну петљу, након филтрирања проверавамо да ли је број задржаних елемената једнак броју елемената низа, и ако јесте, петљу прекидамо.

```
int a[50000];
int n;
// učitavamo niz
// ...
while (true) {
    // uklanjamo nepoželjne elemente
    int k = 0;
    for (int i = 0; i < n; i++)
        if (n % a[i] != 0)
            a[k++] = a[i];
    // ako se broj elemenata nije promenio, tada nema nepoželjnih
    if (n == k)
        break;
    // skraćujemo dužinu niza
    n = k;
}

// računamo i ispisujemo zbir preostalih elemenata
// ...
```

Учитаване низа, филтрирање и сабирање елемената можемо реализовати и у посебним функцијама, чиме се добија мало прегледнији кôд.

```
// izbacujemo sve elemente niza a dužine n koji dele broj x
void izbacuDeliocce(int a[], int& n, int x) {
    int k = 0;
    for (int i = 0; i < n; i++)
        if (x % a[i] != 0)
            a[k++] = a[i];
    n = k;
}
```



```

}

// ...
int main() {
    // učitavamo niz
    int a[50000];
    int n;
    ucitajNiz(a, n);

    while (true) {
        // trenutni broj elemenata niza pre uklanjanja nepoželjnih
        int n0 = n;
        // izbacujemo nepoželjne elemente
        izbaciDelioce(a, n, n);
        // ako se broj elemenata nije promenio, tada u nizu nema
        // nepoželjnih i prekidamo petlju
        if (n0 == n)
            break;
    }
    // ispisujemo zbir preostalih elemenata
    cout << saberiNiz(a, n) << endl;
    return 0;
}

```

Види груписања решења овој задајци.

Задатак: Различити елементи низа

Написати програм који учитани низ целих бројева трансформише тако да се свака вредност појављује тачно једном у низу, при чему се чува редослед елемената (редослед њиховог првог појављивања).

Улаз: У једној линији стандардног улаза налази се број елемената низа N ($0 < N \leq 10000$), а затим се, у свакој од N наредних линија стандардног улаза, налази по један члан низа.

Излаз: У свакој линији стандардног излаза исписује се по један елемент трансформисаног низа.

Пример

Улаз	Излаз
6	1
1	3
3	2
1	
2	
3	
2	

Решење

Уклањање дупликата из низа се може извршити много ефикасније ако се низ претходно сортира, међутим, у овом задатку се инсистира на задржавају оригиналног редоследа елемената, тако да сортирање није могуће једноставно применити.

Једно могуће решење је да читавамо један по један од n елемената и да за сваки од њих проверавамо да ли се раније већ појавио. Можемо одржавати низ елемената који су се до сада јавили и проширивати га текућим елементом ако и само ако се он већ не налази у том низу. Пошто се не зна унапред колико ће тај низ садржати елемената, најбоље је употребити `vector`.

```

// provera da li se broj x javlja u vektoru a
bool pripada(int x, const vector<int>& a) {
    for (int y : a)
        if (x == y)
            return true;
    return false;
}

```

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

```
}

int main() {
    // u niz a upisivacemo one elemente koji vec nisu u njemu
    vector<int> a;

    // ucitavamo n elemenata
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        // ako se x ne nalazi vec u nizu a, dodajemo ga
        if (!pripada(x, a))
            a.push_back(x);
    }

    // ispisujemo elemente niza a
    for (int x : a)
        cout << x << endl;

    return 0;
}
```

Такво решење је елегантно, али у тексту задатка се наглашава да се сви елементи већ налазе у низу и да је потребно у истом том низу задржати само по једно појављивање сваког елемента. Низ ћемо трансформисати тако да се елементи који се задржавају стављају у почетни, сређени део низа. При обиласку низа елемент по елемент, за сваки наредни елемент треба проверити да ли је већ укључен у тај сређени део низа. Ако јесте, треба га прескочити, а ако није додајемо га у сређени део низа, на његов крај, и увећавамо дужину сређеног дела низа. Приметимо да се овде заправо примењује алгоритам филтрирања елемената низа који смо описали у задатку **Избацивање елемената**, при чему је својство потребно да би се елемент уклонио то да се не јавља у сређеном почетном делу низа. Провера припадности елемента почетном делу низа може се вршити алгоритмом линеарне претраге (слично као, на пример, у задатку **Негативан број**). Претрагу је могуће извршити и помоћу засебне функције.

Алгоритам се не разликује значајно у зависности од тога да ли се користи статички низ или вектор. Ако се користи статички низ, након избацивања елемената број његових попуњених елемената n треба поставити на дужину сређеног дела k , а у случају да се користи вектор, могуће му је методом `resize` променити димензију.

```
// ...
// provera da li se broj x javlja medju prvih k elemenata vektora a
bool pripada(int x, const vector<int>& a, int k) {
    for (int i = 0; i < k; i++)
        if (a[i] == x)
            return true;
    return false;
}

int main() {
    vector<int> a = unos(); // ucitavamo niz
    int k = 0; // duzina pocetnog dela
    for (int i = 0; i < a.size(); i++) { // obradjujemo sve elemente
        if (!pripada(a[i], a, k)) // ako a[i] nije u pocetnom delu
            a[k++] = a[i]; // dodajemo ga na kraj
    }
    a.resize(k); // zadržavamo samo pocetni deo
    ispis(a); // ispisujemo elemente
    return 0;
}
```

Види груґачија решења овој задајќи.

Задатак: Обртање низа

Написати програм који читава низ целих бројева a затим га трансформише тако што се окрећу задати делови низа, од елемента са индексом p до елемента са индексом q , све док се не унесе пар бројева, p и q , у коме је p веће од q .

Улаз: У једној линији стандардног улаза налази се број елемената низа, природан број N ($2 \leq N \leq 10000$), а затим се, у свакој од N наредних линија стандардног улаза, налази по један члан низа. У наредним редовима (њих највише N) се уносе по два цела броја p и q ($0 \leq p \leq q < N$), одвојена празнином док се не унесе ред у коме је први број већи од другог.

Излаз: У свакој линији стандардног излаза исписује се по један елемент трансформисаног низа.

Пример

Улаз	Излаз
4	3
1	4
2	1
3	2
4	
0 1	
2 3	
0 3	
1 0	

Решење

Обртање дела низа вршимо тако што размењујемо први елемент тог дела низа са последњим, други са претпоследњим, и тако редом, док не дођемо до средине тог дела низа. Обртање можемо реализовати засебном функцијом.

Постоји неколико начина да се имплементира петља која ово ради (слично ономе што смо приказали у задатку **Ниска палиндром**). Једна варијанта је да се употребе две променљиве i и j , и да се прва иницијализује на вредност p , а друга на вредност q . У сваком кораку петље се размењују елементи на позицијама i и j , i се увећава за 1, а j се умањује за 1. Петља се извршава све док је $i < j$.

За размену два елемента у језику C++ можемо употребити функцију `swap`.

```
// ...
// obrtanje dela niza izmedju pozicija p i q (ukljucujuci i njih)
void obrni(int a[], int p, int q) {
    for (int i = p, j = q; i < j; i++, j--)
        swap(a[i], a[j]);
}
```

Види груґачија решења овој задајќи.

Задатак: Циклично померање за једно место

Написати програм који читава низ целих бројева a затим га трансформише тако што се циклично померају задати делови низа од позиције p до позиције q све док се не унесу две једнаке позиције. При томе вршити циклично померање удесно ако је $p < q$, а померње улево вршити ако је $p > q$.

Улаз: У једној линији стандардног улаза налази се број елемената низа n ($1 < n \leq 200$), а затим се, у свакој од n наредних линија стандардног улаза, налази по један члан низа. У наредним редовима се уносе по два цела броја, p и q ($0 \leq p, q < n$), одвојена празнином док се не унесе ред у коме су бројеви једнаки.

Излаз: У свакој линији стандардног излаза исписује се по један елемент трансформисаног низа.

Пример

Улаз	Излаз
4	2
1	1
2	4
3	3
4	
2 3	
2 0	
1 2	
0 0	

Задатак: Необрисани бројеви

Посматрајмо скуп природних бројева 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, ... Прво избришемо сваки други број и посматрани скуп постаје 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, ... затим из добијеног скупа бришемо сваки трећи број и добијамо скуп 1, 3, 7, 9, 13, 15, 19, 21, ... затим из добијеног скупа бришемо сваки четврти број и тако даље. Скуп необрисаних бројева је 1, 3, 7, 13, 19, ... Написати програм којим се за дати природан број n проверава да ли припада скупу необрисаних бројева.

Улаз: Прва линија стандарног улаза садржи природан број n ($n < 5 \cdot 10^5$).

Излаз: Реч да ако број n припада скупу необрисаних бројева, у супротном реч не.

Пример 1

Улаз	Излаз
39	da

Пример 2

Улаз	Излаз
41	ne

5.1.4 Низови као репрезентација вектора, полинома и великих бројева

Низови се често користе да представе неке математичке објекте: векторе, полиноме и бројева.

Величине које су одређене само својом бројевном вредношћу (и одговарајућом јединицом) називамо *скаларним величинама* или скраћено *скаларима*. То су, на пример, маса и дужина. Постоје величине које се поред своје бројевне вредности (кажемо и интензитета) задају и својим правцем и смером и њих називамо *векторским величинама* или *векторима*. Таква је на пример брзина или убрзање.

Вектори се могу представљати као уређене торке бројева и у програмима се могу представити низовима својих координата. Наиме, вектор димензије n задаје се са n координата $\vec{a}(a_1, a_2, \dots, a_n)$. Два тако представљена вектора се могу сабирати (тако што им се саберу одговарајуће координате) и вектор се може помножити скаларом (тако што се свака координата помножи тим скаларом). Значајна је и операција скаларног производа (скаларни производ два вектора се израчунава тако што им се одговарајуће координате помноже и сви тако добијени производи саберу).

Полином често представљамо низом његових коефицијената. Нпр, полином $x^3 + 2x - 5$ представићемо низом коефицијената $(-5, 0, 2, 1)$. Слично, природни број можемо представити низом његових цифара. То је нарочито значајно код јако великих бројева, који се не могу представити основним типовима података.

Задатак: Скаларни производ

Написати програм који израчунава скаларни производ два вектора чије се координате уносе са стандардног улаза. Скаларни производ два вектора је збир производа њихових одговарајућих координата (скаларни производ вектора (a_1, \dots, a_n) и (b_1, \dots, b_n) једнак је $a_1 \cdot b_1 + \dots + a_n \cdot b_n$. Димензија вектора није унапред позната, али није већа од 50.

Улаз: У првој линији стандардног улаза задата је димензија вектора n ($2 \leq n \leq 50$). Након тога, у n наредних линија се уносе координате првог вектора, а након тога, у n наредних линија се уносе координате другог вектора.

Излаз: Исписати вредност скаларног производа првог и другог вектора на стандардни излаз.

Пример 1

Улаз	Изназ
3	10
1	
2	
3	
3	
2	
1	

Решење

Скаларни производ вектора је збир производа одговарајућих координата вектора. Да бисмо израчунали њихов производ, морамо најпре прочитати векторе са стандардног улаза.

У решењу се учитавају координате првог вектора у први низ и координате другог вектора у други низ. Затим се редом (у петљи) множе одговарајуће координате оба вектора и у сваком се кораку збир (почетно иницијализован на нулу) увећава за производ одговарајућих координата вектора (приметимо да се користи алгоритам сабирања серије бројева, описан у задатку **Збир n бројева**).

```
// izracunavanje skalarnog proizvoda
int skalarniProizvod = 0;
for (int i = 0; i < n; i++)
    skalarniProizvod += a[i]*b[i];
```

Приметимо да за израчунавање производа није неопходно чувати у меморији оба низа, већ је довољно учитати само први вектор у низ, а затим учитавати једну по једну координату другог вектора, помножити је са одговарајућом координатом првог вектора и додавати на текући збир.

Задатак: Вредност полинома

Са стандардног улаза се уносе степен n и реални коефицијенти полинома $y = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \dots + x \cdot a_1 + a_0$. Напиши програм који израчунава вредност тог полинома у k равномерно распоређених тачака интервала $[p, q]$.

Улаз: У првој линији стандардног улаза унети n ($2 \leq n \leq 9$) - степен полинома, у следећих $n + 1$ линија реалне вредности коефицијената полинома (редом бројеви $a_n, a_{n-1}, \dots, a_1, a_0$), затим, у наредној линији k ($2 \leq k \leq 40$) - број равномерно распоређених тачака на интервалу $[p, q]$, у наредној линији реалну вредност p - почетак интервала, и у наредној линији реалну вредност q - крај интервала.

Изназ: У k линија исписати вредност полинома у равномерно распоређеним тачакама интервала $[p, q]$ заокружену на две децимале.

Пример

Улаз	Изназ
2	4.00
1.0	9.00
2.0	16.00
1.0	25.00
7	36.00
1.0	49.00
7.0	64.00

Решење

Хорнерова шема

Вредност полинома

$$y = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \dots + x \cdot a_1 + a_0$$

се може израчунати без коришћења операције степеновања ако се полином представи у Хорнеровом облику (који смо описали, на пример, у задатку **Јарди, стопе и инчи** или **UNIX време**):

$$y = (((\dots((a_n \cdot x + a_{n-1}) \cdot x + a_{n-2}) \cdot x + \dots + a_1) \cdot x + a_0$$

Ако се y иницијализује са 0, y $n + 1$ итерација се израчунава $y = y \cdot x + a_n, y = y \cdot x + a_{n-1}, \dots, y = y \cdot x + a_0$.

Генерисање k равномерних тачака у интервалу $[p, q]$ описали смо у задатку [Подела интервала на једнаке делове](#). Да би се вредност полинома израчунала у k равномерно распоређених тачака интервала $[p, q]$ - вредност полинома се израчунава у тачкама $p + i \cdot h$ за $i = 0, 1, \dots, k - 1$ и $h = (q - p)/(k - 1)$.

```
double vrednostPolinoma(double a[], int n, double x) {
    // Hornerova shema
    double y = 0.0;
    for (int i = n; i >= 0; i--)
        y = y*x + a[i];
    return y;
}

int main() {
    // ucitavamo stepen n i koeficijente polinoma a
    // broj tacaka k i granice intervala p, q
    // ...
    double h = (q - p) / (k - 1); // duzina intervala
    double x; // tekuca tacka
    int i;
    for (i = 0, x = p; i < k; i++, x += h)
        cout << fixed << showpoint << setprecision(2)
            << vrednostPolinoma(a, n, x) << endl;
}

```

Класична дефиниција

Уместо Хорнерове шеме може се употребити и класична дефиниција вредности. Тада се у сваком кораку израчунава степен x^i . У језику C++ степен се може израчунати функцијом `pow` декларисаном у заглављу `<cmath>`. Израчунати степен се множи са коефицијентом a_i (који се налази у низу коефицијената `a` на позицији `i`) и додаје на текући збир (који је иницијално постављен на нулу). Једна могућа оптимизација овог поступка (која избегава употребу степеновања) је да се одржава вредност степена (која се иницијализује на 1) и да се у сваком кораку, након увећања збира вредност степена помножи са x (слично како је то приказано у задацима [Цифре сдесна](#) и [Број формиран од датих цифара здесна на лево](#)). Ипак, Хорнерова шема је најелегантније и најефикасније решење.

```
double vrednostPolinoma(double a[], int n, double x) {
    // klasicna definicija
    double y = 0.0;
    for (int i = 0; i <= n; i++)
        y += a[i] * pow(x, i);
    return y;
}

```

Задатак: Аритметика над полиномима

Дата су два полинома, P и Q степенима и низовима својих коефицијената. Одредити њихов збир и производ.

Улаз: У првој линији стандардног улаза налази се степен n ($0 \leq n \leq 20$) првог полинома, а у следећих $n + 1$ линија реални коефицијенти првог полинома и то редом почев од коефицијента уз највећи степен. Затим се на стандардном улазу налази степен m ($0 \leq m \leq 20$) другог полинома, а у следећих $m + 1$ линија реални коефицијенти другог полинома и то редом почев од коефицијента уз највећи степен.

Издаз: Приказати редом збир и производ, сваки у посебној линији. За сваки полином приказати у једној линији његове коефицијенте, на две децимале и то редом почев од коефицијента уз највећи степен.

Пример

Улаз	Изназ
2	2.00 2.00 1.00
2	2.00 -1.00 1.00 -2.00
1	
2	
1	
1	
-1	

Решење

Нека су дати полиноми

$$P_n(x) = p_n \cdot x^n + p_{n-1} \cdot x^{n-1} + \dots + x \cdot p_1 + p_0$$

$$Q_m(x) = q_m \cdot x^m + q_{m-1} \cdot x^{m-1} + \dots + x \cdot q_1 + q_0$$

Збир полинома одређујемо на следећи начин:

- ако су полиноми истог степена ($n = m$) резултат је полином степена n и његови коефицијенти су једнаки збиру одговарајућих коефицијената полинома P и Q ,
- ако су полиноми различитих степени нпр. $n > m$ онда је збир полином степена n и коефицијенти од n до $m - 1$ једнаки су коефицијентима полинома P , а коефицијенти од m до 0 једнаки су збиру одговарајућих коефицијената полинома P и Q .

Производ полинома P и Q је полином R степена $n + m$. Производ одређујемо тако што множимо сваки моноом једног полинома сваким мономом другог полинома и сабирамо добијене мономе истог степена.

$$P \cdot Q = \sum_{i=0}^n \sum_{j=0}^m p_i \cdot q_j \cdot x^{i+j}$$

Сабирање вршимо поступно - на почетку све коефицијенте резултујућег полинома c_k постављамо на нулу, а затим за свако i од 0 до n и за свако j од 0 до m увећамо коефицијент c_{i+j} за вредност производа $p_i \cdot q_j$ (приликом множења монома $p_i \cdot x^i$ мономом $q_j \cdot x^j$ добијамо моноом $p_i \cdot q_j \cdot x^{i+j}$, који се додаје моному $c_{i+j}x^{i+j}$ тренутно акумулираном у резулату).

У имплементацији коефицијенте полинома и степен полинома можемо чувати обједињене унутар структуре.

```

struct Polinom {
    int n; // stepen
    vector<double> a; // koeficijenti an, ..., a0
};

// pravi nula polinom stepena n
Polinom napravi(int n) {
    Polinom P;
    P.n = n;
    P.a.resize(n+1, 0);
    return P;
}

// cita stepen i koeficijente sa standardnog ulaza
Polinom unos() {
    int n;
    cin >> n;
    Polinom P = napravi(n);
    for (int i = P.n; i >= 0; i--)
        cin >> P.a[i];
    return P;
}

```

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

```
// ispisuje koeficijente na standardni izlaz
void prikaz(Polinom P) {
    for (int i = P.n; i >= 0; i--)
        cout << fixed << setprecision(2) << showpoint << P.a[i] << " ";
    cout << endl;
}

// sabira polinome P i Q
Polinom zbir(Polinom P, Polinom Q) {
    Polinom R = napravi(max(P.n, Q.n));
    for (int i = 0; i <= R.n; i++) {
        if (i <= P.n)
            R.a[i] += P.a[i];
        if (i <= Q.n)
            R.a[i] += Q.a[i];
    }
    return R;
}

// mnozi polinome P i Q
Polinom proizvod(Polinom P, Polinom Q) {
    Polinom R = napravi(P.n + Q.n);
    for (int i = 0; i <= P.n; i++)
        for (int j = 0; j <= Q.n; j++)
            R.a[i + j] += P.a[i] * Q.a[j];
    return R;
}

int main() {
    Polinom P, Q, R, S;
    P = unos();
    Q = unos();
    R = zbir(P, Q);
    prikaz(R);
    R = proizvod(P, Q);
    prikaz(R);
    return 0;
}
```

Задатак: Збир два троцифрена

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Сабирање се може уопштити и на случај вишецифрених бројева, ако се они представе низовима цифара и употреби се петља.

```
// učitavamo sve cifre
int a[3];
int b[3];
cin >> a[2] >> a[1] >> a[0];
cin >> b[2] >> b[1] >> b[0];
// cifre zbira
int z[4];

// sabiramo cifre odmah vrseci prenos
int prenos = 0;
for (int i = 0; i < 3; i++) {
```



```

    int zbirCifara = a[i] + b[i] + prenos;
    z[i] = zbirCifara % 10;
    prenos = zbirCifara / 10;
}
// upisujemo preostali prenos
z[3] = prenos;

// preskacemo vodece nule
int i;
for (i = 3; i > 0 && z[i] == 0; i--)
    ;
// ispisujemo rezultat
for (; i >= 0; i--)
    cout << z[i];
cout << endl;

```

Задатак: Линеарна функција над великим бројевима

У великом броју језика основни типови података (нпр. `int`) могу исправно да представе само ограничен распон бројева. Напиши програм који израчунава вредност функције $a \cdot x + b$, ако су a , b и x јако велики цели бројеви.

Улаз: Са стандардног улаза се учитавају три цела броја a , b и x (са најмање 3, а највише 100 цифара), сваки у посебној линији.

Излаз: На стандардни излаз исписати вредност израза $a \cdot x + b$.

Пример

Улаз	Излаз
123456789	123456789864197532
987654321	
999999999	

Решење

Бројеве ћемо представити низовима цифара.

Пошто ћемо у неким специјалним случајевима низове проширивати додатним водећим цифрама или ћемо из њих уклањати водеће нуле, уместо класичних низова можемо употребити библиотечке колекције. У језику C++ можемо употребити векторе (тј. тип `vector<int>`). Цифре ћемо смештати тако да се на почетку низа налазе цифре најмање тежине (цифре јединица), а на крају цифре највеће тежине. Раније смо видели да је вредност броја чије су цифре a_0, \dots, a_n једнака $a_n \cdot 10^n + \dots + a_0 \cdot 10^0$. Дакле, у запису који смо одабрали коефицијент уз 10^i ће се налазити у низу на позицији i .

Алгоритам сабирања можемо вршити на исти начин као у задатку **Збир два троцифрена**. Чувамо тренутни пренос (који иницијализујемо на нулу) и у сваком кораку сабирамо текуће цифре два броја (ако нека не постоји, јер је један број краћи од другог, тада на месту те цифре рачунамо да се налази нула) и пренос са претходне позиције, цифру резултата одређујемо као остатак добијеног збира при дељењу са 10, а нови пренос као целобројни количник тог збира при дељењу са 10. Ако нам након сабирања свих цифара остане пренос који је већи од нуле, тада њега уписујемо на место највеће тежине резултата (додајемо га на крај низа цифара којима је представљен резултат).

Ипак, елегантнији код можемо добити ако операцију извршимо у две фазе (и та је техника приказана у задатку **Збир два троцифрена**). У првој фази можемо само сабрати коефицијенте на одговарајућим позицијама (поново водећи рачуна о томе да бројеви не морају бити исте дужине). Суштински, у овој фази израчунавамо збир два полинома, исто као у задатку **Аритметика над полиномима**. У другој фази вршимо нормализацију добијеног резултата, тако што вршимо пренос све док не обезбедимо услов да су све цифре броја између 0 и 9. Крећемо од почетка низа (цифре најмање тежине), преносимо на следећу позицију целобројни количник те цифре и броја 10, а на позицију те цифре уписујемо целобројни остатак те цифре при дељењу са 10. Обратимо пажњу да број цифара треба да буде за 1 већи од броја цифара дужег од два броја који се сабирају. Ако се приликом нормализације не изврши пренос на ту позицију (ако тај последњи елемент низа остане нула), та се цифра може уклонити из низа.

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

Са функцијом нормализације на располагању, веома је једноставно извршити и множење. Примењујемо алгоритам множења полинома описан у задатку **Аритметика над полиномима** и затим позивамо функцију нормализације.

```
void normalizuj(vector<int>& c) {
    for (int k = 0; k < c.size() - 1; k++) {
        c[k + 1] += c[k] / 10;
        c[k] %= 10;
    }
    if (c.back() == 0)
        c.pop_back();
}

vector<int> saberi(const vector<int>& a, const vector<int>& b) {
    int n = max(a.size(), b.size());
    vector<int> c(n + 1);
    for (int i = 0; i < n; i++) {
        c[i] = 0;
        if (i < a.size())
            c[i] += a[i];
        if (i < b.size())
            c[i] += b[i];
    }
    normalizuj(c);
    return c;
}

vector<int> pomnozi(const vector<int>& a, const vector<int>& b) {
    vector<int> c(a.size() + b.size(), 0);
    for (int i = 0; i < a.size(); i++)
        for (int j = 0; j < b.size(); j++)
            c[i + j] += a[i] * b[j];
    normalizuj(c);
    return c;
}

string pisiBroj(const vector<int>& broj) {
    string rez = "";
    for (int i = broj.size() - 1; i >= 0; i--)
        rez += '0' + broj[i];
    return rez;
}

vector<int> citajBroj() {
    string s;
    cin >> s;
    int n = s.size();
    vector<int> rez(n);
    for (int i = 0; i < n; i++)
        rez[i] = s[n - 1 - i] - '0';
    return rez;
}

int main() {
    vector<int> a, b, x;
    a = citajBroj();
    b = citajBroj();
    x = citajBroj();
}
```

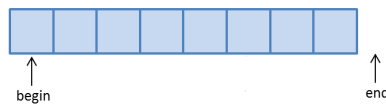
```
cout << pisiBroj(saberi(pomnozi(a, x), b)) << endl;
return 0;
}
```

Библиотека репрезентација великих бројева

5.1.5 Библиотеке функције за рад са низовима

У овом поглављу ћемо приказати решења заснована на употреби библиотечких колекција (низова, листи, вектора и сл.) и функција. Нагласимо да се већина задатака из овог поглавља може решити и без употребе колекција. Иако је програмски код таквих решења обично дужи, она су свакако меморијски ефикаснија. Такође, таква решења се могу сматрати илустративнијим, јер се кроз њих изучавају веома важни, фундаментални алгоритми. Са друге стране, у неким ситуацијама у програмирању ћемо елементе морати сместити у низ или ћемо их из неког разлога већ имати у том облику. Тада се применом библиотечких функција могу добити веома кратка и елемента решења. Библиотечке функције ћемо у овом поглављу приказати кроз веома једноставне задатке у којима њихова примена не мора да буде оптималан избор.

Библиотечке функције у језику C++ су дизајниране тако да се могу примењивати над различитим колекцијама података (статичким и динамичким низовима, векторима, листама и слично). Колекција (или неки њен део тј. опсег узастопних елемената) задају се **итераторима**. Итератори се најчешће користе у пару и ограничавају опсег узастопних елемената колекције тако што први итератор указује на почетни елемент тог опсега, а други указује непосредно иза последњег елемента опсега.



Слика 5.1: Итератори `begin` и `end`

Неке од најчешће коришћених функција декларисане у заглављу `<iterator>` су:

- `begin`, `end` – враћају итераторе који ограничавају опсег датог низа, вектора и слично. Функција `begin` враћа итератор који указује на први елемент, а `end` враћа итератор који указује непосредно иза последњег елемента (на пример `begin(v)` враћа итератор који указује на почетак вектора `v`). Многе колекције подржавају и методе `begin` и `end` (на пример, `v.begin()` враћа итератор који указује на почетак вектора `v`). Име низа је могуће употребити као итератор који указује на почетак низа;
- `distance` – враћа растојање (број елемената) у опсегу ограниченом са два итератора која се прослеђују као аргументи функције (први итератор показује на почетак тј. на први елемент опсега, а други непосредно иза краја тј. последњег елемента опсега). На пример, ако је `it` итератор који указује на неки елемент унутар вектора `v`, тада се његов индекс може одредити помоћу `distance(begin(v), it)`;
- `next`, `prev` – враћа итератор који показује на елемент дате колекције који је испред тј. иза прослеђеног итератора, на датом растојању; ако се као други аргумент не проследи растојање, подразумевано се тражи итератор на наредни тј. претходни елемент колекције. На пример, `next(begin(v))` је итератор који указује на други елемент вектора `v` (ако такав постоји), док је `prev(end(a), 2)` итератор који указује на претпоследњи елемент низа `a` (ако такав постоји).
- Над итераторима се могу примењивати и аритметичке операције: `it + n` одговара итератору који се добија када се итератор `it` помери надесно `n` пута (исто као и `next(it, n)`). На пример, ако низ `a` има `n` елемената тада се његов опсег може задати итераторима `a` и `a+n`. Разлика два итератора одређује број елемената између њих (укључујући први и не укључујући последњи (исто као и функција `distance`)).

Неке од најчешће коришћених функција декларисане у заглављу `<algorithm>` су:

- `min_element` и `max_element` – враћа итератор на елемент са најмањом тј. највећом вредношћу у опсегу задатом помоћу два дата итератора; као трећи аргумент се функцији може проследити и функција која врши поређење два елемента (подразумевано се користи поређење оператором `<`);
- `transform` – пресликавају се сви елементи из опсега датог помоћу два итератора; као трећи аргумент се прослеђује итератор на почетак колекције у коју се резултат смешта, а као четврти функција (најчешће анонимна) којом се елементи пресликавају;
- `count` – враћа број појављивања елемента у опсегу задатом помоћу два итератора;

- `copy_if` – колекција се филтрира тако што се копирају они елементи из опсега задатог помоћу два итератора који задовољавају услов који се испитује функцијом која се задаје као последњи аргумент функције `copy_if` (та функција прима елемент колекције и враћа `true` ако он треба да буде копиран); филтрирани елементи смештају се у колекцију чији се итератор почетка прослеђује функцији `copy_if` као њен трећи аргумент и подразумева се да је она алоцирана тако да може да прихвати све елементе који се копирају. Ако број тих елемената није унапред познат, могуће је проследити `back_inserter(it)` где је `it` итератор који указује на почетак неалоцираног вектора (тада се приликом копирања елементи у вектор убацују помоћу `push_back`).
- `remove`, `remove_if` – колекција се филтрира тако што се из ње уклањају они елементи из опсега задатог помоћу два итератора који задовољавају неки услов; у случају функције `remove` задаје се вредност и уклањају се сви елементи једнаки тој вредности, док се у случају функције `remove_if` услов испитује функцијом која се задаје као последњи аргумент функције `remove_if` (та функција прима елемент колекције и враћа `true` ако он треба да буде обрисан); преостали елементи се премештају на почетак оригиналне колекције, а функција `remove_if` враћа итератор непосредно иза последњег задржаног елемента; након уклањања елемената, колекција се често скраћује функцијом `erase`;
- `erase` – брише елемент из колекције задат итератором или опсег колекције задат помоћу два итератора;
- `any_of`, `all_of` – проверава да ли неки елемент тј. да ли сви елементи опсега задатог помоћу два итератора задовољавају услов који се проверава `bool` функцијом која се задаје као трећи аргумент функције `any_of`; функција `any_of` тј. `all_of` такође враћа вредност типа `bool`;
- `find`, `find_if` – враћа итератор који показује на прво појављивање елемента која се тражи у опсегу задатом помоћу два итератора; ако се елемент не јавља у овом опсегу елемената враћа се итератор који показује на позицију иза краја датог опсега; функција `find` прима вредност која се тражи, док функција `find_if` прима функцију која проверава да ли текући елемент има захтевано својство и проналази први елемент који то својство задовољава.
- `equal` – проверава да ли су два опсега једнака; као аргументе прима итераторе који ограничавају први опсег и итератор који показује на почетак другог опсега;
- `reverse` – обрће редослед елемената опсега задатог помоћу два итератора.
- `sort` – сортира елементе опсега задатог помоћу два итератора. Они се подразумевано пореде оператором `<`, а могуће је као трећи аргумент функцији `sort` проследити и функцију којом ће се два елемента поредити (она треба да врати `true` ако и само ако први треба да се нађе испред другог у сортираном редоследу).

У неким примерима ћемо користити и наредне функције декларисане у заглављу `<numeric>`:

- `iota` – попуњава елементе колекције (низа или вектора) из датог опсега задатог помоћу два итератора узастопним бројевима почев од вредности коју наводимо као трећи аргумент функције;
- `accumulate` – рачуна збир елемената у опсегу задатом помоћу два итератора; као трећи параметар наводи се иницијална вредност збира – најчешће је то `0` или `0.0` у зависности од типа елемената који се сабирају. Функција `accumulate` израчунава збир тако што обрађује један по један елемент опсега у сваком кораку текућу вредност збира сабира са текућим елементом опсега. То се може променити ако се као четврти аргумент функције `accumulate` зада функција која прихвата досадашњу акумулирану вредност и текући елемент опсега и која описује како се нова акумулирана вредност добија на основу њих (та функција треба да врати нову вредност).

Задатак: Бројеви од а до b

Овај задатак је ионовљен у циљу увежбавања различитих техника решавања. *Види икекст задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Рецимо и да у језику C++ постоји библиотечка функција `iota` декларисана у заглављу `<numeric>` која попуњава низ или вектор узастопним бројевима. Као што смо видели иста се функционалност може постићи једном једноставном петљом, тако да функцију `iota` нећемо детаљно објашњавати.

```
int a, b;
cin >> a >> b;
vector<int> v(b - a + 1);
iota(begin(v), end(v), a);
```

```
for (int x : v)
    cout << x << endl;
```

Задатак: Збир n бројева

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текснi задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

У језику С++ ако су бројеви смештени у низ или вектор, збир њихових елемената је могуће добити функцијом `accumulate`. Елементе можемо сместити и у вектор `a`, који алоцирамо на дужину `n`, одмах након што учитамо вредност `n`. У том случају, итератори који ограничавају цео вектор су `a.begin()` и `a.end()` (а могу се добити и помоћу `begin(a)` и `end(a)`).

```
int n; // број бројева
cin >> n;
// учитавамо n бројева у вектор
vector<int> a(n);
for (int i = 0; i < n; i++)
    cin >> a[i];
// одреджимо i исписујемо њихов збир
int zbir = accumulate(begin(a), end(a), 0);
cout << zbir << endl;
```

Задатак: Просек свих бројева до краја улаза

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текснi задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Ако би сви елементи били смештени у неку колекцију, тада бисмо просек могли једноставно израчунати библиотечким функцијама. Пошто не знамо унапред колико ће елемената бити на улазу потребно је да користимо неку колекцију којој можемо додавати елементе. У језику С++ погодно је да елементе један по један додајемо у вектор (методом `push_back`). У језику С++ просек се може израчунати као количник збира којег добијамо функцијом `accumulate` описаном у задатку **Збир n бројева** и броја елемената којег за вектор можемо добити методом `size`. У језику С++ просек се може израчунати као количник збира којег добијамо функцијом `sum` описаном у задатку **Збир n бројева** и броја елемената листе који можемо добити функцијом `len`.

```
vector<int> a;
int x;
while (cin >> x)
    a.push_back(x);
int zbir = accumulate(begin(a), end(a), 0);
int broj = a.size();
double prosek = (double)zbir/(double)broj;
cout << fixed << showpoint << setprecision(5)
    << prosek << endl;
```

Задатак: Факторијел

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текснi задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Библиотечке функције

У језику С++ производ елемената низа или вектора је могуће израчунати функцијом `accumulate` коју смо, подсетимо се, у задатку **Збир n бројева** употребили за израчунавање збира елемената вектора. Наиме, ова

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

функција може да прими и четврти параметар у виду функције која одређује бинарну операцију којом се број добијеним обрадом претходних елемената у низу обједињава са наредним елементом низа (тај број се иницијализује на вредност дату као трећи параметар). Ако се за трећи параметар наведе вредност 1, а за четврти функција која множи своја два аргумента, `accumulate` ће извршити множење. Низ или вектор можемо напунити бројевима од 1 до n функцијом `iota`.

```
int n;
cin >> n;
vector<int> a(n);
iota(begin(a), end(a), 1);
int f = accumulate(begin(a), end(a), 1,
    [](int acc, int x) {
        return acc * x;
    });
cout << f << endl;
```

Задатак: Најнижа температура

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текснi задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Још једно решење већ описано у задатку **Најлошији контролни** је да се температуре у свим градовима учитају у један низ и да се онда за проналажење минимума употреби библиотечка функција. У језику C++ се може употребити функција `min_element` декларисана у заглављу `<algorithm>`.

```
// broj gradova
int n;
cin >> n;
// učitavamo temperature u svim gradovima
vector<int> temperature(n);
for (int i = 0; i < n; i++)
    cin >> temperature[i];

// ispisujemo minimum
cout << *min_element(begin(temperature), end(temperature)) << endl;
```

Задатак: Претицање

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текснi задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Низови и библиотечке функције

Задатак опет можемо решити применом библиотечких функција. Могуће је да сва пронађена времена претицања убацимо у низ и да применимо библиотечку функцију за налажење максимума низа (о овоме је било речи у задатку **Најлошији контролни**). Проблем са овим решењем је то што не знамо унапред колико ће претицања бити тј. колико елемената низ треба да има. У таквим ситуацијама можемо употребити неки облик низа који допушта да му се елементи додају један по један, аутоматски прилагођавајући своју величину тренутно додатим бројем елемената. У језику C++ можемо употребити колекцију типа `vector<int>`, којој се елемент додаје на крај методом `push_back`. Приступ елементима вектора, одређивање максимума и минимума и многе друге операције се векторима се врше на идентичан начин као и са обичним низовима. Број елемената можемо одредити методом `size`, проверу да ли је вектор празан методом `empty`, а максимум функцијом `max_element`.

```
double vremePreticanja(double tA, double vA, double tB, double vB) {
    return (vB*tB - vA*tA) / (vB - vA);
}
```

```

int main() {
    double t1, t2, t3;
    double v1, v2, v3;

    // ...
    vector<double> preticanja;

    if (v2 > v1)
        preticanja.push_back(vremePreticanja(t1, v1, t2, v2));
    if (v3 > v1)
        preticanja.push_back(vremePreticanja(t1, v1, t3, v3));
    if (v3 > v2)
        preticanja.push_back(vremePreticanja(t2, v2, t3, v3));

    if (preticanja.size() > 0)
        cout << fixed << showpoint << setprecision(2)
            << *max_element(preticanja.begin(), preticanja.end()) << endl;
    else
        cout << "nema" << endl;

    return 0;
}

```

Задатак: Најтоплији дан

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види итексиј задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Једно елегантно решење је да се употребе низови и библиотечке функције за одређивање максимума низа.

У задатку **Најлошији контролни** видели смо да је за одређивање максимума низа могуће употребити било библиотечку функцију `max` са анонимним низом (листом иницијализатора) било функцију `max_element` са неким обликом именованог низа. Функција `max` враћа само вредност максимума, те је боље употребити функцију `max_element` која враћа итератор (практично позицију) пронађеног максимума, тако да се његов редни број може одредити израчунавањем растојања од почетка низа (функцијом `distance`) и увећавањем за 1 (јер бројање дана треба да крене од 1, а позиције у низу се броје од 0).

```

// niz temperatura
int t[7];
cin >> t[0] >> t[1] >> t[2] >> t[3] >> t[4] >> t[5] >> t[6];
int rbrNajtoplijegDana = distance(t, max_element(t, t+7)) + 1;
cout << rbrNajtoplijegDana << endl;

```

Задатак: Минимално одступање од просека

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види итексиј задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Пошто су елементи смештени у низ (вектор или листу), задатак могуће решити и применом библиотечких функција за рад са колекцијама.

У језику C++ просек можемо израчунати коришћењем функције `accumulate` за сабирање која је декларисана у заглављу `<numeric>`.

Одређивање минимума колекције или неког њеног сегмента може се урадити коришћењем функције `min_element`, коју смо већ видели у задатку **Најнижа температура**. Позив `distance(a, min_element(a, next(a, n)))` одређује позицију првог појављивања најмањег у низу бројева.

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

Пошто ми желимо да поређење вршимо на основу растојања од просека, функцији `min_element` можемо проследити функцију која прима два броја, рачуна њихова растојања од просека и проверава да ли је први на мањем растојању. Најлакши начин је да за то употребимо анонимну функцију (тзв. λ -функцију). Овим добијамо вредност елемента из вектора `a` који најмање одступа од просека (његово растојање од просека потребно је поново израчунати).

```
// ...
// racunamo prosek kao kolicnik zbira i broja elemenata
double prosek = accumulate(a.begin(), a.end(), 0.0) / n;

// odredjujemo element koji ima najmanje rastojanje od proseka
double minRastojanje =
    *min_element(a.begin(), a.end(),
        [prosek](double a, double b) {
            return abs(a - prosek) < abs(b - prosek);
        });
```

Још једна могућност је да прво елементе низа заменимо њиховим растојањем од просека, а онда да нађемо минимум тако трансформисаног низа (мана овог решења је то што се губи почетни садржај низа и то бисмо могли решити тако што бисмо увели нови низ за смештање ових растојања). Трансформацију можемо извршити помоћу функције `transform` коју смо већ употребили у задатку [Прерачунавање миља у километре](#).

```
const int MAX_N = 100;

int main() {
    // broj elemenata niza
    int n;
    cin >> n;
    // učitavamo sve elemente u niz
    double a[MAX_N];
    for (int i = 0; i < n; i++)
        cin >> a[i];
    // izracunavamo prosek elemenata (kao kolicnik zbira i broja elementa)
    double prosek = accumulate(a, next(a, n), 0.0) / n;
    // elemente niza menjamo njihovim apsolutnim razlikama od proseka
    transform(a, next(a, n), a,
        [prosek](double a) { return abs(a - prosek); });
    // odredjujemo najmanji element transformisanog niza
    double minDist = *min_element(a, next(a, n));
    // ispisujemo konacan rezultat
    cout << fixed << showpoint << setprecision(2)
        << minDist << endl;
}
```

Задатак: Максимална разлика суседних

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. [Види тегскај зајатка](#).

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Коришћење низа

Једно решење је да се сви бројеви прво прочитају у низ, да се након тога креира низ апсолутних разлика суседних елемената и да се и да се затим максимална разлика одреди библиотечком функцијом. Ово решење троши више меморије него што је неопходно.

```
// ...
// kreiramo niz razlika
vector<int> razlike(n-1);
for (int i = 1; i < n; i++)
```



```

    razlike[i-1] = abs(a[i] - a[i-1]);
// odredjujemo maksimum
int max = *max_element(begin(razlike), end(razlike));
// ...

```

Задатак: Прерачунавање миља у километре

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шекст задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Под претпоставком да је серија елемената смештена у неку колекцију, могуће је употребити библиотечку подршку за пресликавање.

У језику С++ за пресликавање можемо искористити функцију `transform` декларисану у заглављу `algorithm`.

```

int a, b, korak;
cin >> a >> b >> korak;

// popunjavamo vektor milja koje treba превести
vector<int> milje;
for (int brojMilja = a; brojMilja <= b; brojMilja += korak)
    milje.push_back(brojMilja);

// broj elemenata serije
int n = milje.size();

// preslikavamo sve elemente niza funkcijom koja конвертује миље
// у километре и резултат преваремо у вектор бројева
vector<double> kilometri(n);
transform(begin(milje), end(milje), begin(kilometri),
    [](int mi) {
        const double kilometaraUMilji = 1.609344;
        return mi * kilometaraUMilji;
    });

// ispisujemo rezultat (elemente oba vektora)
for (int i = 0; i < n; i++)
    cout << fixed << showpoint << setprecision(6)
        << milje[i] << " mi" << " = " << kilometri[i] << " km" << endl;

```

Задатак: Транслација тачака

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шекст задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Задатак можемо решити и применом библиотечких функција. У језику С++ просек елемената низа можемо одредити функцијом као у задатку **Средине**, као количник збира којег добијамо функцијом `accumulate` (о којој је било речи у задатку **Збир n бројева** и **Минимално одступање од просека**) и броја тачака. Трансформацију низа елемената у језику С++ можемо извршити функцијом `transform` о којој је било речи у задатку **Прерачунавање миља у километре**, али и **Минимално одступање од просека**.

```

// ...
// izracunavamo teziste, kao aritmeticku sredinu koordinata
double xT = accumulate(x.begin(), x.end(), 0.0) / n;
double yT = accumulate(y.begin(), y.end(), 0.0) / n;

// transliramo за вектор (-xT, -yT)

```

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

```
transform(x.begin(), x.end(), x.begin(),
         [xT](double x) { return x - xT; });
transform(y.begin(), y.end(), y.begin(),
         [yT](double y) { return y - yT; });

// ...
```

Задатак: Магацин сокова

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Потребно је израчунати збир елемената серије бројева коју добијамо пресликавањем тј. применом функције $h(f) = \left\lfloor \frac{f}{k} \right\rfloor$ на серију бројева флаша на свакој полици. Стога до решења можемо доћи и применом библиотечких колекција и функција. Пошто је број елемената колекције унапред познат, можемо употребити класични низ. Сабирање пресликане серије можемо постићи тиме што функцији `accumulate` проследимо функцију као четврти аргумент (слично као у задатку [Средине](#)).

```
// izracunavamo ukupan broj gajbica
int ukupanBrojGajbica =
    accumulate(begin(brojFlasa), end(brojFlasa), 0,
               [k](int acc, int f) {
                   return acc + (f + k - 1) / k;
               });
```

У језику C++ можемо учитати све бројеве флаша на полицама у низ или вектор, прсликати га у нови низ тј. вектор који садржи број гажбица за сваку полицу помоћу функције `transform` коју смо описали у задатку [Прерачунавање миља у километре](#), а затим све елементе сабрати помоћу функције `accumulate` коју смо описали у задатку [Збир n бројева](#).

```
// ...
// izracunavamo broj gajbica potrebnih za flase na svakoj polici
vector<int> brojGajbica(n);
transform(begin(brojFlasa), end(brojFlasa), begin(brojGajbica),
         [k](int f) { return (f + k - 1) / k; });

// izracunavamo ukupan broj gajbica
int ukupanBrojGajbica =
    accumulate(begin(brojGajbica), end(brojGajbica), 0);
```

Задатак: Средине

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Ако претпоставимо да све брзине учитавамо у неку колекцију, до решења можемо једноставно стићи применом библиотечких функција. У језику C++ број елемената у вектору можемо добити методом `size`, збир његових елемената можемо израчунати функцијом `accumulate`, а вектор реципрочних вредности можемо добити функцијом `transform`, који опет можемо сабрати методом `accumulate`. Креирање помоћног низа реципрочних вредности можемо избећи тако што модификујемо функцију сабирања навођењем њеног четвртог параметра (слично као у задатку [Факторијел](#)) - функције која досадашњи збир увећава за реципрочну вредност текућег елемента.

```
vector<double> a;
double x;
while (cin >> x)
    a.push_back(x);
```

```

int broj = a.size();
double zbir = accumulate(begin(a), end(a), 0.0);
double zbirReciprocnih = accumulate(begin(a), end(a), 0.0,
    [](double acc, double x) {
        return acc + 1.0 / x;
    });
double aritmetickaSredina = zbir / broj;
double harmonijskaSredina = broj / zbirReciprocnih;
cout << fixed << showpoint << setprecision(2)
    << aritmetickaSredina << endl
    << harmonijskaSredina << endl;

```

Задатак: Бројеви дељиви са 3

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексат задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Ако све елементе сместимо у низ, задатак можемо решити и применом библиотечких функција. У језику С++ филтрирање елемената можемо постићи функцијом `copy_if` декларисаном у заглављу `<algorithm>`.

```

// učitavamo sve elemente u niz
int n;
cin >> n;
vector<int> a(n);
for (int i = 0; i < n; i++)
    cin >> a[i];
// izdvajamo u novi niz one koji su deljivi sa 3
vector<int> deljiviSa3;
copy_if(begin(a), end(a), back_inserter(deljiviSa3),
    [](int x) { return x % 3 == 0; });
// ispisujemo sadrzaj rezultujuceg niza
for (int x : deljiviSa3)
    cout << x << endl;

```

Задатак: Просек одличних

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексат задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Под претпоставком да су просеци смештени у низ, задатак можемо једноставно решити и применом библиотечких функција. У језику С++ филтрирање можемо извршити функцијом `copy_if` (слично као у задатку), а затим просек израчунати сабирањем помоћу функције `accumulate` декларисане у заглављу `<numeric>` и дељењем збира са бројем одличних које можемо прочитати методом `size` (слично као у задатку [Средине](#)).

```

// učitavamo proseke svih ucenika
int n;
cin >> n;
vector<double> proseci(n);
for (int i = 0; i < n; i++)
    cin >> proseci[i];
// izdvajamo proseke samo odlicnih ucenika
vector<double> proseciOdlisnih;
copy_if(begin(proseci), end(proseci), back_inserter(proseciOdlisnih),
    [](double p) { return p >= 4.5; });

```

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

```
// ako ima odlicnih učenika
if (proseciOdlicnih.size() != 0) {
    // izracunavamo i ispisujemo njihov prosek kao zbir / broj
    double prosekOdlicnih =
        accumulate(begin(proseciOdlicnih), end(proseciOdlicnih), 0.0) /
        proseciOdlicnih.size();
    cout << fixed << setprecision(2) << showpoint
        << prosekOdlicnih << endl;
} else
    cout << "-" << endl;
```

Задатак: Парни и непарни елементи

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Филтрирање библиотечким функцијама

Филтрирање је могуће остварити и помоћу библиотечких функција које смо видели у задатку .

У језику С++ можемо употребити функцију `copy_if` којом се копирају они елементи који задовољавају услов који се испитује функцијом која се прослеђује као аргумент. У нашем случају функцији `copy_if` прослеђује се анонимна функција која прима број и проверава његову парност (у првом случају враћа `true` за парне, а у другом за непарне елементе). Пошто не знамо унапред колико ће елемената бити у резултату и не можемо да коректно унапред алоцирамо колекцију, користимо `back_inserter` који служи да у вектор који добије као свој аргумент убацује на крај један по један елемент који се копира.

```
// ucitavamo brojeve u niz
vector<int> a = ucitajVektor();

// izdvajamo parne elemente u niz parni
vector<int> parni;
copy_if(a.begin(), a.end(),
        back_inserter(parni),
        [](int x) { return x % 2 == 0; });

// izdvajamo neparne elemente u niz neparni
vector<int> neparni;
copy_if(a.begin(), a.end(),
        back_inserter(neparni),
        [](int x) { return x % 2 != 0; });

// ispisujemo niz ucitanih parnih brojeva
ispisiVektor(parni);
// ispisujemo niz ucitanih neparnih brojeva
ispisiVektor(neparni);
```

Задатак: Избацивање елемената

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Коришћење библиотечких функција

У језику С++ филтрирање се може извршити применом функције `remove_if` која реорганизује садржај низа тако да на његов почетак слаже елементе који не задовољавају дато својство и враћа позицију (итератор) иза дела низа у који су ти елементи сложени. Функцији `remove_if` прослеђујемо као аргумент анонимну

функцију у чијем се затворењу мора налазити број елемената низа, а која прима као параметар текући елемент низа и проверава да ли је број елемената низа дељив њиме.

За складиштење података можемо користити и вектор и тада се након реорганизације елемената функцијом `remove_if` вишак елемената може уклонити (на пример, тако што се позове метода `erase` класе `vector` којој се итератор који врати функција `remove_if` и итератор који указује иза краја вектора који се добија методом `end()` или тако што се позове метода `resize` којом се може смањити димензија вектора).

```
// učitavamo elemente u vektor
int n;
cin >> n;
vector<int> a(n);
// ...
while (true) {
    int n = a.size(); // duzina vektora
    // elemente koji su delioci duzine premestamo na pocetak
    auto it = remove_if(a.begin(), a.end(),
                       [n](int x) { return n % x == 0; });
    // ako nista nije izbaceno, prekidamo petju
    if (it == a.end())
        break;
    // brisemo deo vektora iza sredjenog pocetnog dela
    a.erase(it, a.end());
}

// izracunavamo i ispisujemo zbir preostalih elemenata
cout << accumulate(a.begin(), a.end(), 0) << endl;
```

Задатак: Бројање гласова за омиљеног глумца

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шекст задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Решење можемо засновати и на читавању свих елемената у низ и примени библиотечких функција за бројање елемената. У језику C++ можемо употребити функцију `count` чији су параметри два итератора који ограничавају распон у којем се врши пребројавање (обично су то итератори добијени помоћу `begin` и `end`) и елемент који се броји.

```
int x;
cin >> x;
int n;
cin >> n;
vector<int> a(n);
for (int i = 0; i < n; i++)
    cin >> a[i];
int brojPojavljivanja = count(begin(a), end(a), x);
cout << brojPojavljivanja << endl;
```

Задатак: Број максималних

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шекст задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Ако елементе сместимо у низ можемо употребити библиотечке функције за одређивање максимума и пребројавање. У језику C++ можемо употребити функцију `max_element` за одређивање максимума и функцију `count` за одређивање броја појављивања датог елемента у датом серији (обе су декларисане у заглављу `<algorithm>`).

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

```
// broj takmicara
int n;
cin >> n;
// učitavamo poene svih takmicara
vector<int> poeni(n);
for (int i = 0; i < n; i++)
    cin >> poeni[i];
// odredjujemo maksimum
int maks = *max_element(begin(poeni), end(poeni));
// odredjujemo broj njegovih pojavljivanja
int brojPojavljivanjaMaksimuma = count(begin(poeni), end(poeni), maks);
```

Задатак: Различити елементи низа

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текснi задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Библиотечке функције

Претрагу је могуће извршити и помоћу библиотечке функције. У језику С++ претрагу можемо извршити библиотечком функцијом `find` декларисаном у заглављу `<algorithm>`.

```
int k = 0;
for (int i = 0; i < a.size(); i++) {
    // proveravamo da li se a[i] nalazi u delu niza a[0..k)
    auto b = begin(a), e = next(b, k);
    if (find(b, e, a[i]) == e) {
        a[k] = a[i];
        k++;
    }
}
a.resize(k);
ispis(a);
```

Задатак: Први и последњи приступ

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текснi задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Уместо да тражимо први или последњи елемент који задовољава неки услов, тражимо прво или последње појављивање датог елемента у низу. У тој варијанти можемо у два посебна низа сместити бројеве рачунара са којих су се ђаци логовали и имена ученика који су се логовали. У првом низу проналазимо прву тј. последњу позицију на којој се јавља број рачунара који нас занима а онда из другог низа са тих позиција читамо имена ученика и исписујемо их.

У језику С++ претрагу вредности унутар низа можемо извршити функцијом `find`. Њена прва два аргумента су итератори који ограничавају распон који се претражује, а трећи је вредност која се тражи. Функција враћа итератор који указује на прво појављивање пронађене вредности тј. на позицију иза краја распона ако елемент не постоји у низу. Ако се жели последње појављивање, потребно је користити обратне итераторе (`rbegin` и `rend`, како је описано у задатку [Последња негативна температура](#)).

Друга могућност је да чувамо један низ парова или структура, а да онда претрагу поново вршимо на основу услова, а не на основу вредности.

```
int x; cin >> x;
int n; cin >> n;
vector<int> a(n);
vector<string> t(n);
```

```

for (int i = 0; i < n; i++)
    cin >> a[i] >> t[i] >> ws;

auto prvo = find(begin(a), end(a), x);
if (prvo != end(a)) {
    auto poslednje = find(rbegin(a), rend(a), x);
    cout << t[distance(begin(a), prvo)] << endl;
    cout << t[distance(begin(a), prev(poslednje.base()))] << endl;
} else {
    cout << "nema" << endl;
}

```

Задатак: Негативан број

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Под претпоставком да су сви елементи смештени у низ претрагу можемо једноставно извршити библиотечком функцијом. У језику C++ једна могућност је да искористимо функцију `any_of` која проверава да ли неки елемент датог опсега задовољава жељени услов – у овом случај да је негативан. Функцији `any_of` ћемо као последњи аргумент проследити анонимну функцију која проверава да ли је број негативан.

```

int n;
cin >> n;
vector<double> a(n);
for (int i = 0; i < n; i++)
    cin >> a[i];

if (any_of(begin(a), end(a), [](int x){ return x < 0; }))
    cout << "da" << endl;
else
    cout << "ne" << endl;

```

Друга могућност је да употребимо функцију `find_if`. Функција враћа итератор на први елемент који задовољава дато својство или итератор непосредно иза краја низа, ако такав елемент постоји. Упоредивање резултата функције `find_if` са вредношћу `end` нам даје могућност да дамо тражени одговор.

```

int n;
cin >> n;
vector<double> a(n);
for (int i = 0; i < n; i++)
    cin >> a[i];

if (find_if(begin(a), end(a), [](int x){ return x < 0; }) != end(a))
    cout << "da" << endl;
else
    cout << "ne" << endl;

```

Задатак: Дељив бројевима од 1 до n

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење**Библиотечке функције**

У језику C++ функцијом `all_of` можемо проверити да ли сви елементи неког опсега задовољавају неко својство. Можемо направити низ свих бројева од 1 до n , а онда за сваки од њих проверити да ли дели број k . Као трећи аргумент функцији `all_of` можемо проследити анонимну функцију чији је параметар текући делилац,

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

међутим, пошто она мора да има приступ променљивој k , ту променљиву ћемо навести у листи променљивих којима анонимна функција може да приступи (унутар заграда []).

```
int n, k;
cin >> k >> n;
vector<int> d(n);
iota(begin(d), end(d), 1);
if (all_of(begin(d), end(d),
           [k](int i) { return k % i == 0; }))
    cout << "da" << endl;
else
    cout << "ne" << endl;
```

Задатак: Парно непарни

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текснi задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Ако претпоставимо да су сви бројеви смештени у низ (пошто не знамо колико ће их бити, најбоље је да употребимо вектор у језику С++), тада задатак веома елегантно можемо решити и библиотечким функцијама заснованим на линеарној претрази. У језику С++ функцијом `find_if` можемо пронаћи позицију првог непарног елемента, а затим функцијом `all_of` можемо проверити да ли су сви преостали елементи (од те позиције, па до краја низа) непарни.

```
bool непаран(int x) {
    return x % 2 != 0;
}

int main() {
    vector<int> a;
    int x;
    while (cin >> x)
        a.push_back(x);

    auto првиНепаран = find_if(begin(a), end(a), непаран);
    bool ОК = all_of(првиНепаран, end(a), непаран);
    cout << (ОК ? "da" : "ne") << endl;

    return 0;
}
```

Задатак: Погођена комбинација

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текснi задатак.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Библиотечке функције

Једнакост два низа можемо испитати и библиотечким функцијама. У језику С++ на располагању имамо функцију `equal` која може да испита једнакост садржаја две серије података (два низа, вектора, ниске или њихових делова).

```
int n;
cin >> n;
int a[MAX], b[MAX];
for (int i = 0; i < n; i++)
    cin >> a[i];
for (int i = 0; i < n; i++)
```



```
cin >> b[i];
cout << (equal(begin(a), next(a, n), begin(b)) ? "da" : "ne") << endl;
```

Задатак: Ниска палиндром

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текстови задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Библиотечке функције

Краћи код (али не обавезно и ефикасно решење) могуће је постићи применом библиотечких функција.

Једна идеја је да се примети да је ниска палиндром ако и само ако је једнака низу који се добија њеним обраћањем. Постоји неколико начина да се обрне ниска `s` у језику `C++`. Један је да се направи његова копија `t`, а онда да се употреби функција `reverse(t.begin(), t.end())`, међутим, још бољи начин је да се употреби конструктор којем се прослеђују реверзни итератори (они ниску обилазе од десна на лево) тј. `string(s.rbegin(), s.rend())`.

```
string s;
cin >> s;
if (s == string(s.rbegin(), s.rend()))
    cout << "da" << endl;
else
    cout << "ne" << endl;
```

Једна оптимизација овог приступа је да се примети да није потребно поредити целу ниску, већ само проверити да ли је њена прва половина једнака обратной другој половини (при чему се у случају непарног броја карактера средишњи карактер не урачунава ни у једну од две половине). У језику `C++` део ниске (подниску) можемо издвојити методом `substr`.

```
string s;
cin >> s;
if (string(s.begin(), next(s.begin(), s.size()/2)) ==
    string(s.rbegin(), next(s.rbegin(), s.size()/2)))
    cout << "da" << endl;
else
    cout << "ne" << endl;
```

Обраћање ниске на начин које смо приказали узрокује изградњу нове ниске у меморији, што је меморијски непотребно захтевно и тиме су претходна решења заснована на библиотечкој функционалности неефикаснија него она ручно имплементирана.

У језику `C++` се проблем може елегантно разрешити ако се употреби функција `equal` која проверава да ли су два сегмента елемената низа једнака (слично као у задатку [Погођена комбинација](#)). Функцији `equal` се као други итератор може проследити и реверзни итератор, чиме се постиже да се приликом поређења други сегмент обилази уназад, с десна на лево. Тиме се услов палиндрома своди на `equal(s.begin(), s.end(), s.rbegin())`. Ово решење је веома ефикасно.

```
string s;
cin >> s;
if (equal(s.begin(), s.end(), s.rbegin()))
    cout << "da" << endl;
else
    cout << "ne" << endl;
```

Задатак: Обраћање низа

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текстови задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Коришћење библиотечких функција и колекција

У језику C++ на располагању имамо функцију `reverse` која обрће део ограничен са два дата итератора (први указује на почетак интервала који се обрће, а други непосредно иза краја интервала). Ако се обрће део низа `a` између позиција `p` и `q`, то се може постићи помоћу `reverse(next(a, p), next(a, q+1))` или са `reverse(a+p, a+q+1)`. Применом ове процедуре мења се редослед података у оригиналном низу (функција је `void` и не враћа резултат).

```
// ucitavamo niz
int a[50000];
int n = unosNiza(a);

// obrcemo delove niza
while (true) {
    // ucitavamo interval [p, q] koji treba obrnuti
    int p, q;
    cin >> p >> q;
    // ako je interval prazan, prekidamo postupak
    if (p > q)
        break;
    // vrsimo obrtanje
    reverse(next(a, p), next(a, q+1));
}

// ispisujemo rezultat
ispisNiza(a, n);
```

Уместо низа у језику C++ можемо за смештање елемената низа користити и вектор (`vector`). Обртање елемената вектора врши се истом функцијом `reverse` при чему би позив овај пут био `reverse(next(a.begin() + p), next(a.begin() + q + 1))`. Применом ове процедуре мења се редослед података у оригиналном вектору (функција је `void` и не враћа резултат).

Задатак: Бројање у игри жмурке

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текст задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Једнакост растојања

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текст задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Степен

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текст задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Просечан раст цена

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текст задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Редни број максимума

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текст задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Најмањи круг

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шексџи задатак.*
Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Победник у три дисциплине

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шексџи задатак.*
Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Разлика сума до мах и од мах

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шексџи задатак.*
Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Просечно одступање од минималног

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шексџи задатак.*
Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Најближи просеку

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шексџи задатак.*
Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Берза

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шексџи задатак.*
Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Палиндромска реченица

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шексџи задатак.*
Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Прва негативна температура

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шексџи задатак.*
Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Последња негативна температура

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види шексџи задатак.*
Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

5.1.6 Сортирање

Сортирање података један је од најпроучаванијих проблема у рачунарству. С обзиром на огромне примене сортирања њему ће посебна пажња бити посвећена у другом тому ове збирке.

Задатак: Сортирање бројева

Напиши програм који уређује (сортира) низ бројева неопадајуће (сваки наредни мора да буде већи или једнак од претходног).

Улаз: Са стандардног улаза се уноси број n ($1 \leq n \leq 5 \cdot 10^4$) а затим и n природних бројева мањих од $2n$, сваки у посебном реду.

Издаз: На стандардни издаз исписати учитане бројеве у сортираном редоследу.

Пример

Улаз	Издаз
5	1
3	1
1	3
6	6
8	8
1	

Решење

Сортирање вектора библиотечком функцијом

У језику С++ сортирање је најбоље вршити функцијом `sort`, која прима два итератора - први који указује на почетак дела низа тј. вектора који се сортира, а други указује иза последњег елемента у делу низа који се сортира. Када се сортира цео вектор `a` ти итератори се могу добити помоћу `a.begin()` и `a.end()`.

Сложеност овог приступа је $O(n \log n)$.

```
// ucitavanje brojeva
int n;
cin >> n;
vector<int> a(n);
for (int i = 0; i < n; i++)
    cin >> a[i];

// sortiranje
sort(a.begin(), a.end());

// ispis rezultata
for (int i = 0; i < n; i++)
    cout << a[i] << endl;
```

Сортирање низа библиотечком функцијом

Када се сортира низ `a` са `n` елемената почетни итератор је `a`, а завршни се може добити као `next(a, n)` или евентуално `a+n`.

Сложеност овог приступа је $O(n \log n)$.

```
// ucitavanje brojeva
int n;
cin >> n;
int a[MAX];
for (int i = 0; i < n; i++)
    cin >> a[i];

// sortiranje
sort(a, next(a, n));

// ispis rezultata
for (int i = 0; i < n; i++)
    cout << a[i] << endl;
```

Сортирање селекцијом (SelectionSort)

Алгоритам сортирања селекцијом (*SelectionSort*) низ сортира тако што се у првом кораку најмањи елемент доводи на прво место, у наредном кораку се најмањи од преосталих елемената доводи на друго место и тако редом док се низ не сортира.

Најједноставнији начин да се алгоритам реализује је да се у спољној петљи разматрају редом позиције i у низу од прве до претпоследње, да се у унутрашњој петљи разматрају све позиције j од $i + 1$ па до последње и да се у телу унутрашње петље размењују елементи на позицији i и j , ако су тренутно у наопаком редоследу.

У најгорем случају и број поређења и број размена је $O(n^2)$.

```
void selection_sort(vector<int>& a) {
    for (int i = 0; i < a.size(); i++) {
        for (int j = i + 1; j < a.size(); j++)
            if (a[i] > a[j])
                swap(a[i], a[j]);
    }
}
```

Бољи начин имплементације је да се за сваку позицију i прво пронађе позиција најмањег елемента у делу низа од позиције i до краја, и да се онда елемент на позицији i размени са пронађеним минимумом (тима се број размена значајно смањује). Позицију најмањег елемента проналазимо на исти начин као у задатку **Редни број максимума**.

Нагласимо да је чак и са овом бољом имплементацијом овај алгоритам прилично неефикасан и није га пожељно примењивати осим у случају релативно кратких низова (мада се и они сортирају једноставније библиотечком функцијом).

Алгоритам врши увек $O(n)$ размена и $O(n^2)$ поређења, па је сложеност квадратна.

```
void selection_sort(vector<int>& a) {
    // na svaku poziciju i dovodimo najmanji element iz dela niza na
    // pozicijama [i, n)
    for (int i = 0; i < a.size(); i++) {
        // pozicija najmanjeg elementa u delu [i, n)
        int pmin = i;
        for (int j = i + 1; j < a.size(); j++)
            // ako je element na poziciji j manji od elementa na poziciji
            // pmin, nasli smo novi minimum, pa azuriramo vrednost pmin
            if (a[pmin] > a[j])
                pmin = j;
        // razmenjujemo element na poziciji i sa minimumom
        swap(a[i], a[pmin]);
    }
}
```

Сортирањем уметањем (InsertionSort)

Алгоритам сортирања уметањем (*InsertionSort*) заснива се на томе да се сваки наредни елемент у низу умеће на своје место у сортираном префиксу низа испред њега.

Најједноставнија имплементација је таква да се у спољној петљи разматрају све позиције од друге, па до краја низа и да се уметање врши тако што се сваки елемент размењује са елементима испред себе, све док се испред њега не појави елемент који није мањи од њега или док елемент не стигне на почетак низа.

У најгорем случају је и број поређења и број размена квадратни тј. $O(n^2)$.

```
void insertion_sort(vector<int>& a) {
    // element sa pozicije i umecemo u vec sortirani prefiks na
    // pozicijama [0, i)
    for (int i = 1; i < a.size(); i++) {
        // dok je element na poziciji j manji od njemu prethodnog
        // razmenjujemo ih
        for (int j = i; j > 0 && a[j] < a[j-1]; j--)
            swap(a[j], a[j-1]);
    }
}
```

У бољој имплементацији избегавају се размене током уметања. Елемент који се умеће се памти у привременој променљивој, затим се сви елементи мањи од њега померају за једно место удесно и на крају се запамћени елемент уписује на своје место.

У овом алгоритму се не врше размене, него појединачне доделе. Размена захтева три доделе, па се овом

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

оптимизацијом фаза довођења елемента на своје место може убрзати око три пута. Међутим, у најгорем случају је и број поређења и број додела квадратни тј. $O(n^2)$.

```
void insertion_sort(vector<int>& a) {
    // element sa pozicije i umesimo u vec sortirani prefiks na
    // pozicijama [0, i)
    for (int i = 1; i < a.size(); i++) {
        // pamtimo element na poziciji i
        int tmp = a[i];
        // sve elemente koji su veci od njega pomeramo za
        // jednu poziciju udesno
        int j;
        for (j = i - 1; j >= 0 && a[j] > tmp; j--)
            a[j + 1] = a[j];
        // stavljamo element sa pozicije i na njegovo mesto
        a[j + 1] = tmp;
    }
}
```

Задатак: Сортирање такмичара

Дат је низ такмичара, за сваког такмичара познато је његово име и број поена на такмичењу. Написати програм којим се сортира низ такмичара нерастуће по броју поена, а ако два такмичара имају исти број поена, онда их уредити по имену у неоппадајућем поретку.

Улаз: У првој линији стандардног улаза налази се природан број n ($n \leq 50000$). У следећих n линија налазе се редом елементи низа. За сваког такмичара, у једној линији, налази се одвојени једним блатком симболом, његово име (дужине највише 20 карактера) и број поена (природан број из интервала $[0, 10000]$) које је такмичар освојио.

Изназ: На стандардни излаз исписати елементе уређеног низа такмичара, за сваког такмичара у једној линији приказати његово име и број поена, одвојени једним блатком симболом.

Пример

Улаз	Изназ
5	Milica 89
Maја 56	Jovan 78
Marko 78	Marko 78
Krsto 23	Maја 56
Jovan 78	Krsto 23
Milica 89	

Решење

Прво питање које је потребно разрешити је како чувати податке о такмичарима. Најприродније решење да се подаци о такмичару памте у структури `Takmicar` чији су елементи име такмичара (податак типа `string`) и број поена (податак типа `int`). За чување информација о свим такмичарима можемо онда употребити неку колекцију структура. Друге могућности су да се подаци чувају у два низа (низу имена и низу поена) или да се уместо структура користе парови тј. торке.

Сортирање је могуће извршити на разне начине (види задатак [Сортирање бројева](#)).

Сортирање селекцијом

Један од начина је да се ручно имплементира неки алгоритам сортирања (као пример, наводимо најједноставнију имплементацију алгоритма `SelectionSort`), међутим то је обично веома компликовано и неефикасно.

Сложеност сортирања селекцијом је $O(n^2)$.

```
#include <iostream>
#include <algorithm>
#include <string>
using namespace std;
```

```

struct Takmicar {
    string ime;
    int brojPoena;
};

bool uporedi(Takmicar A, Takmicar B) {
    if (A.brojPoena > B.brojPoena)
        return true;
    if (A.brojPoena < B.brojPoena)
        return false;
    return A.ime <= B.ime;
}

int main() {
    int n;
    cin >> n;

    Takmicar a[50000];
    for (int i = 0; i < n; i++)
        cin >> a[i].ime >> a[i].brojPoena;

    for (int i = 0; i < n - 1; i++) {
        int iMax = i;
        for (int j = i + 1; j < n; j++)
            if (uporedi(a[j], a[iMax]))
                iMax = j;
        swap(a[iMax], a[i]);
    }

    for (int i = 0; i < n; i++)
        cout << a[i].ime << " " << a[i].brojPoena << endl;

    return 0;
}

```

Библиотека функција сортирања

Најбољи и уједно и најједноставнији начин је употребити библиотеку функцију сортирања. У језику C++ то је функција `sort` (без обзира на то да ли се користи вектор или низ).

Све варијанте коришћења библиотеке функције у којој се поређење и размештање два елемента може извршити у константној сложености су квази-линеарне сложености $O(n \log n)$.

Функцији сортирања је потребно доставити и функцију поређења којом се заправо одређује редослед елемената након сортирања. Тело те функције поређења врши вишекритеријумско (лексикографско) поређење уређених двојки података, слично оном које смо видели у задатку [Пунолетство](#). Прво се пореди број поена и ако је број поена првог такмичара мањи функција поређења враћа `false` (јер он треба да иде иза другог такмичара), ако је већи враћа се `true` (јер он треба да иде испред другог такмичара), а ако су бројеви поена једнаки, прелази се на поређење имена. За то се може упоредити библиотечко абecedно (лексикографско) поређење ниски (слично као у задатку [Лексикографски минимум](#)). У језику C++ за то се могу употребити оператори поређења (`<`, `>`, `<=`, `>=`, `==`, `!=`).

У језику C++ постоји неколико начина да дефинишемо функцију поређења.

Именована функција поређења

Први је да се дефинише класична (глобална) функција која прима две структуре (податке о два такмичара) и која враћа вредност типа `bool` и то вредност `true` ако први такмичар треба да буде испред другог у коначном сортираном низу, тј. вредност `false` у супротном. Ефикасности ради, структуре које се пореде има смисла пренети преко константних референци. Та функција добија своје име (на пример, `poredi`) и то име се онда наводи као трећи параметар у позиву библиотечке функције `sort` (након два итератора који одређују распон који се сортира).

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

```
bool uporedi(const Takmicar& a, const Takmicar& b) {  
    ...  
}
```

```
sort(begin(a), end(a), uporedi);
```

Комплетно решење се може имплементирати на следећи начин.

```
#include <iostream>  
#include <algorithm>  
#include <string>  
using namespace std;  
  
struct Takmicar {  
    string ime;  
    int brojPoena;  
};  
  
bool uporedi(const Takmicar& A, const Takmicar& B) {  
    if (A.brojPoena > B.brojPoena)  
        return true;  
    if (A.brojPoena < B.brojPoena)  
        return false;  
    return A.ime <= B.ime;  
}  
  
int main() {  
    int n;  
    cin >> n;  
  
    Takmicar a[50000];  
    for (int i = 0; i < n; i++)  
        cin >> a[i].ime >> a[i].brojPoena;  
  
    sort(a, a+n, uporedi);  
  
    for (int i = 0; i < n; i++)  
        cout << a[i].ime << " " << a[i].brojPoena << endl;  
    return 0;  
}
```

Анонимна функција поређења

Други начин је да се уместо именоване функције, функција поређења дефинише као анонимна функција тј. као λ -израз. Параметри и повратна вредност те функције су исти као у случају именоване функције, при чему се повратни тип не мора експлицитно навести. Пошто функција поређења у овом случају не мора да користи друге податке осим података о такмичарима који се пореде, као параметре затворења те функције (параметри који се наводе у заградама []) нема потребе наводити ништа. Анонимна функција се наводи директно у позиву функције `sort` као њен трећи аргумент.

```
sort(a.begin(), a.end(),  
    [](const Takmicar& a, const Takmicar& b) {  
        ...  
    });
```

Комплетно решење се може имплементирати на следећи начин.

```
#include <iostream>  
#include <algorithm>  
#include <vector>  
#include <string>  
using namespace std;
```



```

struct Takmicar {
    string ime;
    int brojPoena;
};

int main() {
    int n;
    cin >> n;

    vector<Takmicar> a(n);
    for (int i = 0; i < n; i++)
        cin >> a[i].ime >> a[i].brojPoena;

    sort(a.begin(), a.end(),
        [] (const Takmicar& A, const Takmicar& B) {
            if (A.brojPoena > B.brojPoena)
                return true;
            if (A.brojPoena < B.brojPoena)
                return false;
            return A.ime <= B.ime;
        });

    for (int i = 0; i < n; i++)
        cout << a[i].ime << " " << a[i].brojPoena << endl;

    return 0;
}

```

Објекат упоређивач

Трећи начин је да се дефинише структура (или класа) која имплементира функционалност поређења два такмичара у оквиру своје методе `operator()`. Параметри и повратна вредност те функције је иста као и у претходним случајевима, а у позиву функције `sort` наводи се један објекат те структуре (или класе).

```

struct PoredjenjeTakmicara {
    bool operator() (const Takmicar& a, const Takmicar& b) {
        ...
    }
}

```

```
sort(a.begin(), a.end(), PoredjenjeTakmicara());
```

Природно је да такав објекат “упоређивач” буде објекат посебне структуре или класе, а могуће је додати га и некој већ постојећој структури (на пример, структури `Takmicar`). Комплетно решење се може имплементирати на следећи начин.

```

#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;

struct Takmicar {
    string ime;
    int brojPoena;
};

struct PoredjenjeTakmicara {
    bool operator() (const Takmicar& A, const Takmicar& B) {
        return A.brojPoena > B.brojPoena ||

```

5.1. ЈЕДНОДИМЕНЗИОНАЛНЕ КОЛЕКЦИЈЕ ПОДАТАКА

```
        (A.brojPoena == B.brojPoena && A.ime <= B.ime);
    }
};

int main() {
    int n;
    cin >> n;

    vector<Takmicar> a(n);
    for (int i = 0; i < n; i++)
        cin >> a[i].ime >> a[i].brojPoena;

    sort(a.begin(), a.end(), PoredjenjeTakmicara());

    for (int i = 0; i < n; i++)
        cout << a[i].ime << " " << a[i].brojPoena << endl;

    return 0;
}
```

Оператор поређења

Четврти начин је да се у самој структури која чува податке о такмичару дефинише метода под именом `operator<` којом се обезбеђује да се структуре А и В могу поредити помоћу `A < B`. Та метода има један параметар који представља другог такмичара, док је први такмичар она структура на којој се тај метод позива.

```
struct Takmicar {
    string ime;
    int brojPoena;

    bool operator<(const Takmicar& A) {
        ...
    }
};
```

```
sort(a.begin(), a.end());
```

Комплетно решење се може имплементирати на следећи начин.

```
#include <iostream>
#include <algorithm>
#include <string>
using namespace std;

struct Takmicar {
    string ime;
    int brojPoena;

    bool operator<(const Takmicar& A) {
        if (brojPoena > A.brojPoena)
            return true;
        if (brojPoena < A.brojPoena)
            return false;
        return ime <= A.ime;
    }
};

int main() {
    int n;
    cin >> n;
```

```

Takmicar a[50000];
for (int i = 0; i < n; i++)
    cin >> a[i].ime >> a[i].brojPoena;

sort(a, a+n);

for (int i = 0; i < n; i++)
    cout << a[i].ime << " " << a[i].brojPoena << endl;
return 0;
}

```

Коришћење парова или торки

Рецимо и да се за репрезентацију података могу користити парови тј. торке (слично као у задатку **Пунолетство**). У језику С++ парови су представљени типом `pair`, а торке типом `tuple`. Пошто се парови тј. торке подразумевано пореде лексикографски (прво прва компонента, а тек ако је прва компонента једнака, онда друга) и неопадајуће, функцију поређења није неопходно наводити. Зато је паметно парове организовати тако да се као прва компонента памти супротан број од броја освојених поена (да би се добио опадајући редослед броја поена), а као друга име такмичара.

```

#include <iostream>
#include <algorithm>
#include <string>
#include <utility>
#include <vector>
using namespace std;

int main() {
    int n;
    cin >> n;

    vector<pair<int, string>> a(n);
    for (int i = 0; i < n; i++) {
        string ime; int brojPoena;
        cin >> ime >> brojPoena;
        a[i] = make_pair(-brojPoena, ime);
    }

    sort(a.begin(), a.end());

    for (int i = 0; i < n; i++)
        cout << a[i].second << " " << -a[i].first << endl;

    return 0;
}

```

Задатак: Просек скокова

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види шексџи задатак.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Највреднији предмети

За сваки предмет који је на продају дата је шифра и цена. Купац има на располагању одређени износ динара и жели да купи што скупле предмете. Редом узима предмете почев од најскупљег, док има новца. Ако нема новца за најскупљи, узима најскупљи за који има новца. Приказати шифре предмета које купац купује и, ако му је остало, преостали износ новца. Напомена: ова стратегија не гарантује да ће предмети које купи бити укупно највеће могуће вредности (нпр. ако има 5 динара и ако су цене предмета 4, 3 и 2 динара, он ће купити предмет само предмет од 4 динара, а могао би да купи предмете од 3 и 2 динара).

5.2. КАРАКТЕРИ И НИСКЕ

Улаз: У првој линији стандардног улаза налази се износ новца (реалан број) који има купац, у другој број предмета, N , а затим се, у сваке две линије стандардног улаза, уносе, редом, шифра (ниска карактера) па цена (реалан број) предмета, свака у посебном реду, за свих N предмета.

Изназ: У свакој линији стандардног излаза исписују се шифре и цене купљених предмета (раздвојене размаком), ако их има. У последњој линији приказује се преостали износ новца, ако постоји.

Пример1

Улаз	Изназ
1250.75	predmet4 1125.5
5	predmet5 115.75
predmet1	9.50
1010.30	
predmet2	
357.35	
predmet3	
725.45	
predmet4	
1125.5	
predmet5	
115.75	

Пример2

Улаз	Изназ
10000	predmet3 5725.00
6	predmet1 3010.00
predmet1	predmet4 1265.00
3010	
predmet2	
3005	
predmet3	
5725	
predmet4	
1265	
predmet5	
2075	
predmet6	
385	

Задатак: Сортирање на основи растојања од О

Петра је проучавао градове које жели да посети. На мапи је нацртао координатни систем у којем се његов град налази у координатном почетку, а остали градови су одређени њиховим Декартовим координатама. Написати програм којим се низ градова сортира у неоппадајућем пореку на основу удаљености од Периног града (од најближег до најдаљег). Ако су два града подједнако удаљена од Периног, уредити их у нерастућем поретку прво по координати x па по координати y .

Улаз: У првој линији стандардног улаза налази се број градова, природан број n ($n \leq 50000$). У следећих n линија налазе се елементи низа: у свакој линији, налазе се, x и y координата једног града (два цела броја из интервала $[-10^3, 10^3]$ одвојена једним размаком).

Изназ: Стандардни излаз садржи елементе уређеног низа градова - за сваки град у једној линији приказати редом, његову координату x и y , раздвојене размаком.

Пример

Улаз	Изназ
4	1 -1
8 2	-2 -3
1 -1	2 -3
2 -3	8 2
-2 -3	

5.2 Карактери и ниске

5.2.1 Елементи програмског језика

Опишимо мало детаљније како се могу представити подаци текстуалног типа у програму. Текст се представља у облику *ниске* (engl. string) која се састоји од карактера.

У језику C++ је најбољи избор да користимо податак типа `string` (из заглавља `<string>`), јер тада на располагању имамо многе библиотечке операције и функције (већина су методе класе `string`).

Постоји могућност да се уместо података типа `string` користи низ карактера (низ елемената типа `char`) у коме се за ознаку краја користи карактер чији је ASCII кôд нула (за такве ниске се каже да су *терминисане завршном нулом*, односно, на енглеском језику *null terminated strings*). Тај начин рада са текстом је типичан за програмски језик C и много га је теже користити него рад са типом `string`, тако да га у решењима задатака у овој збирци нећемо користити.

Константне ниске се записују између двоструких наводника. На пример,

```
string s = "Zdravo svima";
```

Ниску која садржи једно или више појављивања неког карактера можемо добити на следећи начин.

```
string s(5, 'a'); // s sadrži tekst "aaaaa"
```

Ако је `str` променљива типа `string`, наредба `cin >> str;` учитава једну реч са стандардног улаза (до позиције прве белине, занемарујући белине на почетку речи). Све речи, до краја улаза, можемо учитавати на следећи начин:

```
string rec;
while (cin >> str)
    ...
```

Други начин да се ово постигне јесте да се употреби итератор улазног тока.

```
istream_iterator<string> pocetak(cin), kraj;
for (auto it = pocetak; it != kraj; it++) {
    string rec = *it;
    ...
}
```

Наредбом `getline(cin, str);` се у ниску `str` учитава цела линија текста (текст до преласка у нови ред или до краја улаза). Прелазак у нови ред се не смешта у ниску `str`. Функција враћа вредност која се када се употреби унутар услова наредбе `while` или `if` аутоматски конвертује у вредност `true` ако је читање било успешно тј. вредност `false` ако није (то се дешава ако се пре покушаја читања дошло до краја улаза). Тако се читање линија до краја улаза може остварити са:

```
string str;
while (getline(cin, str))
    ...
```

У заглављу `<string>` декларисано је мноштво метода помоћу којих се може радити са нискама (објектима типа `string`). Најважније су следеће:

- `size`, `length` – враћа дужину ниске тј. број њених карактера;
- `find`, `rfind` – враћа позицију првог тј. последњег појављивања неког карактера или подниске унутар ниске тј. специјалну вредност `string::npos` ако тражени карактер или подниска не постоје;
- `substr` – издваја поднику дате ниске, као параметре прихвата позицију на којој почиње подниска која се издваја и дужину подниске. Ако се не наведе дужина подниске, издваја се подниска која почиње на задатој позицији и протеже се до краја ниске;
- `replace` – мења део ниске другом ниском;
- `insert` – умеће ниску на дату позицију унутар друге ниске;
- `erase` – из текуће ниске брише се њена подниска која почиње на задатој позицији и задате је дужине.

Две ниске се могу надовезати коришћењем оператора `+` и резултат је нова ниска. Проширивање ниске неким суфиксом (додавање текста на њен десни крај) се може вршити оператором `+=`. Нагласимо да проширивање наредбом облика `str += s` може бити знатно ефикасније него наредбом облика `str = str + s`, јер се у другој понекад прво креира нова ниска у коју се копира садржај ниске `str` и ниске `s`, након чега би се тај нови садржај додељује ниски `s`, што доводи до велике неефикасности (ово је чувени пример лошег решења у програмирању познато под именом *Schlemiel the Painter's algorithm*).

Подаци типа `string` имају двојако тумачење. Са једне стране можемо их сматрати елементарним, атомичким подацима (на пример, видели смо да се могу "сабирати", слично бројевима). Са друге стране ниске се могу третирати и као колекције (низови) карактера и над њима је могуће спроводити исте алгоритме које смо спроводили над низовима и векторима. На пример, иако је `s` објекат типа `string`, а не низ или вектор, изразом `s[i]` може се одредити његов `i`-ти карактер (под претпоставком да је `i` мање од дужине ниске). Методама `begin` и `end`, односно истоименим функцијама могуће је добити итераторе на почетак и непосредно изнад краја ниске, које је могуће проследити функцијама које обрађују опсеге. На пример, `sort(s.begin(), s.end())` сортира све карактере ниске `s` (у абecedном поретку). Такође, кроз све карактере ниске могуће је проћи петљом облика

```
for (char c : str)
    ...
```

5.2.2 Подниске

Задатак: Презиме па име

Ученици који су заинтересовани на такмичењу су направили списак у електронском облику, али су на том списку увек писали прво име, па онда презиме. Међутим, електронски систем пријављивања захтева да се унесе прво презиме, па онда име. Напиши програм који поправља списак у складу са тим захтевом.

Улаз: Свака линија стандардног улаза, све до његовог краја, садржи име и презиме ученика (која су откуцана само коришћењем слова енглеске абецедe), раздвојене са тачно једним размаком. Напомена: при интерактивном тестирању крај улаза се може унети тастерима `ctrl+z` (Windows) тј. `ctrl+d` (Unix/Linux).

Изназ: На стандардни излаз исписати презимена и имена свих учитаних ученика раздвојене са по једним размаком, у истом редоследу у којем су учитани.

Пример

Улаз	Изназ
Petar Petrovic	Petrovic Petar
Ana Anic	Anic Ana
Jelena Jelenkovic	Jelenkovic Jelena
Mika Mikic	Mikic Mika

Решење

Потребно је читати линију по линију текста све док се не дође до краја улаза. Поступак је сличан ономе у задатку [Читање до краја улаза](#). У језику C++ линију текста са стандардног улаза у ниску `str` (променљиву типа `string`) можемо учитати помоћу `getline(cin, str)`.

Када имамо учитану линију са именом и презименом (рецимо да је резултат смештен у променљивој `ime_i_prezime`), потребно је да из ње издвојимо име и презиме (који су раздвојени размаком).

Један од начина да извршимо поделу је да се прво пронађе позиција размака, а онда да се издвоје делови ниске до позиције размака и после ње. У језику C++ позицију неког карактера у ниски можемо пронаћи методом `find` (на пример, `ime_i_prezime.find(' ')` проналази позицију размака). Издвајање делова ниске (подниски) се може вршити методом `substr` чији су параметри позиција на којој почиње подниска која се издваја и њена дужина. Ако се дужина изостави, подразумева се да се издваја подниска која почиње на наведеној позицији и која се простире до краја ниске. Тако се име може издвојити са `ime_i_prezime.substr(0, p)` док се презиме може издвојити са `ime_i_prezime.substr(p + 1)`. Када се издвоје име и презиме, резултат се може лако добити (тако што се испише прво презиме, затим један размак и на крају име).

```
string ime_i_prezime;
while (getline(cin, ime_i_prezime)) {
    int p = ime_i_prezime.find(' ');
    string ime = ime_i_prezime.substr(0, p);
    string prezime = ime_i_prezime.substr(p+1);
    cout << prezime << " " << ime << endl;
}
```

Задатак се још лакше може делити ако у библиотеци језика постоји функција која дели ниску на делове на основу неког сепаратора и враћа низ ниски. У том низу ће име бити на позицији 0, а презиме на позицији 1. Када се идвоје име и презиме, резултат се може лако добити (тако што се испише прво презиме, затим један размак и на крају име).

Задатак: Све подречи

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текстови задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Када је фиксиран сегмент $[i, j]$ подреч можемо издвојити и коришћењем библиотеке подршке за рад са нискама. У језику C++ можемо употребити методу `substr` у чијем позиву као аргументе наводимо почетну позицију i и дужину подречи која је једнака $j - i + 1$.

```
string s;
cin >> s;
for (int i = 0; i < s.size(); i++)
    for (int j = i; j < s.size(); j++)
        cout << s.substr(i, j - i + 1) << endl;
```

Задатак: Подниска

Написати програм којим се за две дате речи проверава да ли је друга садржана у првој, ако јесте одредити прву позицију на којој се друга реч појављује у првој.

Улаз: На стандардном улазу налазе се две речи свака у посебној линији. Свака реч има највише 20 карактера.

Излаз: На стандардном излазу приказати прву позицију (позиције су нумерисане од 0) на којој се налази друга реч у првој, ако се друга реч не налази у првој приказати -1.

Пример 1

Улаз Излаз
banana 1
ana

Пример 2

Улаз Излаз
branka -1
ana

Решење

Библиотечке функције претраге

Када смо речи у језику C++ учитали у две променљиве типа `string` (назовимо их `igla` и `plast`, јер је тако одмах јасно шта у чему тражимо) онда позицију првог појављивања речи `igla` у речи `plast` можемо одредити помоћу `plast.find(igla)`. Ако реч `plast` не садржи реч `igla` онда `find` враћа специјалну вредност `string::npos`.

```
// ucitavamo reci
string plast, igla;
cin >> plast >> igla;
// pretragu podniske vrsimo koriscenjem bibliotecke metode
size_t poz = plast.find(igla);
// ako niska nije nadjena, rezultat je string::npos
if (poz != string::npos)
    cout << poz << endl;
else
    cout << -1 << endl;
```

Имплементација претраге грубом силом

Осим коришћења библиотечких функција, решење је било могуће испрограмирати и “пешке” тако што ћемо дефинисати своју функцију претраге. Наравно, у реалној ситуацији је увек боље употребити библиотечке функције јер се код једноставније пише, мања је могућност грешке, а алгоритми на којима су засноване библиотечке функције су често напреднији од наивних алгоритама на којима се “пешке” решења понекад заснивају, какво је и ово које ћемо описати у наставку.

Размотримо, на пример, како бисмо у речи `ababcbab` тражили реч `abc`.

- Прво можемо проверити да ли се реч `abc` налази у речи `ababcbab` почевши од позиције 0. То радимо тако што поредимо редимо слова речи `abc` и слова речи `ababcbab` почевши од позиције 0, све док не пронађемо различито слово или док не дођемо до краја речи `abc` а не пронађемо разлику. У овом примеру разлика ће бити пронађена на позицији 2 (слова `a` на позицији 0 и `b` на позицији 1 ће бити једнака, а онда ће се слово `c` разликовати од слова `a`). Пошто смо нашли разлику пре него што смо стигли до краја речи `abc`.
- Настављамо претрагу тако што проверавамо да ли се можда реч `abc` налази у речи `ababcbab` почевши од позиције 1. Овај пут поредимо слово на позицији 0 у речи `abc` са словом на позицији 1 у речи `ababcbab` и одмах наилазимо на разлику, јер `a` није једнако `b`.

- Након тога претрагу настављамо тако што проверавамо да ли се реч `abc` налази у речи `ababcbab` почевши од позиције 2. Тада поредимо слово `a` на позицији 0 у речи `abc`, са словом `a` на позицији 2 у речи `ababcbab`, слово `b` на позицији 1 у речи `abc`, са словом `b` на позицији 3 у речи `ababcbab` и слово `c` на позицији 2 у речи `abc`, са словом `c` на позицији 4 у речи `ababcbab` и пошто не наилазимо на разлику, констатујемо да је подреч пронађена.

Ако бисмо, на пример, тражили реч `cba` у речи `ababcbab` тада бисмо по истом принципу кренули да је тражимо од позиције 0, затим 1 итд. Када реч `cba` не пронађемо ни тако да почиње од позиције 4 речи `ababcbab` (јер `cab` није исто што и `cba`) нема потребе да тражимо даље, јер иза позиције 4 нема довољно карактера да би се реч `abc` пронашла.

Дакле, за сваку позицију i у речи `plast`, од позиције 0, па док важи да је збир позиције i и дужине речи `igla` мањи или једнак дужини речи `plast` проверавамо да ли се реч `igla` налази у речи `plast` почевши од позиције i . То радимо тако што спроводимо алгоритам линеарне претраге који редом, за сваку позицију j у речи `igla` проверава да ли је слово у речи `igla` на тој позицији различито од слова на позицији $i + j$ у речи `plast`. Алгоритам претраге имплементирамо на неки од начина који смо приказали у задатку **Негативан број**. На пример, у петљи увећавамо j од нуле, па све док је j мање од дужине речи `igla` и док важи да су слова у речи `igla` на позицији j и речи `plast` на позицији $i + j$ једнака. Када се петља заврши, проверавамо зашто је прекинута. Ако утврдимо да је вредност j једнака дужини речи `igla`, можемо констатовати да разлика није пронађена, да се прво појављивање речи `igla` у речи `plast` налази на позицији i и функција може да врати резултат i (прекидајући тиме спољашњу петљу). У супротном је пронађено различито слово, и наставља се са извршавањем спољашње петље (i се увећава, и подниска се тражи од следеће позиције, ако евентуално није достигнут услов за завршетак спољашње петље).

Још један начин да имплементирамо претрагу је да уведемо логичку променљиву која бележи да ли је разлика пронађена и коју иницијализујемо на `false`. У сваком кораку унутрашње петље (петље по j) проверавамо да ли је су слова у речи `igla` на позицији j и речи `plast` на позицији $i + j$ једнака. Ако нису, логичкој променљивој постављамо вредност `true` и прекидамо унутрашњу петљу. Након унутрашње петље проверавамо да ли је вредност логичке променљиве остала `false` и ако јесте, функција може да врати вредност i .

```
// Trazimo "iglu u plastu" tj. trazimo poziciju u reci plast
// na kojoj se nalazi rec igla.
// Ako takva pozicija ne postoji, funkcija vraca -1.
int podniska(string igla, string plast) {
    // za svaku poziciju u reci plast takvu da do kraja
    // ima bar onoliko karaktera koliko ima u reci igla
    for (size_t i = 0; i + igla.size() <= plast.size(); i++) {
        // proveravamo da li se rec igla nalazi u reci plast
        // pocvsi od pozicije i
        bool razliciti = false;
        for (size_t j = 0; j < igla.size(); j++) {
            // proveravamo da li je j-to slovo reci igla razlicito od j-tog
            // slova reci plast krenuvsi od pozicije i
            if (plast[i+j] != igla[j]) {
                razliciti = true;
                break;
            }
        }
        // ako smo dosli do kraja reci igla, tada smo je pronasli
        if (!razliciti)
            return i;
    }
    // nismo nasli rec igla i vracamo -1
    return -1;
}

int main() {
    string plast, igla;
    cin >> plast >> igla;
    cout << podniska(igla, plast) << endl;
}
```



```
return 0;
}
```

Задатак: Подниске

Напиши програм који одређује колико пута се дата ниска јавља као подниска друге дате ниске. Више појављивања подниске се могу преклапати.

Улаз: Са стандардног улаза се уносе две ниске, свака у посебном реду. Прва ниска има највише 10 карактера, а друга највише 1000.

Излаз: На стандардни излаз исписати тражени број појављивања.

Пример 1

Улаз	Излаз
aba	4
abababcsababac	

Пример 2

Улаз	Излаз
aa	3
aaaa	

Решење

Претрага од сваке позиције

Директан алгоритам да се овај задатак реши је да се покуша претрага подниске од сваке позиције кренувши од нулте, па до оне позиције на којој се десни крај ниске поклапа са десним крајем подниске. Обилазак позиција, дакле, тече све док је збир текуће позиције и дужине подниске мањи или једнак од дужине ниске. За сваку такву позицију проверавамо да ли на њој почиње подниска и ако почиње, увећавамо бројач иницијално постављен на нулу.

Проверу да ли се подниска налази у датој ниски кренувши од дате позиције можемо урадити на разне начине. Један начин је да издвојимо део ниске од дате позиције помоћу функције `substr` и да га упоредимо са датом подниском помоћу оператора `==`.

Овај приступ може бити неефикасан јер се непотребно сваки пут гради нова ниска пре него што се изврши поређење.

У језику C++ можемо употребити функцију `equal` која проверава једнакост два опсега исте дужине.

За поређење ниски (тј. њихових подниски) можемо и сами испрограмирати функцију која коришћењем линеарне претраге (попут оне у задатку **Погођена комбинација**) проверава да ли постоји карактер у којима се две ниске разликују, при чему функција уз ниске `s` и `t` које се пореде, може добити и индексе позиција `ps` и `pt` од којих ће се поређење вршити и дужину `n` делова ниски који се пореде. У петљи чији бројач `i` креће од 0 и тече све док је мањи од `n` пореде се карактери на позицијама `s[ps + i]` и `t[pt + i]` и ако су различити функција враћа `false`. Ако се петље заврше без пронађене разлике делови су једнаки и функција враћа `true`.

Уместо засебне функције такву претрагу можемо реализовати и у главној функцији у телу петље која набраја потенцијалне позиције почетка, али нам је тада за имплементацију линеарне претраге потребна помоћна логичка променљива и неки начин да унутрашњу петљу прекинемо када се наиђе на први различити карактер.

```
string niska, podniska;
cin >> podniska >> niska;
int brojPojavljanja = 0;
for (int p = 0; p + podniska.length() <= niska.length(); p++)
    if (equal(podniska.begin(), podniska.end(), next(niska.begin(), p)))
        brojPojavljanja++;
cout << brojPojavljanja << endl;
```

Библиотека претрага подниске

Библиотека функција за претрагу подниске обично има додатни параметар којим се контролише позиција од које претрага треба да крене. То важи за методу `find` у језику C++, која враћа позицију првог пронађеног појављивања подниске (које није испред наведене позиције) или специјалну ознаку `string::npos` када нема појављивања подниске који нису испред наведене позиције. Сваки пут када се пронађе подниска, увећава се бројач, а нова претрага креће од позиције непосредно иза позиције где је подниска пронађена.

Рецимо да би се пажљивом анализом ниске која се тражи то могло оптимизовати. На пример, ако смо нашли ниску `abcabcabc` унутар шире ниске, ново појављивање те подниске не треба тражити од наредне позиције

5.2. КАРАКТЕРИ И НИСКЕ

(позиције првог карактера *b*), већ од треће позиције на којој се појављује карактер *a* (то је најдужи суфикс који је уједно и префикс ниске која се тражи). Ово је основа алгоритма КМП о ком ће бити више речи у наредном тому ове збирке.

```
string niska, podniska;
cin >> podniska >> niska;
int brojPojavljivanja = 0;
int p = 0;
while ((p = niska.find(podniska, p)) != string::npos) {
    brojPojavljivanja++;
    p++;
}
cout << brojPojavljivanja << endl;
```

Задатак: Реч Франкенштајн

Напиши програм који од унете речи прави нову реч састављену из делова почетне речи. На пример, за унето

```
dabrakabmга
6 2
3 3
1 1
0 3
9 2
```

програм гради и исписује реч *abrakadabra* (прво иду 2 карактера кренувши од позиције 6 тј. *ab*, затим 3 карактера од позиције 3 тј. *гак*, затим 1 карактер од позиције 1 тј. *а*, затим три карактера од позиције 0 тј. *dab* и на крају два карактера од позиције 9 тј. *га*).

Улаз: У првој линији стандардног улаза је ниска који представља дату реч (дужине највише 50000 карактера) и која садржи само слова енглеске абецете. У наредним линијама (њих највише 10000) се уносе парови: позиција у датој ниски и дужина подниске (позиција и дужина исправно задају подниску).

Излаз: Реч која се добија наведеним поступком.

Пример

<i>Улаз</i>	<i>Излаз</i>
маганаакkabbikора	коракabana
13 3	
16 1	
8 3	
3 3	

Решење

Реч Франкенштајн која се тражи добија се лепљењем мањих делова оригиналне ниске, учитане из прве линије текста. Учитавање једне линије у језику C++ вршимо функцијом `getline`.

Након тога у петљи читавамо позицију и број карактера сваког дела, све док не дођемо до краја улаза. То радимо на начин веома сличан оном описаном у задатку **Читање до краја улаза**. За учитану позицију и дужину, део оригиналне ниске (подниску) издвајамо на начин који смо већ срели у задатку **Презиме па име**. У језику C++ користимо методу `substr` чији су параметри управо позиција и дужина подниске која се издваја.

Сваки издвојени део лепимо на ниску у којој се чува до сада формиран део решења. Њу на почетку, пре петље, иницијализујемо на празну ниску, а онда јој у телу петље, на крај, дописујемо део који смо управо одредили. Дописивање на десну страну ниске можемо вршити оператором `+=`. Приметимо велику сличност изградње ниске на овај начин са алгоритмом сабирања серије бројева описаним у задатку **Збир *n* бројева** – и у овом случају добијамо решење “сабирањем” мањих делова.

Подсетимо се да је у језику C++ јако значајно за надовезивање користити оператор `+=`, уместо оператора `+` и оператора `=` тј. облик `s += x` уместо `s = s + x`, да би се избегло стално креирање нових ниски и тиме добио много ефикаснији програм.

Задатак: Избацивање поднски

Написати програм којим се одређује текст који се добија брисањем из датог текста свих појављивања подречи из датог скупа. Бришу се прво сва појављивања прве речи, затим друге, треће и тако до краја. Тај поступак се понавља све док се текст њиме мења. Приликом брисања свих појављивања речи поступак се исцрпно понавља све док је брисање могуће.

На пример, за текст `babrakadabrabrr` и скуп речи `{br, ka, aa}`, прво се исцрпно брише `br` и добија се `baakadaa`, затим се брише `ka` и добија се `baadaa`, затим се брише `aa` и добија се `brd`. Након тога се креће из почетка, исцрпно се брише `br` и добија `d`, покушава се са брисањем `ka` и `aa` које не успева, пролази се кроз трећи круг у којем реч остаје иста и пријављује се резултат `d`.

Улаз: У првој линији стандардног улаза је текст, дужине највише $5 \cdot 10^5$ карактера из којег се бришу речи које се читавају из наредних линија. Речи се читавају до краја стандардног улаза и има их највише 10^5 , а свака је дужине највише 10 карактера.

Излаз: Текст који се добија након брисања подречи које садржи.

Пример

<i>Улаз</i>	<i>Излаз</i>
<code>babrakadabrabrr</code>	<code>d</code>
<code>br</code>	
<code>ka</code>	
<code>aa</code>	

Решење

У главном програму читавамо линију текста која ће се мењати, а затим и низ поднски које ће бити брисане. Пошто не знамо колико их је, сместићемо их у динамички низ. Након тога, у петљи ћемо пролазити кроз колекцију поднски, брисати једну по једну (позивом помоћне функције), чувајући, при том, у логичкој променљивој информацију да ли је дошло до неког брисања. Поступак ћемо понављати (помоћу спољашње петље) све док се не деси да се прође кроз целу колекцију поднски, а да не дође ни до једног брисања (што ћемо знати на основу вредности логичке променљиве).

Приметимо да је најпогоднија петља за ово петља `do-while`, која се, како смо видели, мало ређе користи од петљи са провером услова на почетку (петљи `while` и њој веома сродне петље `for`).

Кључни елемент решења је функција која исцрпно уклања сва појављивања дате подниске из дате ниске.

У језику C++ на располагању нам стоји метода `erase` која као параметар прима позицију и број карактера и брише поднику која почиње на тој позицији и садржи тај број карактера. Да бисмо обрисали конкретну поднику прво треба да пронађемо њено појављивање што можемо да урадимо коришћењем методе `find` о којој је било речи у задатку **Подниска**. У петљи вршимо претрагу памћећи пронађену позицију све док она не добије вредност `string::npos`. У услову петље истовремено додељујемо вредност променљивој која чува позицију и проверавамо да ли је та додељена вредност различита од `string::npos` (тај идиом смо описали у задатку **Абеледно огледало**). У телу петље позивамо функцију `erase` и вршимо брисање. Пошто функција мора да врати податак о томе да ли је извршена нека замена, уводимо логичку променљиву коју пре петље иницијализујемо на вредност `false` (ништа до тада још није обрисано), а коју у телу петље (у коме се врши брисање) постављамо на вредност `true` (нешто јесте обрисано). На крају функције враћамо вредност те променљиве.

```
// brise sva pojavljivanja stringa s u stringu str i vraca podatak
// o tome da li je nesto obrisano
bool obrisi(string& str, const string& s) {
    size_t poz;
    // u pocetku nije nista obrisano
    bool obrisano = false;
    // dok god uspevam da pronadjemo podstring s na nekoj poziciji poz
    while ((poz = str.find(s)) != string::npos) {
        // brisemo podstring s sa pozicije poz
        str.erase(poz, s.length());
        // registrujemo da je nesto obrisano
        obrisano = true;
    }
}
```

5.2. КАРАКТЕРИ И НИСКЕ

```
// vraćamo podatak o tome da li je nešto obrisano
return obrisano;
}

int main() {
    // učitava znake u string niska do znaka za kraj linije
    string tekst;
    getline(cin, tekst);

    // učitavamo sve stringove u niz (vektor)
    vector<string> reci;
    string rec;
    while (cin >> rec)
        reci.push_back(rec);

    // brisemo delove dok god ima promena
    bool obrisano;
    do {
        obrisano = false;
        // brisemo iscrpno pojavljivanja svih podniski redom
        for (auto rec: reci)
            if (obrisi(tekst, rec))
                // belezimo da je nešto bilo obrisano
                obrisano = true;
    } while (obrisano);

    // ispisujemo konacni rezultat
    cout << tekst << endl;
}
```

Задатак: Подела линије на речи

Напиши програм који издваја све речи из учитане линије текста. Речи се састоје само од малих слова и раздвојене су по једним размаком.

Улаз: Са стандардног улаза се учитава једна линија текста.

Излаз: На стандардни излаз се исписује свака реч у посебној линији (без размака).

Пример

Улаз	Излаз
ko gano gani dve sreće grabi	ko gano gani dve sreće grabi

Решење

Ручна имплементација поделе ниске на речи

Обраду речи је могуће урадити и тако што се цела линија текста учита у једну ниску, а онда се из те ниске издваја једна по једна реч. Током тог поступка одржавамо променљиву `start` у којој се чува почетак наредне речи и коју можемо иницијализовати на нулу. Сваку наредну реч проналазимо тако што пронађемо позицију наредног размака `poz` у линији и издвојимо карактере између позиције `start` и позиције `poz` (укључујући прву, али не и другу). Након сваке издвојене речи, позиција `start` се помера на позицију `poz+1`. Последња реч је специфична по томе јер иза ње нема размака, међутим, и она се може издвојити на униформан начин као и све остале, ако се након учитавања линије на њен крај дода један размак.

За проналажење првог појављивања неког карактера (или подниске) `s` почевши од неке позиције `p` у ниски `s` у језику C++ можемо користити позив `s.find(c, p)`, који враћа индекс првог појављивања ако се `s` пронађе,

односно специјалну вредност `string::npos` ако се `s` не пронађе. Издвајање дела ниске `s` између позиција `a` и `b` укључујући карактер на позицији `a`, а не укључујући онај на позицији `b` можемо извршити позив `s.substr(a, b - a)` (аргументи су почетна позиција и број карактера који се издвајају).

Издавање речи може трајати све док се након позиције `start` може пронаћи размак или док позиција `start` не пређе дужину линије из које се речи издвајају.

Рецимо да се, ако је то потребно, прва реч може издвојити и посебно, испред петље.

```
// ucitavamo liniju teksta koja sadrzi reci
string str;
getline(cin, str);
// linija se prosiruje prazninom da bi se sve reci
// izdvajale na uniforman nacin
str += " ";
int start = 0;
// radi dok ne stignes do kraja stringa str
while (start < str.size()) {
    // pronalazimo narednu prazninu krenuvsi od pozicije start
    int pos = str.find(' ', start);
    // izdvajamo rec od pozicije start do praznine (ne ukljucujuci nju)
    string tekucaRec = str.substr(start, pos - start);
    cout << tekucaRec << endl;
    // startna pozicija sledece pretrage
    start = pos + 1;
}
```

5.2.3 Поређење ниски

Задатак: Лексикографски поредак

Речи се у речницима ређају по абечеди. Напиши програм који упоређује две речи на основу тог поретка.

Улаз: Са стандардног улаза се учитавају две речи, свака у посебном реду.

Излаз: На стандардни излаз исписати `-1` ако је прва у речнику испред друге, `1` ако је друга испред прве тј. `0` ако су речи једнаке.

Пример

Улаз	Излаз
abcde	-1
abcefg	

Решење

Библиотечка подршка

У језику `C++` лексикографско поређење ниски се добија применом уобичајених релацијских оператора `<`, `<=`, `>`, `>=` и `==`.

```
string rec1, rec2;
cin >> rec1 >> rec2;
if (rec1 < rec2)
    cout << -1 << endl;
else if (rec1 > rec2)
    cout << 1 << endl;
else
    cout << 0 << endl;
```

Ручна имплементација лексикографског поређења

Функцију лексикографског поређења која нам је потребна можемо имплементирати и самостално. Функција ће се заснивати на алгоритму који смо, на пример, анализирали у задатку **Победник у три дисциплине**.

Начин да се лексикографско поређење ручно имплементира је да се пореде карактери на истим позицијама у две ниске, кренувши од почетка све док се не дође до прве разлике или краја неке од две ниске. Ако се наиђе

на карактер који се разликује, може се вратити разлика између кодова та два карактера. Ако се дође до краја неке ниске, може се вратити разлика између дужина две ниске (краћа ниска је тада префикс дуже, па је по дефиницији лексикографски испред друге).

```
int poredi(const string& s1, const string& s2) {
    for (int i = 0; i < s1.size() && i < s2.size(); i++)
        if (s1[i] != s2[i])
            return s1[i] - s2[i];
    return s1.size() - s2.size();
}
```

Задатак: Лексикографски поредак без разлике између великих и малих слова

Речи се у речницима ређају по абеди. Напиши програм који упоређује две речи на основу тог поретка занемарујући разлику између малих и великих слова.

Улаз: Са стандардног улаза се читавају две речи, свака у посебном реду.

Излаз: На стандардни излаз исписати -1 ако је прва реч у речнику испред друге, 1 ако је друга испред прве тј. 0 ако су речи једнаке.

Пример

Улаз	Излаз
aBcdE	-1
abCEfg	

Решење

Поређење помоћу библиотечких функција

У језику C++, на жалост, не постоји стандардна библиотека подршка за лексикографско поређење које не прави разлику између великих и малих слова. Подаци типа `string` се могу поредити релацијским операторима `<`, `<=` и слично, поређење ће бити лексикографско, али ће се разликовати велика и мала слова. У програмском језику C (па и у програмском језику C++, када се ниске представе низовима карактера) се за ово понекад користе функције `strcasestr` и `strcmpi`, међутим, оне нису стандардизоване, тако да није сигурно да ће на свим платформама радити. Уместо тога, постоји општија функција `lexicographical_compare` која врши лексикографско поређење два опсега (низова, вектора, ниски) који се задају итераторима на почетак и иза краја. Овој функцији је могуће проследити и функцију (било именовану, било анонимну) која се користи да упореди два текућа елемента тих сегмената и која треба да врати `true` само ако је текући елемент првог распона мањи од текућег елемента другог распона. Када се пореде ниске, та функција прима два карактера, преводи евентуална мала слова у велика (позивом функције `toupper`) и онда их пореди оператором `<`.

```
bool manja(const string& rec1, const string& rec2) {
    return lexicographical_compare(begin(rec1), end(rec1),
                                   begin(rec2), end(rec2),
                                   [](char c1, char c2) {
                                       return toupper(c1) < toupper(c2);
                                   });
}

int main() {
    string rec1, rec2;
    cin >> rec1 >> rec2;
    if (manja(rec1, rec2))
        cout << -1 << endl;
    else if (manja(rec2, rec1))
        cout << 1 << endl;
    else
        cout << 0 << endl;

    return 0;
}
```

Још један начин, додуше мало неефикаснији, је да се приликом поређења обе ниске преведу у велика или мала слова. Нажалост, у језику С++ ни за то не постоји библиотечка функција, али једноставно ју је имплементирати (на пример, у петљи је могуће проћи кроз све елементе ниске и на сваки применити функцију `toupper` или је могуће позвати библиотечку функцију `transform(str.begin(), str.end(), ::toupper)`).

```
string rec1, rec2;
cin >> rec1 >> rec2;
transform(begin(rec1), end(rec1), begin(rec1), ::toupper);
transform(begin(rec2), end(rec2), begin(rec2), ::toupper);
if (rec1 < rec2)
    cout << -1 << endl;
else if (rec2 < rec1)
    cout << 1 << endl;
else
    cout << 0 << endl;
```

Ручна имплементација функције поређења

Функција поређења се имплементира готово на идентичан начин као у задатку **Лексикографски поредак**. Једина разлика је то што се пре поређења карактера они преводе у велика или мала слова (најбоље библиотечком функцијом `toupper` коју смо увели у задатку **Трансформација карактера**).

```
// poredi dve reci leksikografski, ne praveci razliku izmedju velikih
// i malih slova
int poredi(const string& str1, const string& str2) {
    // dok u obe reci ima karaktera
    for (int i = 0; i < str1.size() && i < str2.size(); i++) {
        // poredimo tekuci karakter, transformisuci ga u veliko slovo
        char c1 = toupper(str1[i]), c2 = toupper(str2[i]);
        // ako su karakteri razliciti, leksikografski odnos niski je
        // odredjen njihovim odnosom
        if (c1 != c2)
            return c1 - c2;
    }
    // procitali smo kracu rec (ili obe), a nismo naisli na karakter
    // koji se razlikuje - odnos je odredjen duzinama reci
    return str1.size() - str2.size();
}
```

Задатак: Лексикографски минимум

Напиши програм који одређује која би од неколико датих речи требало да буде наведена прва у речнику ако се не прави разлика између великих и малих слова (за ту реч кажемо да је лексикографски испред осталих речи, тј. да је лексикографски најмања међу свим речима).

Улаз: Са стандардног улаза се учитава линија текста која садржи више речи раздвојених са по тачно једним размаком (у линији постоји бар једна реч, а иза последње речи нема размака).

Излаз: На стандардни излаз исписати реч из уčitане линије која је лексикографски испред осталих речи.

Пример

Улаз	Излаз
Ko gano gani dve srece grabi	dve

Решење

Задатак захтева да се одреди минимум низа учитаних речи у односу на релацију лексикографског поретка. Да би се написао тражени програм потребно је учитавати реч по реч из дате линије, поредити две речи лексикографски и пронаћи минимум серије речи у односу на лексикографски поредак.

Начини да се из линије издвоје појединачне речи описан је у задатку **Подела линије на речи**.

Алгоритам одређивања минимума

Одређивање најмање речи вршимо коришћењем уобичајеног алгорита за одређивање минимума серије (који смо видели, на пример, у задатку **Најнижа температура**). Најмању реч иницијализујемо на прву реч коју прочитамо (она сигурно постоји, по претпоставци задатка), а затим у петљи читамо остале речи, поредимо их са минималном (коришћењем лексикографског поретка који занемарује величину слова који је приказан у задатку **Лексикографски поредак без разлике између великих и малих слова**) и ако је текућа реч мања од до тада минималне, ажурирамо минималну.

```
// ucitavamo prvu rec i inicijalizujemo minimum na nju
string minRec;
cin >> minRec;
// ucitavamo rec po rec, dok ih ima
string tekucaRec;
while (cin >> tekucaRec)
    // ako je tekuca rec leksikografski ispred najmanje, azuriramo
    // najmanju rec
    if (manja(tekucaRec, minRec))
        minRec = tekucaRec;
// prijavljujemo najmanju rec
cout << minRec << endl;
```

Проналажење минимума библиотечким функцијама

Минимум можемо одређивати и библиотечким функцијама.

Ако су у језику C++ речи представљене у колекцији кроз коју се може пролазити итератором, онда најмањи елемент можемо одредити функцијом `min_element` која враћа итератор на најмањи елемент. То може бити случај или ако учитамо све речи у вектор или ако користимо итератор улазног тока `istream_iterator` (као што смо радили у задатку **Читање до краја улаза**).

Функцији `min_element` се као трећи параметар прослеђује функција поређења која пореди два елемента и враћа логичку вредност `true` ако и само ако је њен први параметар мањи од другог (у овом случају шаљемо функцију `manja` која је приказана у задатку **Лексикографски поредак без разлике између великих и малих слова**).

```
// odredjujemo leksikografski najmanju rec ucitanu sa ulaza
// zanemarujuci razliku izmedju velikih i malih slova
cout << *min_element(istream_iterator<string>(cin),
                    istream_iterator<string>(),
                    manja) << endl;
```

5.2.4 Трансформисање ниски

Задатак: Цезаров кôд

Задата је реч састављена од малих слова римске абецеде (која је подскуп енглеске). Потребно је шифровати односно дешифровати поруку. Цезарова шифра је тип шифре замењивања у коме се свако слово текста који се шифрује мења словом абецеде које се добије померањем полазног слова за одређени број места, циклично по абецеди. На пример, ако се врши померање за три места, слово `a` би се шифровало словом `d`, `b` словом `e`, итд., док би се слово `w` шифровало словом `z`, слово `x` словом `a`, слово `y`, словом `b` и слово `z` словом `c`. На пример, реч `paradaј` би се шифровала са `sdsdјdm`.

Улаз: У првој линији стандардног улаза налази се реч не дужа од 100 слова, у другој линији се налази цео број N , који представља помак ($1 \leq N < 26$), а у трећој цео број S који представља смер шифровања. Ако је $S = 1$ потребно је шифровати, а ако је $S = 2$ потребно је дешифровати реч.

Изназ: У првој линији стандардног излаза приказати реч која се добије након шифровања тј. дешифровања задате речи.

Пример 1

<i>Улаз</i>	<i>Изназ</i>
abcdefghijklmnopqrstuvwxy	defghijklmnopqrstuvwxyzabc
3	
1	

Пример 2

<i>Улаз</i>	<i>Излаз</i>
twornuvkremkp	rumplstinckin
2	
2	

Решење

Централна операција у решењу овог задатка је кодирање појединачног карактера. Кодирање ниске се своди на кодирање сваког њеног карактера. То је веома једноставно урадити ручно у петљи, а могуће је урадити и помоћу библиотечких функција.

У језику C++ трансформацију ниске је могуће урадити и применом функције `transform` декларисане у заглављу `<algorithm>`.

Приметимо сличност са задатком [Абецедно огледало](#) у којем се такође вршило шифровање трансформацијама појединачних карактера. Основна разлика у односу на тај задатак је то што се у овом задатку померај учитава након текста, тако да је неопходно учитати цео текст пре него што се крене са његовим шифровањем, па је неопходно радити са нискама и није могуће учитавати и обрађивати један по један карактер.

Промена појединачног карактера се заснива на аритметичким операцијама над њиховим кодовима (ASCII кодовима у језику C++), слично као што је то приказано у задатку [Абецедно огледало](#). Шифровање подразумева циклично померање удесно по абецеди, што подразумева увећавање кода карактера за дати померај, водећи рачуна о томе да резултат не пређе карактер `z` (ако пређе, од добијеног кода је потребно одузети 26, што је број карактера у абецеди тј. разлика између кодова слова `z` и слова `a`). Дешифровање подразумева циклично померање улево по абецеди, што подразумева умањивање кода карактера за дати померај, водећи рачуна о томе да резултат не постање мањи од кода карактера `a` (ако се то догоди, на добијени код је потребно додати 26). Приликом увећавања тј. умањивања кодова карактера треба водити рачуна да резултат може да пређе опсег карактерског типа (ако је тип `char` означен онда се, на пример, приликом кодирања карактера `z` који има ASCII код 122, јако лако добије прекорачење максималне вредности 127 - довољно је да је померај већи од 5). Зато је приликом израчунавања кодова карактера пожељно прећи на тип `int`.

```
string rec;
cin >> rec;
int pomeraj;
cin >> pomeraj;

int smer;
cin >> smer;
if (smer == 1) { // sifrovanje
    for (int i = 0; i < rec.size(); i++) {
        // sifrujemo slovo na poziciji rec[i]
        int sifrovanKod = rec[i] + pomeraj;
        if (sifrovanKod > 'z')
            sifrovanKod -= 26;
        rec[i] = (char)sifrovanKod;
    }
} else { // desifrovanje
    for (int i = 0; i < rec.size(); i++) {
        // desifrujemo slovo na poziciji rec[i]
        int sifrovanKod = rec[i] - pomeraj;
        if (sifrovanKod < 'a')
            sifrovanKod += 26;
        rec[i] = (char)sifrovanKod;
    }
}
cout << rec << endl;
```

Краћи начин да се изврши кодирање карактера је да се употреби модуларна аритметика. Кључно место при одређивању резултата има редни број карактера у абецеди који се може добити одузимање кода карактера `a` од кода карактера који се обрађује. Померање онда подразумева сабирање са померајем или одузимање помераја по модулу 26. Начин да се то уради описан је у задацима [Операције по модулу](#) и [Монопол](#). Можемо приметити да одузимање помераја одговара додавању супротног помераја, што значи да се и шифровање и де-

шифровање може извршити истом функцијом (приликом дешифровања њој се проследи негативна вредност помераја). Сабирањем по модулу добија се редни број траженог карактера у абеди, док се сам карактер добија сабирањем са кодом карактера а. Дакле, централна формула која нам помаже да одредимо код који се добија шифровањем тј. дешифровањем карактера с је 'а' + (с - 'а' + померај + 26) % 26'. Број 26 је додат да би се осигурало да је број на који се примењује остатак при дељењу позитиван (што је објашњено у задатку **Монопол**).

```
char sifrujSlovo(char c, int pomeraj) {
    return 'a' + (c - 'a' + pomeraj + 26) % 26;
}
```

```
void sifrujRec(string& rec, int pomeraj) {
    for (int i = 0; i < rec.size(); i++)
        rec[i] = sifrujSlovo(rec[i], pomeraj);
}
```

```
int main() {
    string rec;
    cin >> rec;
    int pomeraj;
    cin >> pomeraj;
    int smer;
    cin >> smer;
    if (smer == 1) // sifrovanje
        sifrujRec(rec, pomeraj);
    else // desifrovanje
        sifrujRec(rec, -pomeraj);
    cout << rec << endl;
    return 0;
}
```

5.2.5 Запис бројева и бројевних израза

Задатак: Бројевне основе

Написати програм којим се унети природан број приказује у бинарном, окталном и хексадекадном систему.

Улаз: У првој линији стандардног улаза налази се природан број n ($0 \leq n \leq 2 \cdot 10^9$).

Изназ: У првој линији стандардног излаза налази се бинарни запис, у другој линији октални запис и у трећој линији хексадекадни запис броја n . У хексадекадном запису користити велика слова.

Пример

Улаз	Изназ
234	11101010
	352
	EA

Решење

Природан број n се може превести у запис у било којој другој основи применом алгоритма дуалног Хорнеровој шеми који је заснован на итеративном дељењу са основом, а који смо већ приказали, на пример, у задатку **Број и збир цифара броја** (а раније, на пример, и у задатку **Октални бројеви**). Цифре броја се одређују као остаци при дељењу са основом. Заиста, ако се број n у основи b записује цифрама $(a_k a_{k-1} \dots a_2 a_1 a_0)_b$, тада је $n = a_k b^k + a_{k-1} b^{k-1} + \dots + a_2 b^2 + a_1 b + a_0$. Зато је последња цифра a_0 једнака $n \bmod b$, док вредност $n \operatorname{div} b = a_k b^{k-1} + a_{k-1} b^{k-2} + \dots + a_2 b + a_1$ представља запис броја који се добија брисањем последње цифре из записа броја n тј. запис броја $(a_k a_{k-1} \dots a_2 a_1)_b$. Ако вредност n заменимо вредношћу $n \operatorname{div} b$, наредна цифра a_1 се одређује као остатак при дељењу нове вредности броја n основом b , док се n опет мења вредношћу $n \bmod b$. Поступак се понавља све док вредност променљиве n не постане 0. Дакле, цифре одређујемо у петљи у чијем сваком кораку одређујемо остатак $n \bmod b$ и вредност n мењамо вредношћу $n \operatorname{div} b$. Вредност $n = 0$ представља специјалан случај, због којег је тело петље потребно извршити бар једном, и стога је боље користити петљу са провером услова на крају.

```
do {
    c = n % b; // poslednja cifra u zapisu u osnovi b
    n /= b;   // uklanjamo poslednju cifru iz zapisa
    ...
} while (n > 0);
```

Приметимо да смо на овај начин одредили низ цифара броја у обратном редоследу и да бисмо их исписали у исправном редоследу потребно је некако их упамтити. Додатно, ако је основа већа од 10, цифре се могу представљати и словима А, В, С итд. Зато је пожељно да сваку цифру коју одредимо претворимо у одговарајући карактер. Резултат можемо памтити или у податку типа `string` или у обичном низу карактера (наравно, прво решење је једноставније и више је у духу савремених језика, док је коришћење низа карактера углавном наслеђено из традиције језика С који не препознаје засебан тип `string`).

Конверзију цифре у карактер можемо остварити посебном функцијом: ако је вредност цифре мања од 10, онда карактер који је представља добијамо њеним сабирањем са ASCII кодом карактера 0, а у супротном карактер добијамо сабирањем разлике цифре и броја 10 са ASCII кодом карактера А (сличну технику баратања кодовима карактера видели смо, на пример, у задатку [Трансформација карактера](#)).

Према захтеву овог задатка, циљне основе су бинарна, октална и хексадекадна тако да у главном програму вршимо конверзију у сваку од те три основе и исписујемо резултат.

```
// datu numericku vrednost cifre (broj od 0 do 15) prevodi u
// karakter i to 0, 1, ..., 9, A, B, ..., F
char cifraUZnak(int x) {
    if (x < 10)
        return x + '0';
    else
        return x - 10 + 'A';
}

// odredjuje zapis broja n u osnovi osnova
string prevod(int n, int osnova) {
    string s = "";
    do {
        s = cifraUZnak(n % osnova) + s;
        n /= osnova;
    } while (n > 0);
    return s;
}

int main() {
    // učitavamo broj i odredjujemo mu binarni, oktalni i heksadekadni
    // zapis
    int n;
    cin >> n;
    cout << prevod(n, 2) << endl;
    cout << prevod(n, 8) << endl;
    cout << prevod(n, 16) << endl;
}
```

Свака нова цифра која се одреди, тј. њој придружени карактер треба додати на почетак ниске у коју се акумулира резултат. За то је могуће употребити оператор + којим се врши надовезивање ниски, међутим, вреди на овом месту напоменути да је у случају дугачких ниски додавање карактера на почетак постојеће ниске неефикасна операција коју треба избегавати. У овом случају ради се о веома кратким нискама, тако да овај проблем не долази до изражаја.

Ако се уместо податка типа `string` користи обичан низ карактера, онда је сваки нови карактер најбоље додати на крај низа, а када се све цифре одреде, обрнути садржај низа (начини да се ово уради описани су у задатку [Обртање низа](#)). Када се користи низ карактера, стандардни начин у језику С++ (наслеђен из језика С) да се означи његов крај је да се употреби терминална нула (карактер чији је ASCII код 0). Напоменимо и да је уместо обртања низа било могуће исписати га унатраг, но то решење је веома специфично и није увек

применљиво.

```
// obrce niz karaktera s duzine d
void obrni(char s[], int d) {
    for (int i = 0, j = d-1; i < j; i++, j--)
        swap(s[i], s[j]);
}

// odredjuje cifre u zapisu broja n u osnovi osnova i smesta ih u niz
// karaktera s koji je terminisan nulom
void prevod(int n, int osnova, char s[]) {
    int d = 0;
    do {
        s[d++] = cifraUznak(n % osnova);
        n /= osnova;
    } while (n > 0);
    obrni(s, d);
    s[d++] = '\0';
}

int main() {
    // ucitavamo broj i odredjujemo mu binarni, oktalni i
    // heksadekadni zapis
    int n;
    cin >> n;
    // 32 karaktera i terminalna nula
    char s[32 + 1];
    prevod(n, 2, s);
    cout << s << endl;
    prevod(n, 8, s);
    cout << s << endl;
    prevod(n, 16, s);
    cout << s << endl;
    return 0;
}
```

Библиотечке функције

На крају, поменимо да је за ове бројевне основе решење могуће веома једноставно добити и библиотечким функцијама.

У језику C++ манипулатор излазног тока `setbase(8)` проузрокује испис бројева у основи 8. Тако, на пример, `cout << setbase(8) << n << endl` проузрокује да се вредност броја n испише октално. Слично је и са основом 16 (`setbase(16)`). Нажалост, `setbase(2)` не даје жељени резултат и конверзију у бинарни запис морамо урадити на други начин. Једноставан начин да се добију бинарне цифре у запису броја је да се употреби структура података `bitset<>`. Тако, на пример, `bitset<32>(n).to_string()` даје ниску која садржи 32 бинарне цифре броја 8. По условима задатка потребно је још уклонити водеће нуле (обраћајући при том пажњу на специјални случај броја 0).

```
// ucitavamo broj n
int n;
cin >> n;

// odredjivanje binarnog zapisa
bitset<32> bits(n);
string str_bits = bits.to_string();
// uklanjamo vodece nule
// pozicija prve jedinice
auto first1 = str_bits.find('1');
// izdvajamo deo od prve jedinice ili postavljamo "0" ako te jedinice nema
str_bits = first1 != string::npos ? str_bits.substr(first1) : "0";
```

```
cout << str_bits << endl;

// ispis u osnovi 8
cout << setbase(8) << n << endl;
// ispis u osnovi 16, sa velikim heksadekadnim ciframa
cout << setbase(16) << uppercase << n << endl;
```

Задатак: Spreadsheet колоне

У програмима за табеларна израчунавања (Microsoft Excel, LibreOffice Calc и слично) колоне су обележене словима и то као A, B, ..., Z, AA, AB, ..., AZ, BA, BB, ..., ZZ, AAA, ... Напиши програм који омогућава конверзију редног броја (од 1 па навише) у текстуалну ознаку колоне и обратно.

Улаз: Са стандардног улаза се уноси број n ($1 \leq n \leq 10$), а затим n ниски (свака у посебном реду) које садрже велика слова енглеског алфабета (највише 5). Након тога се уноси n бројева (сваки у посебном реду) из интервала од 1 до 12356630.

Излаз: На стандардни излаз исписати прво n бројева, па затим n ниски (свака у посебном реду) које одговарају унетим нискама тј. бројевима.

Пример

Улаз	Излаз
3	4
D	28
AB	12356630
ZZZZZ	R
18	CV
100	BDWGN
1000000	

Решење

Ако слову А придружимо вредност 1, слову В вредност 2, слову С вредност 3 и тако све до слова Z којем придружимо вредност 26, можемо сматрати да се овде ради о запису броја у основи 26, са разликом да се не користи нула. Дакле, број се записује у облику $a_n \cdot 26^n + a_{n-1} \cdot 26^{n-1} + \dots + a_1 \cdot 26 + a_0$, при чему су све цифре a_i између 1 и 26. Заиста, колона AA има редни број $1 \cdot 26 + 1 = 27$, док, на пример, колона CBA има редни број $3 \cdot 26^2 + 2 \cdot 26 + 1 = 2081$. Имајући ово у виду, словну ознаку колоне можемо превести у њен редни број коришћењем Хорнерове схеме (слично као, на пример, у задатку [Вредност једноставног израза](#)). Бројевна вредност сваке цифре се може добити тако што се од њеног кода одузме код карактера 'A' и на то дода 1. Баратање кодовима карактера видели смо, на пример, у задатку [Трансформација карактера](#).

У обратном смеру, мало ћемо модификовати алгоритам одређивања цифара у датој бројевној основи који, подсетимо се, одређује последњу цифру операцијом остатка при дељењу бројевном основом и уклања је операцијом целобројног дељења (исто као, на пример, у задатку [Збир цифара четвороцифреног броја](#) или [Број и збир цифара броја](#)). Редни број се може разложити на $a_n \cdot 26^n + a_{n-1} \cdot 26^{n-1} + \dots + a_1 \cdot 26 + a_0$, при чему је вредности цифара a_i између 1 и 26. Ако се од броја одузме 1, последња цифра је између 0 и 25 и може се одредити одређивањем остатка при дељењу бројем 26. Ако се на тај остатак дода код карактера 'A', добија се код последњег карактера у словној ознаци колоне. Одређивањем целобројног количника при дељењу са 26 уклања се последња цифра и добија се број $a_n \cdot 26^{n-1} + a_{n-1} \cdot 26^{n-2} + \dots + a_1$, при чему је вредност свих цифара a_i између 1 и 26. Ово је проблем исте структуре као полазни и одређивање његових цифара a_i може се вршити на исти начин (одузимањем броја 1, одређивањем остатака при дељењу и целобројним дељењем основом 26).

Рецимо да је изградња ниске узастопним додавањем карактера на њен почетак генерално веома неефикасна операција. Ипак, пошто се у задатку ради са веома кратким нискама, нећемо обраћати пажњу на то.

```
int uBroj(string s) {
    int broj = 0;
    for (char c : s)
        broj = broj * 26 + (c - 'A' + 1);
    return broj;
}
```

```

string odBroja(int broj) {
    string rez = "";
    do {
        broj--;
        rez = string(1, 'A' + broj % 26) + rez;
        broj /= 26;
    } while (broj > 0);
    return rez;
}

int main() {
    int m;
    cin >> m;
    for (int i = 0; i < m; i++) {
        string s;
        cin >> s;
        cout << uBroj(s) << endl;
    }
    for (int i = 0; i < m; i++) {
        int broj;
        cin >> broj;
        cout << odBroja(broj) << endl;
    }
    return 0;
}

```

Задатак: Вредност једноставног израза

Напиши програм који израчунава вредност једноставног аритметичког израза (нпр. $123*456$).

Улаз: Са стандардног улаза учитава се једна линија текста која садржи једноставни аритметички израз добијен применом операција сабирања, одузимања или множења на два природна броја. Учитани текст не садржи размаке. Напомена: вежбе ради, учитати целу ниску која садржи израз, а затим је рашчланити на делове.

Издаз: На стандардни издаз исписати вредност израза (она је сигурно између -10^9 и 10^9).

Пример

Улаз	Издаз
32+73	105

Решење

Ако ниска *s* садржи учитани израз, потребно је рашчланити је на два операнда и оператор који се налази између њих.

Први задатак је наћи позицију оператора. Један начин да се то уради је да се употреби библиотека функција. У језику C++ то може бити метода `find_first_of` која као аргумент добија ниску и у ниски на којој је позвана проналази позицију првог карактера који се налази у ниски наведеној као аргумент.

Након проналажења позиције *p* на којој се налази оператор, потребно је издвојити део ниске испред оператора (он садржи први операнд) и део ниске иза оператора. То је опет могуће учинити библиотечким функцијама.

У језику C++ метода `substr` из дате ниске издваја део који почиње на датој позицији (наведеној као први аргумент) и садржи дати број карактера (наведен као други аргумент). Ако се други аргумент изостави, врши се издвајање до краја ниске. Ову методу употребљавали смо и у задатку [Реч Франкенштајн](#).

На крају остаје питање како од низа карактера (тј. ниске) који садржи искључиво цифре доћи до вредности броја. Опет нам на располагању стоје библиотечке функције. У језику C++ то је функција `stoi` (скраћено од `string to integer`).

```

string s;
cin >> s;

size_t pozicijaOperatora = s.find_first_of("+-*");

```

```

string operand1 = s.substr(0, pozicijaOperatora);
char oper = s[pozicijaOperatora];
string operand2 = s.substr(pozicijaOperatora + 1);

int a = stoi(operand1), b = stoi(operand2);
int rezultat = 0;
switch (oper) {
case '+':
    rezultat = a + b;
    break;
case '-':
    rezultat = a - b;
    break;
case '*':
    rezultat = a * b;
    break;
}

cout << rezultat << endl;

```

Наравно, уместо библиотечких функција можемо направити ручну имплементацију свих релевантних алгоритама, мада је решење уз употребу библиотечких функција свакако много једноставније и боље.

Претрагу је могуће имплементирати и ручно (коришењем алгоритама линеарне претраге, попут, на пример, оног у задатку [Негативан број](#)).

Методу за издвајање ниске можемо имплементирати тако што жељени распон карактера копирамо из једне ниске у другу. У језику C++ можемо резултујућу ниску на почетку алоцирати на жељену дужину (методом `resize`) а затим у њу ископирати жељене карактере (тима се избегава неефикасност до које може доћи сталним проширивањем резултујуће ниске за по један карактер).

Ручна имплементација заснована је на Хорнеровој шеми (слично као у задатку [Збир два троцифрена](#)). У почетку се резултат иницијализује на нулу, ниска цифара се обилази слева надесно (од позиције нула, па до краја) и у сваком кораку се претходни резултат множи са 10 и на њега додаје вредност текуће цифре. Вредност цифре се може добити од кода карактера његовим умањивањем за код карактера 0 (сличну технику баратања са ASCII кодовима видели смо и у задацима [Трансформација карактера](#) и [Абецедно огледало](#)).

```

int string_to_int(const string& s) {
    int n = 0;
    for (char c : s)
        n = 10 * n + (c - '0');
    return n;
}

int pozicijaOperatora(const string& s) {
    for (int i = 0; i < s.size(); i++)
        if (s[i] == '+' || s[i] == '-' || s[i] == '*')
            return i;
    return -1;
}

string podstring(const string& s, int pocetak, int kraj) {
    string rezultat;
    rezultat.resize(kraj - pocetak + 1);
    for (int i = pocetak, j = 0; i <= kraj; i++, j++)
        rezultat[j] = s[i];
    return rezultat;
}

int main() {
    string s;

```

```

cin >> s;

size_t poz = pozicijaOperatora(s);
string operand1 = podstring(s, 0, poz - 1);
char oper = s[poz];
string operand2 = podstring(s, poz + 1, s.size() - 1);

int a = string_to_int(operand1), b = string_to_int(operand2);
int rezultat = 0;
if (oper == '+')
    rezultat = a + b;
else if (oper == '-')
    rezultat = a - b;
else if (oper == '*')
    rezultat = a * b;

cout << rezultat << endl;
return 0;
}

```

Задатак: Арапски у римски

Напиши програм који преводи унети арапски број у римски.

Улаз: Са улаза се уноси природан број n ($1 \leq n \leq 2000$)

Излаз: На стандардни излаз се исписује римски запис броја n .

Пример

Улаз	Излаз
1978	MCMLXXVIII

Решење

Главна функција конверзије издваја цифру по цифру броја n операцијом целобројног остатка и уклања је операцијом целобројног количника при дељењу са 10, кренувши од цифре јединица (слично као у задатку [Збир цифара четвороцифреног броја](#)). Свака тако добијена цифра се конвертује у римски запис помоћном функцијом, прослеђујући јој симболе s_1 , s_5 и s_{10} у зависности од тежине текуће цифре и тај запис се додаје испред досадашњег изграђеног резултата (који креће од празне ниске и проширује се на лево). Поступак се завршава када се обраде све цифре броја n тј. када се он целобројним дељењем сведе на нулу. Ако се то не деси ни након уклањања цифре стотине, преостали број (то је број хиљада) се конвертује у римски тако што се тај број пута на почетак резултата дода симбол М.

```

// zapis jedne rimske cifre - simbol1, simbol5 i simbol 10 odredjuju
// da li se radi i cifri jedinica, desetica ili stotina
string rimskaCifra(int c, char simbol1, char simbol5, char simbol10) {
    if (c < 4) // npr. "", "I", "II", "III"
        return string(c, simbol1);
    else if (c == 4) // npr. "IV"
        return string(1, simbol1) + string(1, simbol5);
    else if (c < 9) // npr. "V", "VI", "VII", "VIII"
        return string(1, simbol5) + string(c-5, simbol1);
    else // npr. "IX"
        return string(1, simbol1) + string(1, simbol10);
}

// prevodi dati broj u rimski zapis
string arapskiURimski(int n) {
    string rezultat = "";
    // cifra jedinica
    rezultat = rimskaCifra(n % 10, 'I', 'V', 'X');
    n /= 10; // uklanjamo je
}

```



```

// ako nema vise cifara vracamo rezultat
if (n == 0) return rezultat;
// cifra desetica
rezultat = rimskaCifra(n % 10, 'X', 'L', 'C') + rezultat;
n /= 10; // uklanjamo je
// ako nema vise cifara vracamo rezultat
if (n == 0) return rezultat;
// cifra stotina
rezultat = rimskaCifra(n % 10, 'C', 'D', 'M') + rezultat;
n /= 10; // uklanjamo je
// ako nema vise cifara vracamo rezultat
if (n == 0) return rezultat;
// dodajemo hiljade na pocetak rezultata i vracamo ga
return string(n, 'M') + rezultat;
}

int main() {
// ucitavamo broj
int n;
cin >> n;
// prevodimo ga u rimski zapis i ispisujemo rezultat
cout << arapskiURimski(n) << endl;
return 0;
}

```

Задатак: Римски у арапски

Напиши програм који конвертује римске у арапске бројеве.

Улаз: Једина линија стандардног улаза садржи један римски број из интервала од 1 до 2000.

Излаз: На стандардни излаз исписати унети број записан у уобичајеном (арапском) запису.

Пример

Улаз	Излаз
MCMCLXXVIII	1978

Решење

Овај задатак је дуалан задатку [Арапски у римски](#), али им се решења прилично разликују. Вредност исправно записаног римског броја се може одредити тако што читамо његове цифре с лева на десно и на текући збир додељујемо вредност сваке цифре (М има вредност 1000, D вредност 500, С вредност 100, L вредност 50, X вредност 10, V вредност 5 и I вредност 1). Једини изузетак од овог правила је случај када се у запису појави мања испред веће цифре и тада се од текућег збира та цифра одузима. На пример, израчунајмо вредност записа MCMCLXXIV. На основу претходне дискусије, вредност овог броја је $1000 - 100 + 1000 + 50 + 10 + 10 - 1 + 5$ тј. 1974. Резултат се иницијализује на нулу. Прва цифра је М иза које није већа цифра, па се резултат увећава за 1000. Након тога наступа случај у којем следи цифра С која је мања од наредне цифре М, тако да се од збира одузима вредност 100 и добија се вредност 900. Након тога се на збир додаје 50, па два пута 10 и добија се збир 1970, док се не наиђе на цифру I која је мања од њој наредне цифре V. Зато се од збира одузима вредност 1 и добија се 1969, пре него што се на крају дода вредност последње цифре V и добије коначан резултат 1974.

Пролаз кроз све римске цифре датог римског броја вршимо у склопу једне петље у којој индексна променљива i мења своју вредност од нуле, па све док је мања од дужине ниске којом је записан римски број. У телу петље се за сваку цифру проверава да ли непосредно иза ње постоји цифра која је од ње већа (при том се мора водити рачуна и о томе да текућа цифра није последња). Ако таква цифра постоји, онда се текући збир умањи за вредност текуће цифре, у супротном се на збир додаје вредност текуће цифре. У сваком случају индексна променљива се увећава за 1 (прескаче се текућа и прелази се на наредну цифру, ако она постоји).

Вредност дате цифре можемо одредити гранањем (најбоље наредбом `switch`, слично као, на пример, у задатку [Број дана у месецу](#)).

```
// vrednost svake rimske cifre
int cifra(char c) {
    switch(c) {
        case 'M': return 1000;
        case 'D': return 500;
        case 'C': return 100;
        case 'L': return 50;
        case 'X': return 10;
        case 'V': return 5;
        case 'I': return 1;
    }
    return 0;
}

int rimskiUArapski(string rimski) {
    // rezultat - vrednost rimski zapisanog broja
    int arapski = 0;
    // prolazimo kroz sve cifre rimskog broja
    for (int i = 0; i < rimski.size(); i++) {
        // ako neposredno iza tekuće postoji cifra veća od nje
        if (i + 1 < rimski.size() && cifra(rimski[i]) < cifra(rimski[i+1]))
            // rezultat umanjujemo za vrednost tekuće cifre
            arapski -= cifra(rimski[i]);
        else
            // inace rezultat uvecavamo za vrednost tekuće cifre
            arapski += cifra(rimski[i]);
    }
    // vracamo konacan rezultat
    return arapski;
}
```

Види групаџија решења овој загађка.

5.3 Пресликавања

Често у програму имамо потребу да представимо пресликавање тј. функцију која одређеним *кључевима* једнозначно придружује одређене *вредности*. На пример, желимо да сваком ученику придружимо просек оцена или да сваком месецу придружимо број дана у том месецу. Таква пресликавања можемо представити посебном врстом низова који се називају *асоцијативни низови* (каже се и *речници* или *мапе*), а у неким посебним случајевима и класичним низовима.

5.3.1 Класични низови у функцији пресликавања

Када је скуп кључева интервал природних бројева облика $[0, n)$ или када се кључеви једноставно могу трансформисати у тај облик, тада је осим речником пресликавање могуће представити и класичним низом. Вредност придружену кључу i памтићемо на позицији i у низу. У наредних неколико задатака приказаћемо употребу низова у улози пресликавања.

Задатак: Назив месеца

Овај загађак је поновљен у циљу увежбавања различитих техника решавања. Види тџекст загађка.

Покушај да загађак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Потребно је извршити класификацију на основу једне од 12 могућих вредности.

Веома језгровит начин укључује коришћење низа који садржи називе месеца. Такав низ одређује пресликавање којим се вредност одређује на основу кључа. Пошто су кључеви бројеви месеца, а вредности њихова имена, кључеви су између 1 и 12 и у низу називе можемо поређати почевши од позиције 1.

```
int mesec;
cin >> mesec;
string meseci[] =
    {"", "januar", "februar", "mart", "april", "maj", "jun",
     "jul", "avgust", "septembar", "oktobar", "novembar", "decembar"};
cout << meseci[mesec] << endl;
```

Задатак: Успех ученика

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текстови задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Уместо гранања, могуће је да се поруке које треба исписати за сваки успех смештају у низ и да се испише одговарајући елемент низа.

```
double prosek;
cin >> prosek;
int uspeh = floor(prosek + 0.5);
string uspesi[] =
    {"nedovoljan", "dovoljan", "dobar", "vrlodobar", "odlican"};
cout << uspesi[uspeh - 1] << endl;
```

Задатак: Различите цифре

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текстови задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Постоје различити начини да се провери да ли су све цифре броја различите.

Скуп виђених цифара можемо представити низом од 10 елемената типа `bool`, тако да елемент на позицији 0 означава да ли се цифра 0 јавља у скупу, на позицији 1 означава да ли се цифра 1 јавља у скупу и тако даље. На почетку је сваком члану низа потребно доделити вредност `false` (на почетку је скуп виђених цифара празан). У језику C++ се то може остварити помоћу `bool skupCifara[10] = {false};`.

```
bool razliciteCifre(int n) {
    bool skupCifara[10] = {false};
    do {
        int cifra = n % 10;
        if (skupCifara[cifra])
            return false;
        skupCifara[cifra] = true;
        n /= 10;
    } while (n > 0);
    return true;
}
```

Задатак: Да ли се даме нападају

Дата је шаховска табла на којој је распоређено осам дама. Напиши програм који проверава да ли се неке две даме нападају (две даме се нападају ако се налазе у истој врсти, истој колони или на истој дијагонали).

Улаз: Са стандардног улаза учитава се 0 – 1 матрица димензије 8×8 чијих 8 јединица описује положај 8 дама.

Излаз: На стандардном излазу исписати текст `NE` ако се даме не нападају или `DA` ако се неке две даме нападају.

Пример 1

Улаз

```
0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0
```

Израз

NE

Пример 2

Улаз

```
0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0
```

Израз

DA

Решење

Током учитавања позиција дама желимо да за сваку врсту, сваку колону, сваку главну и сваку споредну дијагоналу знамо да ли већ садржи неку даму. То можемо остварити тако што направимо пресликавања која су представљена низовима логичких вредности. Постоји 8 врста, 8 колона, 15 главних и 15 споредних дијагонала, па ћемо димензије низова одредити на основу тога. Дама на позицији (v, k) припада врсти број v , колони k , главној дијагонали број $v + k$, и споредној дијагонали број $v - k + 7$ (број 7 додајемо да бисмо избегли негативне редне бројеве дијагонала).

```
bool vrste[8] = {false};
bool kolone[8] = {false};
bool glavne_dijagonale[15] = {false};
bool sporedne_dijagonale[15] = {false};
bool napadajuSe = false;
for (int v = 0; v < 8 && !napadajuSe; v++){
    for (int k = 0; k < 8 && !napadajuSe; k++){
        int element;
        cin >> element;
        if (element == 1) {
            if (vrste[v]) napadajuSe = true;
            else vrste[v] = true;
            if (kolone[k]) napadajuSe = true;
            else kolone[k] = true;
            if (glavne_dijagonale[v+k]) napadajuSe = true;
            else glavne_dijagonale[v+k] = true;
            if (sporedne_dijagonale[v - k + 7]) napadajuSe = true;
            else sporedne_dijagonale[v - k + 7] = true;
        }
    }
}
cout << (napadajuSe ? "DA" : "NE") << endl;
```

Види групачија решења овој задатку.

Задатак: Фреквенција знака

Написати програм који чита једну реч текста коју чине само велика слова енглесне абецедне и исписује слово који се најчешће појављује и колико пута се појављује. Ако се више слова најчешће појављује, исписује се слово које се пре појавило у речи.

Улаз: У једној линији стандардног улаза налази се једна реч текста са не више од 20 слова.

Израз: У првој линији стандардног излаза приказати слово које се најчешће појављује, а у другој линији стандардног улаза исписати и колико пута се појављује.

Пример 1

Улаз

```
РОРОКАТЕРЕТЛ
```

Израз

```
Р
3
```

Пример 2

Улаз

```
ВАСАСВ
```

Израз

```
В
2
```

Решење

Кључни део задатака је да се за свако велико слово енглеског алфабета изброји колико се пута појављује у датој речи. Потребно је, дакле, увести 26 различитих бројача - по један за свако велико слово енглеског алфабета. Јасно је да увођење 26 различитих променљивих не долази у обзир. Потребна је структура података која пресликава дати карактер у број његових појављивања.

Најбоља таква структура у овом задатку је низ бројача у којем се на позицији нула чува број појављивања слова А, на позицији један слова В итд., све до позиције 25 на којој се налази број појављивања слова Z. Централно питање је како на основу карактера одредити позицију његовог бројача у низу. За то се користи чињеница да су карактерима придружени нумерички кодови (ASCII у језику C++) и то на основу редоследа карактера у енглеском алфабету. Редни број карактера је зато могуће одредити одузимањем кода карактера А од кода тог карактера (на пример, код карактера D је 68, док је код карактера А 65, одузимањем се добија 3, што значи да се бројач карактера D налази на месту број 3 у низу). Сличну технику баратања кодовима карактера средли смо, на пример, и у задатку [Абеледно огледало](#).

Задатак можемо решавати тако што ћемо два пута проћи кроз низ унетих слова тј. реч. У првом пролазу бројач појављивања сваког великог слова на које наиђемо увећавамо за 1. Након тога (или паралелно са тим) одређујемо и максималан број појављивања неког слова (за то користимо уобичајене начине за одређивање максимума низа, као, на пример, у задатку [Минимално одступање од просека](#)). У другом пролазу кроз низ унетих слова тј. реч тражимо прво слово речи чији је број појава једнак већ нађеном највећем броју појава неког слова. Када га нађемо исписујемо га и прекидамо петљу (на пример, наредбом `break`). Приметимо да овде заправо вршимо једноставну линеарну претрагу (попут оне у задатку [Негативан број](#)).

```
string rec;
cin >> rec;

int brojPojavljivanja[26] = {0};
for (char c : rec)
    brojPojavljivanja[c - 'A']++;

int maxPojavljivanja = 0;
for (int i = 0; i < 26; i++)
    if (brojPojavljivanja[i] > maxPojavljivanja)
        maxPojavljivanja = brojPojavljivanja[i];

for (char c : rec)
    if (brojPojavljivanja[c - 'A'] == maxPojavljivanja) {
        cout << c << endl
            << maxPojavljivanja << endl;
        break;
    }
```

Види групаџија решења овој задатку.

Задатак: Изоморфне ниске

Две ниске су изоморфне ако се друга може добити неким 1-1 пресликавањем слова прве - сваком слову абеледе одговара неко слово (његова слика у том пресликавању), свако појављивање тог слова у првој ниски се замењује том сликом, при чему не постоје два слова која имају исту слику. На пример, речи `filip` и `cilin` су изоморфне (f се слика у c, i у i, l у l и p у m), док речи `filip` и `madam` нису изоморфне (слово m би требало да буде слика и слова f и слова p, што није допуштено). Напиши програм који испитује да ли су две унете речи изоморфне.

Улаз: Са стандардног улаза се читавају две речи састављене од малих слова енглеске абеледе, свака у посебном реду.

Излаз: На стандардни излаз исписати да ако су речи изоморфне, тј. не у супротном.

Пример 1		Пример 2	
Улаз	Излаз	Улаз	Излаз
filip	da	filip	ne
leden		melem	

Решење

На почетку је потребно проверити да ли су ниске исте дужине (ако нису, одмах констатујемо да нису изоморфне). Затим пролазимо кроз све карактере прве ниске, одржавајући при том тренутно (парцијално) пресликавање. За свако слово прве ниске проверавамо да ли му је слика дефинисана у тренутном пресликавању - ако јесте, проверавамо да ли је једнака одговарајућем слову друге ниске - ако није, ниске нису изоморфне. Ако слика слова није дефинисана, онда би морала да буде дефинисана на одговарајуће слово друге ниске. То, међутим, није могуће ако је то слово већ раније постављено као слика неког слова (ако се то деси, ниске нису изоморфне).

Прво питање је како представити пресликавање, како проверити да ли је неко слово у њему дефинисано, која му је придружена слика и да ли је неко слово већ дефинисано као слика у том пресликавању. Пошто домен тог пресликавања чине мала слова енглеске абетеце (њих 26), могуће је користити и обичан низ карактера, где ћемо на месту 0 чувати слику карактера *a*, на месту 2 слику карактера *b* итд. Позицију на којој се чува слика карактера *chg* лако одређујемо као разлику кода тог карактера и карактера *a* (слично као у задатку Фреквенција знака).

Друго питање је како испитати да ли је неки карактер већ придружен као слика. Наивни начин да се то уради је да се прође кроз цело пресликавање. Међутим, боље решење је да се уместо тога, паралелно са пресликавањем одржава и скуп карактера које су додељене као слике у том пресликавању и да се проверава да ли тражени карактер припада том скупу. Пошто је универзални скуп (скуп свих могућих слика) мали, могуће је употребити обичан низ у којем се карактери пресликавају у истинитосне вредности (*true* ако карактер припада скупу слика тренутног пресликавања тј. *false* ако не припада) - индекси у том низу се поново одређују једноставним рачунањем разлике кода траженог карактера и кода карактера *a*.

```
bool izomorfne(const string& a, const string& b) {
    if (a.size() != b.size())
        return false;
    char sifra[26] = {0};
    bool upotrebljen[26] = {false};
    for (int i = 0; i < a.size(); i++) {
        if (sifra[a[i] - 'a'] != 0) {
            if (b[i] != sifra[a[i] - 'a'])
                return false;
        } else {
            if (upotrebljen[b[i] - 'a'])
                return false;
            else {
                sifra[a[i] - 'a'] = b[i];
                upotrebljen[b[i] - 'a'] = true;
            }
        }
    }
    return true;
}
```

Види групаџија решења овог задатка.

Задатак: Број дана у месецу

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види текст задатка.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Задатак: Хистограм

Група биолога аматера је посматрала популацију жаба. За сваку жабу су забележили величину. Биологе је занимала расподела величина жаба (колико је било малих жаба, колико мало већих, колико највећих и слично). Зато су интервал величина $[a, b]$ поделили на n делова једнаке дужине и избројали колико жаба је припадало сваком од тих делова (ако је величина жабе тачно на граници између два дела, придружује се десном). Напиши програм који им у томе помаже и резултат приказује графички, у облику хистограма.

Улаз: Са стандардног улаза се учитавају бројеви a и b раздвојени једним размаком, а затим у следећем реду и број n . Након тога уноси се број жаба k , а затим и k бројева из интервала $[a, b]$ које представљају величине

жаба.

Излаз: На стандардни излаз се приказују се подеоци (са границама заокруженим на три децимале), након чега се приказује број жаба чија је величина у том подеоку, а затим и процентуални удео жаба у том подеоку у односу на укупан број жаба (тако што се тај проценат заокружи на најближи цео број и затим испише толико карактера *). Звездице су од броја жаба одвојене једним табулатором (знаком Tab).

Пример 1

<i>Улаз</i>	<i>Излаз</i>
0.0 1.0	[0.000, 0.250): 4 *****
4	[0.250, 0.500): 2 *****
10	[0.500, 0.750): 2 *****
0.05	[0.750, 1.000): 2 *****
0.25	
0.95	
0.93	
0.56	
0.73	
0.10	
0.47	
0.23	
0.15	

Задатак: Двојке и тројке дељиве са 3

У датом низу природних бројева дужине n , одреди број група од два или три елемента таквих да је сума свих елемената групе дељива са 3. На пример у низу [2, 1, 3, 10] имамо 4 такве групе: [2, 1], [2, 10], [2, 1, 3], [2, 3, 10].

Улаз: У првој линији стандардног улаза налази се број елемената низа n ($1 \leq n \leq 1000$). У следећих n линија налази се редом елементи низа (природни бројеви између 0 и 1000).

Излаз: На стандардном излазу у једној линији приказати број тражених група у низу.

Пример

<i>Улаз</i>	<i>Излаз</i>
4	4
2	
1	
3	
10	

Задатак: Најчешће мало и велико слово

Уноси се ред по ред текста све до краја улаза. Текст садржи мала и велика слова енглеске абецеде, интерпункцијске знаке и белине. Напиши програм који одређује и испишује најчешће мало и најчешће велико слово у тексту у засебним редовима. Ако се нека слова појављују исти број пута, исписати их сва (у абецедном поретку) али мала слова у једном реду а велика слова у другом реду.

Улаз: Са стандардног улаза учитава се текст који садржи бар једно слово, све док се не дође до краја улаза (он се може унети са Ctrl+Z тј. Ctrl+D).

Излаз: Ако текст садржи и мала и велика слова, излаз се састоји од две линије. У првој линији стандардног излаза приказати мала слова која се најчешће појављују, а у другој линији велика слова која се најчешће појављују.

Ако текст садржи само мала или само велика слова, излаз се састоји од једне линије, а линија која се односи на другу врсту слова се изоставља.

Пример 1

Улаз Излаз
asd ds
sd, asd! C
ABC...CB
CA

Пример 2

Улаз Излаз
asd ads

5.3.2 Асоцијативни низови (мапе, речници)

Програмски језик C++ пружа подршку за креирање мапа (речника, асоцијативних низова) који представљају колекције података у којима се кључевима неког типа придружују вредности неког типа (не обавезно истог). На пример, именима месеци (подацима типа `string`) можемо доделити број дана (податке типа `int`). Речници се представљају објектима типа `map<TipKljuc, TipVrednosti>`, дефинисаном у заглављу `<map>`. На пример,

```
map<string, int> brojDana =
{
    {"januar", 31},
    {"februar", 28},
    {"mart", 31},
    ...
};
```

Приметимо да смо иницијализацију мапе извршили тако што смо навели листу парова облика `{kljuc, vrednost}`. Иницијализацију није неопходно извршити одмах током креирања, већ је вредности могуће додати (а и читати) коришћењем индексног приступа (помоћу заграда `[]`).

```
map<string, int> brojDana;
brojDana["januar"] = 31;
brojDana["februar"] = 28;
brojDana["mart"] = 31;
...
```

Мапу, дакле, можемо схватити и као низ тј. вектор у коме индекси нису обавезно из неког целобројног интервала облика `[0, n)`, већ могу бити произвољног типа.

Претрагу кључа можемо остварити методом `find` која враћа итератор на пронађени елемент тј. итератор из краја мапе (који добијамо методом или функцијом `end`), ако елемент не постоји. На пример,

```
string mesec; cin >> mesec;
auto it = brojDana.find(mesec);
if (it != end(brojDana))
    cout << "Broj dana: " + *it << endl;
else
    cout << "Mesec nije korektno unet" << endl;
```

Све елементе речника могуће је исписати коришћењем петље `for`. На пример,

```
for (auto& it : brojDana)
    cout << it.first << ": " << it.second << endl;
```

Алтернативно, можемо експлицитно користити итераторе

```
for (auto it = brojDana.begin(); it != brojDana.end(); it++)
    cout << it->first << ": " << it->second << endl;
```

Итерација се врши у сортираном редоследу кључева.

Постоји и облик неуређене мапе (`unordered_map` из истоименог заглавља), која може бити у неким ситуацијама мало бржа него уређена (сортирана) мапа, но то је обично занемариво. Кључеви сортиране мапе могу бити само они типови који се могу поредити релацијским операторима, док кључеви неуређене мапе могу бити само они типови који се могу лако претворити у број (тзв. хеш-вредност). Ниске, које ћемо најчешће користити као кључеве, задовољавају оба услова.

Задатак: Редни број месеца

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Потребно је извршити класификацију на основу једне од 12 могућих вредности. Међутим, пошто је потребно унети назив месеца и класификацију извршити на основу тог текста, неопходно је употребити ниску карактера.

Можемо поново реализовати асоцијативно пресликавање којим се кључ (назив месеца) пресликава у вредност (редни број). Међутим, то више не може бити обичан низ (јер кључеви нису мали природни бројеви), већ такозвани асоцијативни низ. У језику С++ можемо употребити мапу тј. колекцију типа `map<string, int>` која називима месеца (нискама карактера) додељује редне бројеве.

```
map<string, int> meseci;
meseci["januar"] = 1;
meseci["februar"] = 2;
meseci["mart"] = 3;
meseci["april"] = 4;
// ...

string mesec;
cin >> mesec;
cout << meseci[mesec] << endl;
```

Задатак: Римски у арапски

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Вредности цифара можемо чувати у асоцијативном низу (мапи, речнику). У језику С++ можемо употребити библиотечку структуру података `map`, декларисану у заглављу `<map>`. У овом задатку је потребно пресликавање карактера у њихове вредности па користимо `map<char, int>`.

```
int rimskiUArapski(string rimski) {
    // vrednost svake rimske cifre
    map<char, int> cifre = { {'M', 1000}, {'D', 500}, {'C', 100},
                          {'L', 50}, {'X', 10}, {'V', 5}, {'I', 1}};
    // rezultat - vrednost rimski zapisanog broja
    int arapski = 0;
    // prolazimo kroz sve cifre rimskog broja
    for (int i = 0; i < rimski.size(); i++) {
        // ako neposredno iza tekuce postoji cifra veca od nje
        if (i + 1 < rimski.size() && cifre[rimski[i]] < cifre[rimski[i+1]])
            // rezultat umanjujemo za vrednost tekuce cifre
            arapski -= cifre[rimski[i]];
        else
            // inace rezultat uvecavamo za vrednost tekuce cifre
            arapski += cifre[rimski[i]];
    }
    // vracamo konacan rezultat
    return arapski;
}
```

Задатак: Изоморфне ниске

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Директан начин да се представи пресликавање је да се употребе библиотечке колекције, чиме се добија мало читљивији текст програма. У језику С++ се може употребити `map` из заглавља `<map>` или `unordered_map` из заглавља `<unordered_map>`. И скуп може бити представљен библиотечким колекцијама. У језику С++ се може употребити `set` из заглавља `<set>` или `unordered_set` из заглавља `<unordered_set>`.

```
bool izomorfne(const string& a, const string& b) {
    if (a.size() != b.size())
        return false;
    map<char, char> sifra;
    set<char> upotrebljeni;
    for (int i = 0; i < a.size(); i++) {
        auto it = sifra.find(a[i]);
        if (it != sifra.end()) {
            if (b[i] != it->second)
                return false;
        } else {
            if (upotrebljeni.find(b[i]) != upotrebljeni.end())
                return false;
            else {
                sifra[a[i]] = b[i];
                upotrebljeni.insert(b[i]);
            }
        }
    }
    return true;
}
```

Задатак: Фреквенција знака

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види тексты задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Пресликавање карактера у њихов број појављивања могуће је остварити и библиотечким структурама података које представљају тзв. асоцијативне низове (мапе, речнике). У језику С++ бисмо могли употребити `map<char, int>` или `unordered_map<char, int>`. Ипак, ове структуре података су примереније за ситуације у којима није тако једноставно на основу кључа добити њихову нумеричку вредност и када је скуп допустивих кључева шири.

```
// rec koja se analizira
string rec;
cin >> rec;

// broj pojavljivanja svakog slova od A do Z
map<char, int> brojPojavljivanja;
// uvecavamo broj pojavljivanja svakog slova iz reci
for (char c : rec)
    brojPojavljivanja[c]++;

// odredjujemo najveći broj pojavljivanja slova
int maxPojavljivanja = 0;
for (auto it = brojPojavljivanja.begin(); it != brojPojavljivanja.end(); it++)
    if (it->second > maxPojavljivanja)
        maxPojavljivanja = it->second;

// ispisujemo slovo koje se prvo pojavljuje u reci sa tim brojem
```

```
// pojavljivanja
for (char c : rec)
    if (brojPojavljanja[c] == maxPojavljanja) {
        cout << c << endl
            << maxPojavljanja << endl;
        break;
    }
}
```

Задатак: Фреквенције речи

Милица је куповала разне производе и сваки пут када се вратила из куповине дописивала је све производе које је купила на списак. Напиши програм који помаже Миlici да одреди производ који је најчешће куповала током претходне године.

Улаз: Са стандардног улаза учитавају се називи производа, све док се не дође до краја улаза. Сваки назив је реч која садржи највише 10 малих слова енглеског алфабета.

Излаз: На стандардни излаз исписати један ред који садржи назив најчешће купованог производа, један размак и број пута колико је тај производ купљен. Ако је више производа купљено исти број пута, исписати онај који је први по абецедном редоследу.

Пример

Улаз	Излаз
jabuka	hleb 3
hleb	
kruska	
jabuka	
sljiva	
hleb	
mleko	
jabuka	
hleb	

Решење**Бројање речи**

Овај задатак је донекле сличан задатку **Фреквенција знака**, једино што се уместо појединачних карактера броје речи. Поново је централно место у задатку избројати појављивања сваке речи која се јавила на улазу. За разлику од ситуације када смо на основу ASCII кода карактера могли одредити његову позицију и тако број појављивања сваког карактера чувати у низу бројача, овај пут то није једноставно и у ефикасном решењу је потребно користити напредније структуре податка.

У језику C++ пресликавање речи у њен број појављивања можемо реализовати или типом `map<string, int>` (који је заснован на претраживачком стаблу) или `unordered_map<string, int>` (који је заснован на хеш-таблици).

Учитавамо реч по реч док не дођемо до краја улаза (слично као у задатку **Читање до краја улаза**). За сваку реч увећавамо њен број појављивања у мапи тј. у речнику.

Изразом `brojPojavljanja[rec]` се приступа броју појављивања речи. У језику C++ се изразом `brojPojavljanja[rec]++` може увећати број појављивања било да реч постоји у мапи или не (ако не постоји, овим се број појављивања поставља на 1).

Једном када је познат број појављивања сваке речи, тада је могуће одредити тражену реч са најмањим бројем појављивања. Пошто се у случају више речи са истим бројем појављивања тражи она која је лексикографски најмања, користимо лексикографско поређење (хијерархијско поређење на основу више критеријума), као, на пример, у задатку **Победник у три дисциплине**.

```
string s;
map<string, int> brojPojavljanja;
while (cin >> s)
    brojPojavljanja[s]++;
```

5.3. ПРЕСЛИКАВАЊА

```
int max = 0; string maxRec;
for (auto it = brojPojavljivanja.begin();
     it != brojPojavljivanja.end(); it++)
    if (it->second > max ||
        (it->second == max && it->first < maxRec)) {
        maxRec = it->first;
        max = it->second;
    }

cout << maxRec << " " << max << endl;
```

Задатак: Миноморија

Ученици играју игру Миноморија. Игра се састоји у томе да играчи могу да током 10 секунди погледају низ од N различитих бројева. Након тога они морају да одговоре на серију од Q питања о томе на ком месту у низу се налази одређени број (места се броје од 1 до N). Ако броја из питања нема у улазном низу, играч треба да одговори 0.

Улаз: Са стандардног улаза се учитава један цео број N ($0 < N \leq 10^5$), који представља број чланова низа. У другом реду улаза налази се N чланова низа A_i ($0 \leq A_i < 10^7$), раздвојених по једним размаком. У трећем реду улаза налази се природан број Q ($0 < Q \leq 10^5$) који представља број упита. Од четвртог реда па надаље у сваком реду се налази по један цео број Q_i ($0 \leq Q_i \leq 10^7$). За сваки број из упита потребно је у посебном реду стандардног излаза исписати на ком месту у улазном низу се он налази или 0 ако број из упита не постоји у задатом низу.

Излаз: За сваки упит треба исписати у посебном реду одговор који се у тексту задатка тражи (позиција броја у низу или 0 ако траженог броја нема у задатом низу).

Пример 1

Улаз	Излаз
10	2
100 233 5 9 11 698 1234 25 113 654	0
6	10
233	0
234	5
654	1
1000000	
11	
100	

Задатак: Телефонски именик

Телефонски именик ради у телефону аутоматски кориснику нуди бројеве телефона који садрже тренутно унете цифре као подниз. Претпоставићемо да се у именику сваки број појављује највише једном и да су сви бројеви телефона деветоцифрени. Претпоставимо и да се, на пример, у именику тренутно налазе бројеви 123456789, 102030000 и 987123456.

- Када се унесе 123, кориснику се нуде бројеви 123456789 и 987123456.
- Када се унесе 30, кориснику се нуди само број 102030000.

Напиши програм који за сваки број у именику одређује најкраћи низ цифара које је могуће откуцати да би се приказао само тај број (ако има више таквих низова, приказати онај који се први јавља у броју, гледајући слева надесно).

Улаз: Са стандардног улаза се учитава број телефонских бројева у именику n ($1 \leq n \leq 10^5$), а затим n деветоцифрених бројева из n наредних редова.

Излаз: На стандардни излаз исписати n бројева који за сваки унети број редом дају минимални број цифара.

Пример 1		Пример 2	
Улаз	Изназ	Улаз	Изназ
3	67	4	2
123456789	0	123456789	193
102030000	98	193456789	13
987123456		134567819	91
		934567891	

5.4 Вишедимензиони низови и матрице

5.4.1 Елементи програмског језика

5.4.1.1 Матрице

Матрица може да се посматра као низ низова: имамо низ врста (редова), а свака врста је низ елемената. Позиција сваког елемента у матрици је дефинисана са два броја, тј. две димензије – у којој *врсти* тј. *реду*, и у којој *колони* се елемент налази.

Статичка матрица се у језику C++ алоцира слично као што се алоцира статички низ. Димензија статички алоциране матрице мора бити позната већ у фази превођења и зато је потребно унапред знати максимални број врста и колоне те матрице. Тако, ако рецимо хоћемо да дефинишемо матрицу, која има 50 врста и 100 колоне, а елементи су целобројне променљиве, то бисмо урадили на следећи начин:

```
int brojevi[50][100];
```

Овим се декларише матрица од 5000 елемената под називом `brojevi`. Врсте ове матрице су нумерисане од 0 до 49, а колоне су нумерисане од 0 до 99.

Матрице је могуће и динамички алоцирати, слично као и низове, али се као напреднија тема неће обрађивати у овом материјалу.

Једном елементу матрице приступамо тако што у угластој загради, након назива матрице наведемо број врсте, а након тога у другој угластој загради наведемо и број колоне у којој се тај елемент налази, нпр. `brojevi[1][4]` означава елемент који се налази у другој врсти и петој колони матрице. Када приступимо елементу матрице, можемо са њим радити све што можемо и са обичном променљивом: користити га у рачуници, постављати му вредност, итд.

На пример, дигитална, црно-бела слика може да се представи помоћу целобројне матрице тако што ће број 1 да означава црно, а број 0 бело поље, на следећи начин:

```
int slika[10][15] = {
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0 },
    { 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
    { 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0 },
    { 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0 },
    { 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0 },
    { 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
};
```

У овом случају смо меморију за матрицу резервисали истовремено са попуњавањем вредности. Уколико при алокацији матрице вршимо и њену иницијализацију, можемо изоставити прву димензију матрице, односно писати `int slika[][15] = ...`

5.4.1.2 Разуђени низови

Свака врста матрице мора да има исти број елемената. Међутим, могуће је правити дводимензионе низове код којих то не мора бити случај. Такви низови се називају *разуђени низови* (енгл. *jagged array*) који су низови низова у правом смислу те речи.

5.4. ВИШЕДИМЕНЗИОНИ НИЗОВИ И МАТРИЦЕ

Најједноставније их је креирати помоћу библиотечке колекције `<vector>`, као вектор вектора. На пример, уколико желимо да алоцирамо разуђени низ целобројног типа, то можемо да урадимо на следећи начин:

```
vector<vector<int>> razudjenNiz(3);
razudjenNiz[0].resize(5);
razudjenNiz[1].resize(4);
razudjenNiz[2].resize(2);
```

Овим је направљен разуђен низ који има три врсте, у првој се чува пет елемената, у другој четири и у трећој два.

Елементу се приступа помоћу двоструких угластих заграда. На пример, `razudjenNiz[1][3]` означава четврти елемент у другој врсти.

5.4.2 Матрице

Задатак: Број бомби у околини

У игрици Minesweeper на пољима се налазе сакривене бомбе и задатак играча је да их пронађе. Играчу се приказује табла са бројевима где сваки број представља број бомби које се налазе у околини датог поља (гледају се околна поља у свих 8 смерова). Твој задатак је да започнеш програмирање ове игре тако што ћеш написати програм који за дати распоред бомби одређује ове бројеве.

Улаз: Са стандардног улаза се читавају два броја m и n који представљају димензије табле ($3 \leq m, n \leq 100$) и након тога матрица која има m врста и n колона која садржи нуле и јединице (где јединица означава бомбу).

Издаз: Матрица димензије $m \times n$ која одређује број бомби у околини сваког поља полазне матрице.

Пример

Улаз	Издаз
3 4	2 2 3 1
0 1 0 1	2 4 3 2
1 0 1 0	2 2 2 1
0 1 0 0	

Решење

Након алокације и читавања почетне матрице бомби (матрице логичких вредности) за свако поље се пребројавају бомбе у његовој околини. Резултати ће бити представљени посебном матрицом целих бројева, па онда исписани, мада би било могуће и исписивати их одмах након израчунавања, без памћења у посебној матрици.

Пошто обрађујемо свако поље полазне матрице, обилазак ћемо организовати на уобичајени начин, кроз две угнежфене петље. Претпоставимо да спољна, коришћењем индекса v пролази кроз све врсте, а унутрашња, коришћењем индекса k кроз све колоне текуће врсте. За свако поље (одређено индексима v и k) потребно је обићи његових 8 суседних поља (или мање, ако је поље на рубу) и пребројати бомбе на њима. За поље (v, k) суседна су поља $(v-1, k-1)$, $(v-1, k)$, $(v-1, k+1)$, $(v, k-1)$, $(v, k+1)$, $(v+1, k-1)$, $(v+1, k)$, $(v+1, k+1)$. Дакле, сва поља која се од (v, k) добијају додавањем вредности -1 , 0 или 1 на било коју од координата, осим самог поља (v, k) . Најлакши начин да се ова поља наброје је да се опет употребе угнежфене петље. На пример, у спољној петљи се мења увећање врсте dv од -1 до 1 , а у унутрашњој увећање колоне dk од -1 до 1 . Тако се обилазе поља са координатама $v' = v + dv$ и $k' = k + dk$ и за свако од њих се проверава да ли је (v', k') различито од (v, k) , да ли се налази у оквиру матрице (да ли је $0 \leq v' < m$ и $0 \leq k' < n$) и да ли се на њему налази бомба. Ако су сви ти услови испуњени, бројач бомби у околини се увећава за 1 .

```
// maksimalne dimenzije matrice - koristi se za staticku alokaciju
const int MAX_M = 100;
const int MAX_N = 100;

int main() {
    // dimenzije matrice
    int m, n;
    cin >> m >> n;

    // matrica bombi
    bool bomba[MAX_M][MAX_N];
```

```

// ucitavamo matricu bombi
for (int v = 0; v < m; v++)
    for (int k = 0; k < n; k++)
        cin >> bomba[v][k];

// broj_bombi[v][k] - broj bombi u okolini polja (v, k)
int broj_bombi[MAX_M][MAX_N];
for (int v = 0; v < m; v++)
    for (int k = 0; k < n; k++) {
        broj_bombi[v][k] = 0;
        for (int dv = -1; dv <= 1; dv++)
            for (int dk = -1; dk <= 1; dk++) {
                int vv = v + dv, kk = k + dk;
                if ((vv != v || kk != k) &&
                    (0 <= vv && vv < m) &&
                    (0 <= kk && kk < n) &&
                    bomba[vv][kk])
                    broj_bombi[v][k]++;
            }
    }

// ispisujemo rezultat
for (int v = 0; v < m; v++) {
    for (int k = 0; k < n; k++)
        cout << broj_bombi[v][k] << " ";
    cout << endl;
}
return 0;
}

```

Један начин да се избегне провера да ли се елемент у околини налази у оквиру матрице је да се матрици дода оквир који се састоји од две додатне врсте (једна горе и једна доле) и две додатне колоне (једна лево и једна десно) попуњене вредношћу `false`.

Задатак: Број правоугаоника на слици

На папиру подељеном на $n \times n$ јединичних квадратића, нацртано је неколико правоугаоника (паралелно ивицама папира) који се не додирују, осим евентуално једним теменом. Написати програм који одређује број правоугаоника на папиру.

Улаз: У првом реду стандардног улаза дата је димензија табле $n \leq 10$, а затим је у наредним редовима задата матрица која садржи јединице и нуле, тако да су јединице уписане на местима на којима су нацртани правоугаоници.

Излаз: На стандардни излаз исписати тражени број правоугаоника.

Пример

Улаз	Излаз
7	4
1 1 1 0 1 1 1	
1 1 1 0 1 1 1	
0 0 0 0 0 0 0	
0 0 0 0 0 0 0	
1 1 1 0 1 1 1	
1 1 1 0 1 1 1	
1 1 1 0 1 1 1	

Решење

Правоугаоника има тачно онолико колико има њихових горњих левих темена, па је у решењу довољно избројати само горња лева темена правоугаоника. Пошто се правоугаоници додирују највише у једном темену, елемент матрице је горње лево теме правоугаоника ако и само ако се изнад њега и лево од њега не налази

5.4. ВИШЕДИМЕНЗИОНИ НИЗОВИ И МАТРИЦЕ

елемент 1. То је могуће ако се теме које испитујемо налази у првом реду или у првој колони или ако елемент изнад и лево од њега нема вредност 1. Дакле, у имплементацији се мора водити рачуна да ли је индекс врсте $v = 0$ или индекс колоне $k = 0$, да не би дошло до прекорачења опсега у матрици. Нагласимо да се у приликом израчунавања вредности израза $v == 0 \ || \ mat[v-1][k] != 1$, у случају када је вредност променљиве v нула израз $mat[v-1][k]$ не израчунава (због лењог израчунавања оператора $||$).

```
// matrica pravougaonika, sadrzi 0 i 1
int A[MAX][MAX];
// dimenzija matrice
int n;
// ucitavamo matricu
ucitaj(A, n);

// brojac pravougaonika
int brojPravougaonika = 0;
// analiziramo sve elemente matrice
for (int v = 0; v < n; v++)
    for (int k = 0; k < n; k++)
        // ako je A[v][k] gornje levo teme pravougaonika
        if (A[v][k] == 1 &&
            (v == 0 || A[v-1][k] != 1) &&
            (k == 0 || A[v][k-1] != 1))
            // uvecavamo brojPravougaonika
            brojPravougaonika++;

// ispisujemo rezultat
cout << brojPravougaonika << endl;
```

Провера да ли је елемент матрице горње лево теме неког правоугаоника може да се издвоји у посебну функцију.

```
// provera da li je A[v,k] gornje levo teme nekog pravougaonika
bool jeGornjeLevoTemePravougaonika(int A[MAX][MAX], int v, int k) {
    // na tom polju mora da pise 1
    if (A[v][k] != 1)
        return false;
    // iznad njega ne sme da pise 1
    if (v != 0 && A[v-1][k] == 1)
        return false;
    // levo od njega ne sme da pise 1
    if (k != 0 && A[v][k-1] == 1)
        return false;
    return true;
}

int main()
{
    // matrica pravougaonika, sadrzi 0 i 1
    int A[MAX][MAX];
    // dimenzija matrice
    int n;
    // ucitavamo matricu
    ucitaj(A, n);

    // brojac pravougaonika
    int brojPravougaonika = 0;
    // analiziramo sve elemente matrice
    for (int v = 0; v < n; v++)
        for (int k = 0; k < n; k++)
            // ako je A[v][k] gornje levo teme pravougaonika
```



```

if (jeGornjeLevoTemePravougaonika(A, v, k))
    // uvecavamo brojPravougaonika
    brojPravougaonika++;

    // ispisujemo rezultat
    cout << brojPravougaonika << endl;
    return 0;
}

```

Посебна анализа случајева би се могла избећи ако би се вештачки у матрицу додао оквир који садржи нуле (заправо, довољно је додати само горњи и леви део оквира, односно доњи и десни ако проверавамо да ли је елемент матрице доње десно теме правоугаоника), слично као што смо показали у задатку **Број бомби у околини**.

Задатак: Пресликавања матрице

Деца су спремала приредбу у школском дворишту. Свако је обукао костим различите боје и поређали су се у квадратну матрицу. Током тачке деца су се премештала и то на веома правилне начине. Прво су пресликали матрицу око главне дијагонале. Затим су заротирали матрицу за 90 степени удесно (у смеру казаљке на сату). Након тога су је пресликали хоризонтално и на крају су је пресликали око споредне дијагонале. Напиши програм који исписује распоред боја дечијих костима након сваког корака њихове кореографије.

Улаз: Са стандардног улаза се учитава број n ($3 \leq n \leq 10$), а затим и квадратна матрица димензије n која садржи бројеве између 0 и 10 (сваки број означава једну боју костима).

Излаз: На стандардни излаз исписује се 5 матрица. Полазна, учитана и затим матрица добијена након сваке трансформације.

Пример

Улаз	Излаз
3	1 2 3
1 2 3	4 5 6
4 5 6	7 8 9
7 8 9	
	1 4 7
	2 5 8
	3 6 9
	3 2 1
	6 5 4
	9 8 7
	9 8 7
	6 5 4
	3 2 1
	1 4 7
	2 5 8
	3 6 9

Решење

Размотримо шта се дешава са индексима матрице након пресликавања матрице око главне дијагонале (каже се и транспоновања матрице).

00 01 02 03	00 10 20 30
10 11 12 13	01 11 21 31
20 21 22 23	02 12 22 32
30 31 32 33	03 13 23 33

Елементу на позицији (v, k) у транспонованој матрици (матрици пресликаној око главне дијагонале) одговара елемент на позицији (k, v) оригиналне матрице. Један начин да се изврши транспоновање је да се на основу овога од полазне направи нова матрица. Међутим, транспоновање можемо једноставно извести и у месту, тј.

5.4. ВИШЕДИМЕНЗИОНИ НИЗОВИ И МАТРИЦЕ

без изградње нове матрице, тако што разменимо све елементе који се налазе у горњем троуглу изнад главне дијагонале (тј. све елементе на позицијама (v, k) за које важи да је $0 \leq v < n, k < v$), са њима одговарајућим елементима (k, v) .

Размотримо шта се дешава са индексима матрице након ротације удесно.

```
00 01 02 03    30 20 10 00
10 11 12 13    31 21 11 01
20 21 22 23    32 22 12 02
30 31 32 33    33 23 13 03
```

Елементу на позицији (v, k) у ротираној матрици одговара елемент на позицији $(n - k - 1, v)$ у оригиналној матрици. На основу овога, можемо креирати нову матрицу тако што ћемо проћи кроз сваки њен елемент (v, k) (у угнежђеним петљама) и преписати одговарајући елемент из полазне матрице. Након тога, полазну матрицу можемо заменити новодобијеном (тако што препишемо све елементе).

Размотримо шта се дешава са индексима матрице након хоризонталног пресликавања.

```
00 01 02 03    30 31 32 33
10 11 12 13    20 21 22 23
20 21 22 23    10 11 12 13
30 31 32 33    00 01 02 03
```

Елемент на позицији (v, k) у новој матрици је елемент на позицији $(n - v - 1, k)$ у полазној матрици. На ову трансформацију можемо гледати и као на обртање редоследа елемената сваке колоне појединачно и можемо је реализовати техником обртања елемената низа помоћу два бројача (као у задатку [Обртање низа](#)).

На крају, размотримо и пресликавање око споредне дијагонале.

```
00 01 02 03    33 23 13 03
10 11 12 13    32 22 12 02
20 21 22 23    31 21 11 01
30 31 32 33    30 20 10 00
```

Елемент на позицији (v, k) у пресликаној матрици је елемент на позицији $(n - k - 1, n - v - 1)$ у оригиналној матрици. Пресликавање можемо извршити у месту, тако што ћемо сваки елемент горњег троугла изнад споредне дијагонале матрице (елемент на позицији (v, k) таквој да је $0 \leq v < n$ и $0 \leq k < n - v - 1$) разменити са елементом на позицији $(n - k - 1, n - v - 1)$.

Пошто је максимална димензија матрице унапред позната и веома је мала, све матрице које користимо у задатку ћемо статички алоцирати на ту максималну димензију.

Сваку трансформацију реализоваћемо у склопу посебне функције. Свака функција ће мењати матрицу коју прими као аргумент.

```
const int MAX = 100;

// prepisuje elemente matrice B u matricu A
void prepisi(int A[MAX][MAX], int B[MAX][MAX], int n) {
    for (int v = 0; v < n; v++)
        for (int k = 0; k < n; k++)
            A[v][k] = B[v][k];
}

// transponuje matricu A (preslikava elemente oko glavne dijagonale)
void transponuj(int A[MAX][MAX], int n) {
    for (int v = 0; v < n; v++)
        for (int k = 0; k < v; k++)
            swap(A[v][k], A[k][v]);
}

// rotira matricu za 90 stepeni udesno
void rotiraj90(int A[MAX][MAX], int n) {
    // rezultat upisujemo u novu, pomocnu matricu
```

```

int B[MAX][MAX];
for (int v = 0; v < n; v++)
    for (int k = 0; k < n; k++)
        B[v][k] = A[n-k-1][v];
// prepisujemo rezultat u polaznu matricu
prepisi(A, B, n);
}

// preslikava elemente matrice horizontalno (obrce redosled elemenata
// svake kolone)
void preslikajHorizontalno(int A[MAX][MAX], int n) {
    for (int k = 0; k < n; k++)
        for (int vi = 0, vj = n-1; vi < vj; vi++, vj--)
            swap(A[vi][k], A[vj][k]);
}

// preslikava elemente matrice oko sporedne dijagonale
void preslikajOkoSporedneDijagonale(int A[MAX][MAX], int n) {
    for (int v = 0; v < n; v++)
        for (int k = 0; k < n-v-1; k++)
            swap(A[v][k], A[n-k-1][n-v-1]);
}

// ispisuje kvadratnu matricu na standardni izlaz
void ispisi(int A[MAX][MAX], int n) {
    for (int v = 0; v < n; v++) {
        for (int k = 0; k < n; k++)
            cout << A[v][k] << " ";
        cout << endl;
    }
}

// ucitava kvadratnu matricu sa standardnog ulaza - ucitava se
// prvo dimenzija, a zatim i elementi matrice
void ucitaj(int A[MAX][MAX], int& n) {
    cin >> n;
    for (int v = 0; v < n; v++)
        for (int k = 0; k < n; k++)
            cin >> A[v][k];
}

int main() {
    int n;
    int A[MAX][MAX];
    ucitaj(A, n);
    ispisi(A, n); cout << endl;
    transponuj(A, n);
    ispisi(A, n); cout << endl;
    rotiraj90(A, n);
    ispisi(A, n); cout << endl;
    preslikajHorizontalno(A, n);
    ispisi(A, n); cout << endl;
    preslikajOkoSporedneDijagonale(A, n);
    ispisi(A, n);
    return 0;
}

```

Ротацију је могуће реализовати и као трансформацију у месту, тј. трансформацију засновану само на разменама елемената полазне матрице, без коришћења додатног простора. Ротација се може реализовати тако

5.4. ВИШЕДИМЕНЗИОНИ НИЗОВИ И МАТРИЦЕ

што се посебно ротира сваки од “прстена” матрице (код матрице димензије 4 пута 4 постоје два прстена и то спољашњи у ком су редом елементи 00 01 02 03 13 23 33 32 31 30 20 10 и унутрашњи у ком су редом елементи 11 12 22 21.

```
00 01 02 03
10 11 12 13
20 21 22 23
30 31 32 33
```

Ротацију сваког прстена вршимо тако што ротирамо четворке одговарајућих елемената. У првом прстену ротирамо четворку 00, 03, 33 и 30, затим четворку 01, 13, 32 и 20, и на крају четворку 02, 23, 31. У унутрашњем прстену ротирамо једину његову четворку 11, 12, 22 и 21. У општем случају, ротирају се елементи на позицијама (i, j) , $(n - 1 - j, i)$, $(n - 1 - i, n - 1 - j)$ и $(j, n - 1 - i)$ за i из интервала $[0, \lfloor \frac{n}{2} \rfloor]$ (тима је одређен индекс прве врсте у текућем прстену) и j из интервала $[i, n - i - 1)$ (тима је одређен сваки елемент прве врсте прстена који одговара врсти i).

```
// rotira matricu za 90 stepeni udesno
void rotiraj90(int A[MAX][MAX], int n) {
    for (int i = 0; i < n / 2; i++)
        for (int j = i; j < n - i - 1; j++) {
            int tmp = A[i][j];
            A[i][j] = A[n-1-j][i];
            A[n-1-j][i] = A[n-1-i][n-1-j];
            A[n-1-i][n-1-j] = A[j][n-1-i];
            A[j][n-1-i] = tmp;
        }
}
```

Задатак: Да ли се даме нападају

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. *Види текстови задатка.*

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Један начин је да проверимо сваку врсту, сваку колону и сваку дијагоналу, и да избројимо колико се на њима налази дама. Чим установимо да је број дама већи од 1, бројање можемо прекинути и можемо установити да се даме нападају. Даме ћемо чувати у матрици логичких вредности димензије 8×8 . Обилазак свих елемената врста и колона крајње је једноставан. Свака врста одређена је неким бројем v , тако да се провера свих врста врши у петљи у којој се v мења од 0 до 7. На сваку врсту примењује се бројање дама (бројање елемената који задовољавају неки услов одговара задатку **Просек одличних**) - бројач иницијализујемо на нулу, а затим у петљи пролазимо кроз елементе врсте v тако што мењамо индекс k од 0 до 7, проверавамо да ли се на пољу (v, k) налази дама, и ако се налази, увећавамо бројач. При сваком увећавању бројача проверавамо да ли му је вредност већа од 1 и ако јесте прекидамо бројање. Све провере можемо реализовати у засебним функцијама које враћају true ако и само ако су пронађене две даме које се нападају и у том случају је веома једноставно извршити прекид (наредбом return true). Провера колона је аналогна провери врста. Што се тиче провере свих дијагонала, обилазак дијагонала приказан је у задатку **Највећа дијагонала**. На сваку дијагоналу (прво на оне паралелне главној, а затим и на оне паралелне споредној) примењујемо поступак бројања дана и опет, ако број дама на некој дијагонали постане већи од 1, прекидамо поступак. У главном програму проверавамо да ли је нека од провера вратила true.

Рецимо и да се, због лењог израчунавања оператора || којим су четири позива функција повезана, ако нека од функција врати true, оне након ње ни не позивају.

```
int tabla[8][8];

void ucitaj() {
    for (int v = 0; v < 8; v++)
        for (int k = 0; k < 8; k++)
            cin >> tabla[v][k];
}
```

```
bool proveraKolona() {
    for (int k = 0; k < 8; k++) {
        int brojDama = 0;
        for (int v = 0; v < 8; v++)
            if (tabla[v][k]) {
                brojDama++;
                if (brojDama > 1)
                    return true;
            }
    }
    return false;
}

bool proveraVrsta() {
    for (int v = 0; v < 8; v++) {
        int brojDama = 0;
        for (int k = 0; k < 8; k++)
            if (tabla[v][k]) {
                brojDama++;
                if (brojDama > 1)
                    return true;
            }
    }
    return false;
}

bool proveraGlavnihDijagonala() {
    for (int d = -7; d <= 7; d++) {
        int brojDama = 0;
        for (int v = max(0, -d); v < min(8, 8-d); v++)
            if (tabla[v][v+d]) {
                brojDama++;
                if (brojDama > 1)
                    return true;
            }
    }
    return false;
}

bool proveraSporednihDijagonala() {
    for (int d = 0; d <= 14; d++) {
        int brojDama = 0;
        for (int v = max(0, d-7); v < min(8, d+1); v++)
            if (tabla[v][d-v]) {
                brojDama++;
                if (brojDama > 1)
                    return true;
            }
    }
    return false;
}

int main() {
    ucitaj();
    if (proveraVrsta() ||
        proveraKolona() ||
        proveraGlavnihDijagonala() ||
        proveraSporednihDijagonala())
```

```

    cout << "DA" << endl;
else
    cout << "NE" << endl;
return 0;
}

```

Други начин да решимо задатак је да проверимо све парове дама и да за сваке две даме проверимо да ли се налазе на истој врсти, истој колони или на истој дијагонали слично као што смо то радили у задатку **Да ли се две даме нападају**. Положај дама можемо забележити у матрици целих бројева димензије 8×2 тако што ћемо у свакој врсти те матрице чувати број врсте и број колоне у којој се налази нека дама. Два поља (v_1, k_1) и (v_2, k_2) се налазе у истој врсти ако је $v_1 = v_2$, у истој колони ако је $k_1 = k_2$, а на истој дијагонали ако је $|v_1 - v_2| = |k_1 - k_2|$ (тј. ако је хоризонтално растојање та два поља једнако вертикалном).

```

int dame[8][2];

void ucitaj() {
    int d = 0;
    for (int v = 0; v < 8; v++)
        for (int k = 0; k < 8; k++) {
            int p;
            cin >> p;
            if (p == 1) {
                dame[d][0] = v;
                dame[d][1] = k;
                d++;
            }
        }
}

bool proverParovaDama() {
    for (int d1 = 0; d1 < 8; d1++)
        for (int d2 = d1+1; d2 < 8; d2++) {
            if (dame[d1][0] == dame[d2][0] ||
                dame[d1][1] == dame[d2][1] ||
                abs(dame[d1][0] - dame[d2][0]) ==
                abs(dame[d1][1] - dame[d2][1]))
                return false;
        }
    return true;
}

int main() {
    ucitaj();
    if (proveraParovaDama())
        cout << "NE" << endl;
    else
        cout << "DA" << endl;
    return 0;
}

```

Задатак: Највећа дијагонала

Квадратна матрица садржи целе бројеве. Напиши програм који проналази највећи збир бројева на некој од дијагонала матрице (у обзир се узимају главна дијагонала, споредна дијагонала и све дијагонале њима паралелне).

Улаз: Са стандардног улаза се учитава број n ($2 \leq n \leq 100$) и након тога елементи квадратне матрице димензије $n \times n$ (цели бројеви између -10 и 10).

Излаз: На стандардни излаз исписати један цео број - највећи збир елемената на некој од дијагонала.

Пример

Улаз	Израз
3	12
10 -10 2	
1 -9 3	
-2 9 9	

Решење

Посматрајмо индексе елемената у једној квадратној матрици димензије 8.

00	01	02	03	04	05	06	07
10	11	12	13	14	15	16	17
20	21	22	23	24	25	26	27
30	31	32	33	34	35	36	37
40	41	42	43	44	45	46	47
50	51	52	53	54	55	56	57
60	61	62	63	64	65	66	67
70	71	72	73	74	75	76	77

Посматрајмо њене дијагонале паралелне главној. На првој се налази само елемент $(7, 0)$. На другој, елементи $(6, 0)$ и $(7, 1)$, на трећој елементи $(5, 0)$, $(6, 1)$ и $(7, 2)$. Приметимо да је разлика индекса колоне и индекса врсте константна на свакој споредној дијагонали. На првој (која садржи само један елемент) та разлика је $-(n - 1)$, на другој $-(n - 2)$, а на последњој (која опет садржи само један елемент) $n - 1$. Дакле, дијагонале паралелне главној су одређене целим бројевима d из интервала $[-(n - 1), (n - 1)]$. Одредимо индексе врста које имају елементе на дијагонали одређеној бројем d . Прво, за сваки индекс врсте v важи $0 \leq v < n$. Да би елемент врсте v припадао дијагонали одређеној бројем d мора да важи $k - v = d$. Зато је индекс колоне једнак $k = v + d$. Да би елемент припадао матрици мора да важи да је $0 \leq k < n$, тј. да је $0 \leq v + d < n$, тј. да је $-d \leq v < n - d$. Дакле, мора да важи $0 \leq v$ и мора да важи $-d \leq v$. Отуда мора да важи и $\max(0, -d) \leq v$. Слично, из $v < n$ и $v < n - d$ мора да важи и $v < \min(n, n - d)$. Дакле, обраду свих дијагонала вршићемо у петљи по d које узима вредности редом од $-(n - 1)$ до $n - 1$. За свако d пролазићемо кроз елементе на тој дијагонали тако што ћемо у петљи пролазити кроз врсте $\max(0, -d) \leq v < \min(n, n - d)$ и разматрати елементе са координатама $(v, v + d)$. Израчунаваћемо збир тих елемената (применом алгоритма сабирања серије, описаног у задатку **Збир n бројева**). Истовремено ћемо израчунавати и највећу вредност добијеног збира (применом алгоритма одређивања максимума серије бројева, описаног у задатку **Најнижа температура**).

Посматрајмо и дијагонале паралелне споредној. На првој се налази елемент $(0, 0)$, на другој елементи $(0, 1)$ и $(1, 0)$, на трећој елементи $(0, 2)$, $(1, 1)$ и $(2, 0)$ и слично. Може се приметити да на свакој дијагонали индекси врста и колоне имају константан збир и то 0, на првој дијагонали, 1 на другој, па све до $2(n - 1)$ на последњој. Дакле, свака од ових дијагонала одређена је вредношћу d из интервала $[0, 2(n - 1)]$. Одредимо сада индексе врста које имају елементе на дијагонали одређене бројем d . За индекс v мора да важи $0 \leq v < n$. Елемент у врсти v који припада дијагонали d мора да задовољава $v + k = d$, тј. $k = d - v$. Да би елемент припадао матрици мора да важи да је $0 \leq k < n$, тј. $0 \leq d - v < n$, тј. $d - n < v \leq d$, тј. $d - n + 1 \leq v < d + 1$. Пошто је $0 \leq v$ и $d - n + 1 \leq v$, важи да је $\max(0, d - n + 1) \leq v$. Слично, из $v < n$ и $v < d + 1$ следи да је $v < \min(n, d + 1)$. Зато се за фиксирано d елементи одговарајуће дијагонале могу добити тако што вредност v у петљи редом узима вредности од $\max(0, d - n + 1)$, па строго мање од $\min(n, d + 1)$ и тако што се за свако такво v посматра елемент $(v, d - v)$. Поново сабирамо све елементе на дијагонали (алгоритмом сабирања серије елемената), сваки збир поредимо са максималним до тада добијеним збиром и ако је збир текуће дијагонале већи од текућег максимума, ажурирамо вредност максимума (тј. примењујемо алгоритам одређивања максимума серије бројева).

На крају, након обиласка свих дијагонала, пријављујемо вредност до тада пронађеног максимума (он је тада уједно и глобални максимум).

```
const int MAX = 100;

int main() {
    // dimenzije matrice
    int n;
    cin >> n;
    // elementi matrice
    int A[MAX][MAX];
```

```

for (int v = 0; v < n; v++)
    for (int k = 0; k < n; k++)
        cin >> A[v][k];

// najveci zbir postavljamo na "-beskonacno"
int maxZbir = numeric_limits<int>::min();

// proveravamo sve glavne dijagonale
for (int d = -(n-1); d <= n-1; d++) {
    // izracunavamo zbir elemenata na tekucoj glavnoj dijagonali
    int zbir = 0;
    for (int v = max(0, -d); v < min(n, n-d); v++)
        zbir += A[v][v+d];
    // ako je zbir tekuce dijagonalne veci od najveceg dosadasnjeg
    // zbira, azuriramo najveci vidjeni zbir
    if (zbir > maxZbir)
        maxZbir = zbir;
}

// proveravamo sve sporedne dijagonale
for (int d = 0; d <= 2*(n-1); d++) {
    // izracunavamo zbir elemenata na tekucoj sporednoj dijagonali
    int zbir = 0;
    for (int v = max(0, d-n+1); v < min(n, d+1); v++)
        zbir += A[v][d-v];
    // ako je zbir tekuce dijagonalne veci od najveceg dosadasnjeg
    // zbira, azuriramo najveci vidjeni zbir
    if (zbir > maxZbir)
        maxZbir = zbir;
}

// ispisujemo najveci zbir
cout << maxZbir << endl;
return 0;
}

```

Задатак: Спирални испис матрице

Напиши програм који исписује елементе матрице спирално, кренувши од горњег левог угла и кружећи у смеру казаљке на сату.

Улаз: Са стандардног улаза учитава се димензија квадратне матрице n , а затим и елементи матрице.

Излаз: На стандардни излаз исписати низ елемената матрице који се добијају спиралним обиласком.

Пример

Улаз

```

5
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25

```

Излаз

```

1 2 3 4 5 10 15 20 25 24 23 22 21 16 11 6 7 8 9 14 19 18 17 12 13

```

Решење

Примера ради, посматрајмо матрицу


```

1  2  3  4  5
6  7  8  9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25

```

Спирални испис можемо организовати тако што исписујемо један по један “прстен” матрице, од спољашњег ка унутрашњим. У нашем примеру то су

```

1  2  3  4  5      7  8  9      13
6           10     12   14
11          15     17 18 19
16          20
21 22 23 24 25

```

Сваки прстен је одређен својим редним бројем i и вредност броја i се креће од 0 за спољни прстен, па све док је мања од $n/2$. Испис елемената сваког прстена (осим последњег, једночланог) организујемо тако што испишемо његове четири ивице. Започињемо од горње. Индекс врсте свих елемената на горњој ивици прстена је i , док се индекс колоне креће од вредности i , па све док је строго мањи од $n - i$. Преостали елементи на десној ивици имају индекс колоне $n - 1 - i$, а индекс врсте им расте од $i + 1$ па док је строго мањи од $n - i$. Преостали елементи на доњој ивици имају индекс врсте $n - i - 1$, а индекс колоне им опада од вредности $n - i - 2$ па све до i (укључујући и i). На крају преостали елементи леве ивице имају индекс колоне i , док им индекс врсте опада од $n - i - 2$, па све до i (овај пут не укључујући i).

Ако је димензија матрице n непарна, тада се у њеном центру налази последњи, једночлани прстен (заправо само један елемент) и он се исписује само тако што се тај један елемент испише (ово је специјални случај који је потребно проверити након исписа свих прстенова).

```

const int MAX = 100;
int A[MAX][MAX];
int n;
cin >> n;
for (int v = 0; v < n; v++)
    for (int k = 0; k < n; k++)
        cin >> A[v][k];

for (int i = 0; i < n / 2; i++) {
    for (int k = i; k < n-i; k++)
        cout << A[i][k] << " ";
    for (int v = i + 1; v < n-i; v++)
        cout << A[v][n-i-1] << " ";
    for (int k = n-i-2; k >= i; k--)
        cout << A[n-i-1][k] << " ";
    for (int v = n-i-2; v > i; v--)
        cout << A[v][i] << " ";
}

if (n % 2 != 0)
    cout << A[n/2][n/2] << " ";
cout << endl;

```

Један начин да организујемо спирални испис је да у сваком тренутку памтимо четири променљиве `gore`, `dole`, `levo` и `desno` које одређују прстен матрице који тренутно исписујемо. Иницијално, променљиве `gore` и `levo` постављамо на нулу, а `dole` и `desno` на $n - 1$. Након тога, исписујемо горњи ред прстена тако што исписујемо све елементе у врсти `gore`, док се колона мења од `levo` до `desno`. Након тога, увећавамо вредност променљиве `gore`. Затим исписујемо преостале елементе десне ивице тако што исписујемо елементе у колони `desno` за вредности врсте од `gore` до `dole`. Након тога, умањујемо вредност променљиве `desno`. Затим исписујемо преостале елементе доње ивице тако што исписујемо елементе у врсти `dole`, док се индекс колоне мења од `desno` и опада до `levo`. Након тога, умањујемо вредност променљиве `dole`. На крају, елементе леве колоне исписујемо тако што исписујемо елементе у колони `levo` док индекс врсте опада од `dole` па до `gore`. Након тога увећавамо вредност променљиве `levo` и поступак понављамо из почетка. Након сваког увећавања

5.4. ВИШЕДИМЕНЗИОНИ НИЗОВИ И МАТРИЦЕ

променљиве `gore` и умањивања променљиве `dole` проверавамо да ли је вредност `gore` постала строго већа од вредности `dole` - ако јесте, прекидамо поступак. Аналогно, након сваког увећавања променљиве `levo` и умањивања променљиве `desno` проверавамо да ли је вредност `levo` постала строго већа од вредности `desno` - ако јесте, прекидамо поступак.

```
const int MAX = 100;
int A[MAX][MAX];
int n;
cin >> n;
for (int v = 0; v < n; v++)
    for (int k = 0; k < n; k++)
        cin >> A[v][k];

int gore = 0, dole = n-1, levo = 0, desno = n-1;
while (true) {
    for (int k = levo; k <= desno; k++)
        cout << A[gore][k] << " ";
    gore++;
    if (gore > dole) break;
    for (int v = gore; v <= dole; v++)
        cout << A[v][desno] << " ";
    desno--;
    if (desno < levo) break;
    for (int k = desno; k >= levo; k--)
        cout << A[dole][k] << " ";
    dole--;
    if (dole < gore) break;
    for (int v = dole; v >= gore; v--)
        cout << A[v][levo] << " ";
    levo++;
    if (levo > desno) break;
}
cout << endl;
```

Задатак: Множење матрица

Јовани је рођендан и одлучила је да направи тарту да почасте своје другарице и другаре. Она зна рецепте за t различитих торта. У сваку тарту иде s различитих састојака (за сваку тарту позната је количина сваког од тих састојака). Јована ће све ствари набавити у једној од p продавница. За сваку продавницу познате су цене сваког од тих састојака. Напиши програм који помаже Јовани да одреди коју тарту да прави и у којој продавници да купује састојке да би јој остало што више пара за екскурзију.

Улаз: Са стандардног улаза уносе се бројеви t , s и p (сви између 2 и 10), а затим и две матрице. Прва, димензије $t \times s$ одређује количину састојака за сваку од торта (количина је цео број између 1 и 3), а друга, димензије $s \times p$ одређује цену сваког састојка у свакој од продавница (цена је цео број између 100 и 300).

Излаз: На стандардни излаз исписати три цела броја - редни број торте и редни број продавнице (оба се броје од 0), као и укупну цену најјефтиније торте.

Пример

Улаз	Излаз
3 4 2	1 1 1189
2 1 3 2	
3 2 2 1	
2 1 2 3	
250 170	
160 120	
135 142	
145 155	

Решење

Размотримо податке дате у примеру.

```
2 1 3 2   250 170
3 2 2 1   160 120
2 1 2 3   135 142
          145 155
```

Израчунајмо цену прве торте, ако састојке купујемо у првој продавници. Потребно је две јединице првог састојка који кошта 250 динара, једна јединица другог састојка који кошта 160 динара, три јединице трећег састојка који кошта 135 динара и две јединице четвртог састојка који кошта 145 динара. Дакле, укупна цена те торте је $2 \cdot 250 + 1 \cdot 160 + 3 \cdot 135 + 2 \cdot 145 = 1355$. Дакле, да бисмо израчунали цену прве торте, потребно је било да помножимо прву врсту прве матрице, са првом колоном друге матрице (види задатак **Скаларни производ**). Слично, да бисмо израчунали цену торте са редним бројем t у продавници са редним бројем p потребно је да помножимо врсту са индексом t у првој матрици са колоном са редним бројем p у другој матрици. Све тако добијене цене можемо сложити у нову матрицу која ће бити димензије $t \times p$. За матрице из примера добијамо резултат

```
1355 1196
1485 1189
1365 1209
```

Да бисмо израчунали ову матрицу пролазимо кроз све њене елементе (у две угнежђене петље - једној која одређује њену врсту t и креће се од 0, па док је мања од T и другој која одређује њену колону p и креће се од 0, па док је мања од P). У телу тих петљи израчунавамо производ врсте t прве матрице и колоне p друге матрице тако што тај производ (елемент на позицији (t, p) резултујуће матрице) иницијализујемо на нулу, затим у петљи пролазимо кроз све састојке тј. пуштамо да се променљива s мења од 0, па док је мања од S и у сваком кораку производ увећавамо за производ елемента на позицији (t, s) прве матрице (то је количина састојка s у торти t) и елемента на позицији (s, p) у другој матрици (то је цена састојка s у продавници p). Добијена матрица се назива *производ матрица* и операција множења матрица се дефинише баш на начин који смо приказали у овом задатку.

Након одређивања производа, тј. цена свих торти у свим продавницама, потребно је одредити позицију и вредност најјефтиније од њих. То се ради једноставном модификацијом алгоритма за одређивање позиције минимума (види задатак **Редни број максимума**) - позицију минимума иницијализујемо на $(0, 0)$, затим пролазимо кроз све елементе матрице (помоћу две угнежђене петље), поредимо текући елемент са минималним и ако је мањи од минималног, ажурирамо позицију минималног елемента на текућу позицију - нагласимо да није потребно посебно памтити вредност минималног јер се на основу његове позиције у матрици лако може прочитати његова вредност.

```
// odredjuje se proizvod matrica ts dimenzije T puta S i sp dimenzije
// T puta P i rezultat se smesta u matricu tp dimenzije T puta P
void mnoziMatrice(int ts[MAX][MAX], int sp[MAX][MAX], int tp[MAX][MAX],
                  int T, int S, int P) {
    for (int t = 0; t < T; t++)
        for (int p = 0; p < P; p++) {
            // cena torte t u prodavnici p
            tp[t][p] = 0;
            // cenu uvecavamo za cenu svakog sastojka
            for (int s = 0; s < S; s++)
                tp[t][p] += ts[t][s] * sp[s][p];
        }
}

// odredjujemo poziciju (tMin, pMin) minimalnog elementa matrice tp
// dimenzija T puta P
void nadjiMinimum(int tp[MAX][MAX], int T, int P, int& tMin, int& pMin) {
    tMin = 0; pMin = 0;
    for (int t = 0; t < T; t++)
        for (int p = 0; p < P; p++)
            if (tp[t][p] < tp[tMin][pMin]) {
                tMin = t; pMin = p;
            }
}
```

```
    }  
}  
  
// ucitava matricu sa standardnog ulaza  
void ucitaj(int A[MAX][MAX], int m, int n) {  
    for (int v = 0; v < m; v++)  
        for (int k = 0; k < n; k++)  
            cin >> A[v][k];  
}  
  
int main() {  
    // broj torti, sastojaka i prodavnica  
    int T, S, P;  
    cin >> T >> S >> P;  
    // kolicina sastojaka za svaku tortu  
    int ts[MAX][MAX];  
    ucitaj(ts, T, S);  
    // cena sastojaka u svakoj prodavnici  
    int sp[MAX][MAX];  
    ucitaj(sp, S, P);  
    // cena svake torte u svakoj prodavnici  
    int tp[MAX][MAX];  
    mnoziMatrice(ts, sp, tp, T, S, P);  
    // najjeftinija torta  
    int tMin, pMin;  
    nadjiMinimum(tp, T, P, tMin, pMin);  
    cout << tMin << " " << pMin << " " << tp[tMin][pMin] << endl;  
    return 0;  
}
```

Нагласимо и да се задатак могао решити без памћења целе резултујуће матрице, тако што би се након одређивања сваког њеног елемента он поредио са вредношћу текућег минимума (који би у том случају требало памтити).

```
// sa standardnog ulaza se ucitava se matrica datih dimenzija  
void ucitaj(int A[MAX][MAX], int m, int n) {  
    for (int v = 0; v < m; v++)  
        for (int k = 0; k < n; k++)  
            cin >> A[v][k];  
}  
  
int main() {  
    // broj torti, sastojaka i prodavnica  
    int T, S, P;  
    cin >> T >> S >> P;  
    // kolicina sastojaka za svaku tortu  
    int ts[MAX][MAX];  
    ucitaj(ts, T, S);  
    // cena sastojaka u svakoj prodavnici  
    int sp[MAX][MAX];  
    ucitaj(sp, S, P);  
  
    // tekuca pozicija i cena najjeftinije torte  
    int tMin = 0, pMin = 0;  
    int minCena = numeric_limits<int>::max();  
    for (int t = 0; t < T; t++)  
        for (int p = 0; p < P; p++) {  
            // cena torte t u prodavnici p  
            int cena = 0;
```

```

// sabiramo cene svih sastojaka
for (int s = 0; s < S; s++)
    cena += ts[t][s] * sp[s][p];
// ako je tekuca cena jeftinija od dosada najjeftinije
if (cena < minCena) {
    // azuriramo najjeftiniju cenu i njenu poziciju
    minCena = cena;
    tMin = t; pMin = p;
}
}

// prijavljujemo rezultat
cout << tMin << " " << pMin << " " << minCena << endl;
return 0;
}

```

Задатак: Релација зависности

На факултету постоје зависности између испита који се полагају. На пример, да би се полагао испит “Програмирање 2”, претходно је неопходно да је положен испит “Програмирање 1”. Зависности су задате матрицом логичких вредности која представља једну релацију. Напиши програм који проверава да ли су испуњени следећи услови:

- релација је антирефлексивна тј. не постоји ни један предмет који зависи сам од себе,
- релација је антисиметрична тј. не постоје два предмета који међусобно зависе један од другог,
- релација је транзитивна тј. ако један предмет зависи од другог, а тај други зависи од трећег, онда је неопходно да и први предмет зависи од трећег.

Улаз: Са стандардног улаза се уноси број предмета n ($5 \leq n \leq 10$), а затим матрица димензије $n \times n$ која садржи само нуле и јединице - јединица у врсти v и колони k означава да предмет са редним бројем v зависи од предмета са редним бројем k .

Излаз: На стандардни излаз исписати DA ако матрица испуњава све задате услове тј. NE ако нарушава било који од њих.

Пример

Улаз	Излаз
6	DA
0 0 0 0 0 0	
1 0 1 0 1 0	
1 0 0 0 1 0	
1 1 1 0 1 0	
1 0 0 0 0 0	
0 0 0 0 0 0	

Решење

Релација је антирефлексивна ако на дијагонали матрице не постоји ни једна јединица тј. ниједна вредност true ако се за елементе матрице користи логички тип. Примењујемо, дакле, алгоритам линеарне претраге (види задатак **Негативан број**) који се најједноставније имплементира у склопу посебне функције, пролазимо кроз дијагоналне елементе (то су они који имају индексе облика (i, i) за вредности i од 0 па док је i строго мање од n , проверавамо да ли је на дијагонали true и ако јесте, прекидамо извршавање функције враћајући вредност false (матрица није антирефлексивна). Ако се пролазак кроз серију дијагоналних елемената заврши, то значи да није пронађена ниједна вредност true на дијагонали, тако да након петље, на крају функције можемо да вратимо вредност true (матрица јесте антирефлексивна).

Антисиметричност проверавамо на сличан начин. Опет алгоритмом претраге проверавамо да ли постоји неки пар различитих индекса (v, k) такав да и на позицији (v, k) и на позицији (k, v) у матрици стоји вредност true. Довољно је претражити само доњи (или само горњи троугао), јер ако постоји таква вредност са $k > v$, тада ће постојати таква вредност и за $v > k$.

На крају, транзитивност проверавамо тако што за сваки пар тројки (i, j, k) проверавамо да ли је нарушен услов да ако је на позицији (i, j) вредност true и на позицији (j, k) вредност true, тада је и на позицији (i, k)

5.4. ВИШЕДИМЕНЗИОНИ НИЗОВИ И МАТРИЦЕ

вредност `true`. Дакле, опет у склопу посебне функције вршимо линарну претрагу, помоћу три угнежђене петље набрајамо све тројке (i, j, k) и проверавамо да ли је у матрици на позицијама (i, j) и (j, k) вредност `true`, а на позицији (i, j) вредност `false` - ако јесте враћамо вредност `false` (релација није транзитивна). На крају, ако прођемо све тројке, враћамо вредност `true` (релација јесте транзитивна јер нисмо наишли ни на један елемент који нарушава транзитивност).

```
// provera da li je matrica A dimenzije n antisimetricna
bool jeAntisimetricna(bool A[MAX][MAX], int n) {
    // trazimo da li postoji par indeksa (v, k) takav da je A[v][k] =
    // true i A[k][v] = true
    for (int v = 0; v < n; v++)
        for (int k = 0; k < v; k++)
            if (A[v][k] && A[k][v])
                return false;
    return true;
}

// provera da li je matrica A dimenzije n antirefleksivna
bool jeAntirefleksivna(bool A[MAX][MAX], int n) {
    // trazimo da li na dijagonali postoji vrednost true
    for (int i = 0; i < n; i++)
        if (A[i][i])
            return false;
    return true;
}

// provera da li je matrica A dimenzije n tranzitivna
bool jeTranzitivna(bool A[MAX][MAX], int n) {
    // trazimo da li postoji trojka (i, j, k) takva da je
    // A[i][j] = true, A[j][k] = true, ali da ne vazi A[i][k] = true
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
                if (A[i][j] && A[j][k] && !A[i][k])
                    return false;
    return true;
}

int main() {
    // učitavamo matricu
    int n;
    cin >> n;
    bool A[MAX][MAX];
    for (int v = 0; v < n; v++)
        for (int k = 0; k < n; k++)
            cin >> A[v][k];

    // proveravamo uslove i ispisujemo rezultat
    if (jeAntisimetricna(A, n) && jeAntirefleksivna(A, n)
        && jeTranzitivna(A, n))
        cout << "DA" << endl;
    else
        cout << "NE" << endl;
    return 0;
}
```

Задатак: Преседање

Дата је квадратна матрица A димензије N , која представља директне авионске везе између N градова. Елемент $a_{i,j}$ је једнак један ако постоји директан лет из града i за град j , а нула иначе.

Одредити и исписати матрицу исте димензије, која приказује везе између истих градова уз највише једно преседање. Сматрати да је сваки град у директној вези са самим собом, без обзира на улазне вредности.

Улаз: У првом реду стандардног улаза налази се број N , број градова ($1 \leq N \leq 10$). У наредних N редова налази се низ од N нула или јединица. Јединица у реду i на месту j значи да постоји директан лет од града i до града j , а нула да не постоји.

Излаз: На стандардни излаз исписати N редова, у сваком по једну врсту резултујуће матрице. Елемент на позицији (i, j) резултујуће матрице једнак је 1 ако постоји директна веза, или веза са једним преседањем између одговарајућих градова, а 0 иначе.

Пример

Улаз	Излаз
4	1 1 1 1
0 1 0 1	1 1 0 1
1 0 0 0	1 1 1 1
1 1 0 0	1 1 1 1
0 0 1 0	

Објашњење

Из првог у трећи град се стиже преко четвртог града.

Из другог у четврти град се стиже преко првог града.

Из трећег у четврти град се стиже преко првог града.

Из четвртог у први град се стиже преко трећег града.

Из четвртог у други град се стиже преко трећег града.

Задатак: Особине релације

Релација на скупу од N елемената може се представити квадратном логичком матрицом величине $N \times N$.

Испитати да ли је релација дата матрицом

- рефлексивна
- симетрична
- антисиметрична
- транзитивна

Напомена

Релација ρ је рефлексивна на скупу S ако и само ако је $(\forall x \in S)(x\rho x)$, тј. ако и само ако је сваки елемент у релацији са собом. На пример, релација “бити подударан” је рефлексивна на скупу тачака у равни, а релација “бити мањи” није рефлексивна на скупу целих бројева.

Релација ρ је симетрична на скупу S ако и само ако је $(\forall x \in S)(\forall y \in S)(x\rho y \iff y\rho x)$, тј. ако и само ако редослед елемената у релацији није битан. На пример, релација “бити исте парности” је симетрична на скупу целих бројева, а релација “бити мањи” није симетрична на скупу целих бројева.

Релација ρ је антисиметрична на скупу S ако и само ако је $(\forall x \in S)(\forall y \in S)(x\rho y \wedge y\rho x \implies x = y)$, тј. ако и само не постоје два различита елемента који су међусобно у релацији у оба смера. На пример, релација “бити дељив” је антисиметрична на скупу природних бројева, а релација “бити исте парности” није антисиметрична на скупу целих бројева.

Релација ρ је транзитивна на скупу S ако и само ако је $(\forall x \in S)(\forall y \in S)(\forall z \in S)(x\rho y \wedge y\rho z \implies x\rho z)$, тј. ако и само кад год је један елемент у релацији са другим, а други са трећим, онда је и први елемент у релацији са трећим. На пример, релација “бити дељив” је транзитивна на скупу природних бројева, а релација “бити бољи” није транзитивна на скупу $\{, , \}$ у познатој игри (папир је бољи од камена а камен од маказа, али папир није бољи од маказа).

5.4. ВИШЕДИМЕНЗИОНИ НИЗОВИ И МАТРИЦЕ

Улаз: У првом реду стандардног улаза налази се број N , број елемената скупа ($1 \leq N \leq 10$). У наредних N редова налази се низ од N нула или јединица. Нула у реду i на месту j значи да i -ти елемент није у релацији са j -тим, а јединица значи да јесте.

Излаз: У свакој од 4 линије излаз треба исписати реч **да** или **не**. Исписане речи су редом одговори на питања да ли је релација која је задата матрицом рефлексивна, симетрична, антисиметрична и транзитивна.

Пример

Улаз	Излаз
3	да
1 1 0	не
1 1 0	не
1 1 1	да

Задатак: Асоцијативност

Нека операција \star задата је на скупу целих бројева од 0 до $n - 1$ помоћу матрице A димензија $n \times n$, тако да је $i \star j = A_{i,j}$, при чему су индекси (а такође и елементи матрице, тј. резултати операције) цели бројеви из интервала $[0, n - 1]$.

Написати програм који учитава квадратну матрицу и проверава да ли је операција дата том матрицом асоцијативна, тј. да ли за свако i, j, k из интервала $[0, n - 1]$ важи: $(i \star j) \star k = i \star (j \star k)$.

Улаз: У првом реду стандардног улаза налази се број n , број елемената скупа на коме је дефинисана операција ($1 \leq n \leq 10$). У наредних n редова налази се по један низ од n целих бројева из интервала $[0, n - 1]$, раздвојених по једним размаком.

Излаз: На стандардни излаз исписати само реч **да** или **не** (да ако је операција асоцијативна, а не ако није).

Пример

Улаз	Излаз
4	да
2 3 0 1	
3 0 1 2	
0 1 2 3	
1 2 3 0	

Задатак: Гарантовани износ

Два играча играју следећу игру: из дате матрице играч R бира ред а играч K колону и након тога играч R плаћа играчу K своту која је у пресеку изабране врсте и колоне (ако је свота негативна, играч K плаћа играчу R апсолутну вредност своте).

Који је најповољнији исход који може гарантовати сваки од играча, ако мора први да бира?

Улаз: У првом реду стандардног улаза су два природна броја m и n , број врста и број колона матрице ($1 \leq m \leq 10, 1 \leq n \leq 10$). У следећих m редова налази се по n целих бројева $a_{i,j}, j = 0, 1, \dots, n - 1$, раздвојених по једним размаком, који представљају један ред матрице (за сваки елемент матрице важи $-10^9 \leq a_{i,j} \leq 10^9$).

Излаз: У првом реду исписати своту коју може да гарантује играч R , као најповољнију за себе.

У другом реду исписати своту коју може да гарантује играч K , као најповољнију за себе.

Пример

Улаз	Излаз
2 3	1
-2 1 0	0
0 -1 2	

Објашњење

Ако први бира играч R , он може да гарантује да неће платити више од 1 избором горње врсте.

Ако први бира играч K , он може да гарантује да неће примити мање од 0 избором последње колоне.

Задатак: Осигурање

Компанија која се бави осигурањем покрива D врста штетних догађаја, а исплаћује осигуранику S_j динара у случају j -те врсте штетног догађаја, $0 \leq j < D$. Осим тога, за сваког од K клијената (осигураника) процењена је вероватноћа сваке врсте штетног догађаја $P_{i,j}$, $0 \leq i < K, 0 \leq j < D$ (i је редни број клијента, а j редни број врсте штетног догађаја).

Претпостављајући да штетни догађаји наступају независно један од другог, компанија израчунава очекивану исплату клијенту i као

$$\sum_{j=0}^{D-1} P_{i,j} S_j = P_{i,0} S_0 + P_{i,1} S_1 + \dots + P_{i,D-1} S_{D-1}.$$

Одредити редни број i оног осигураника који је најризичнији за осигуравајућу компанију, то јест оно i за које је наведена сума највећа. У случају да има више таквих клијената, одредити најмањи од њихових редних бројева.

Улаз: У првом реду стандардног улаза налазе се бројеви D и K , број врста штетних догађаја и број клијената ($1 \leq D \leq 10, 1 \leq K \leq 20$). У следећем реду је D реалних позитивних бројева раздвојених по једним размаком, који представљају своте које се исплаћују за сваки од штетних догађаја. У сваком од наредних K редова се налази D реалних бројева из интервала $[0, 1]$ раздвојених по једном размаком, који представљају редом вероватноће штетних догађаја за једног клијента.

Издаз: Исписати само један цео број, (најмањи) редни број клијента за ког је очекивана исплата максимална.

Пример

<i>Улаз</i>	<i>Издаз</i>
3 5	3
50000 15500 100000	
0.002 0.019 0.019	
0.003 0.021 0.013	
0.001 0.016 0.017	
0.001 0.014 0.021	
0.002 0.022 0.018	

Објашњење

Очекиване исплате штете за клијенте су редом 2294.5, 1775.5, 1998, 2367, 2241. Највећа од њих је 2367, која се добија за клијента са редним бројем 3.

Задатак: Магичан квадрат

Матрица $n \times n$ целих бројева је магичан квадрат ако су зборови свих врста и свих колона исти. Магичан квадрат је јак магичан квадрат ако су елементи матрице сви бројеви из скупа $\{1, 2, \dots, n^2\}$. Написати програм који проверава да ли је дата матрица магичан квадрат, и ако јесте, да ли је јак магичан квадрат.

Улаз: У првом реду стандардног улаза је природан број n , који представља димензију матрице ($2 \leq n \leq 10$).

У следећих n редова налази се по n целих бројева, раздвојених по једним размаком, који представљају по један ред матрице. За сваки елемент матрице важи $1 \leq a_{i,j} \leq n^2$.

Издаз: На стандардни издаз исписати један од текстова "nije magican", "magican" или "jak magican" (без наводника).

Пример

<i>Улаз</i>	<i>Издаз</i>
4	jak magican
1 15 14 4	
12 6 7 9	
8 10 11 5	
13 3 2 16	

5.4. ВИШЕДИМЕНЗИОНИ НИЗОВИ И МАТРИЦЕ

Задатак: Матрица 1248

Дата је матрица формата $V \times K$ попуњена на неки начин бројевима 1, 2, 4, 8. Одредити број подматрица формата 2×2 код којих су сви елементи различити.

Улаз: У првом реду стандардног улаза налазе се бројеви V и K , који представљају димензије матрице ($2 \leq V \leq 10, 2 \leq K \leq 10$). У сваком од наредних V редова се налази K бројева из скупа $\{1, 2, 4, 8\}$ раздвојених по једним размаком, који представљају један ред матрице.

Изназ: Исписати само један цео број, број тражених подматрица.

Пример

Улаз	Изназ
3 4	3
1 2 1 4	
4 8 8 2	
1 2 8 4	

Задатак: Премештаљка

Премештаљка има облик матрице од V врста и K колона. Једно поље премештаљке је празно, а на осталим местима су плочице са бројевима од 1 до $m \cdot n - 1$. Плочица поред празне позиције може са њом да замени место. Померање неке плочице на лево, на десно, навише, односно наниже означаваћемо редом са L, D, V, N.

За дате димензије матрице, почетни распоред бројева (празнина је означена нулом) и низ потеза претстављен словима L, D, V, N, приказати распоред бројева након овог низа потеза.

Улаз: У првом реду стандардног улаза налазе се бројеви V и K , димензије матрице ($2 \leq V \leq 10, 2 \leq K \leq 10$). У наредних K редова налази се низ од V бројева из опсега $[0, m \cdot n - 1]$ раздвојених по једним размаком, тако да се сваки од тих бројева појављује по једном у матрици. Након тога, у последњем реду је ниска слова из скупа $\{L, D, V, N\}$, дужине не веће од 20.

Подаци су такви да је премештања увек могуће извести.

Изназ: У V редова исписати редом врсте матрице бројева која се добија када се на датој матрици изведу премештања задата ниском слова.

Пример

Улаз	Изназ
3 4	1 2 3 4
1 2 3 4	5 10 6 8
5 6 7 8	9 7 0 11
9 10 11 0	

DNDVL

Задатак: Троуглови и широка дијагонала

Ученици три одељења су села на столице у биоскопској сали квадратног облика. Ученици једног одељења су попунили места на главној дијагонали сале и неколико дијагонала уз њу, ученици другог одељења су попунили горњи, а трећег одељења доњи троугао сале. На пример, ако је димензија сале 7, а ако су ученици првог одељења сели на две дијагонале уз главну, онда је распоред седења следећи:

```
1112222
1111222
1111122
3111112
3311111
3331111
3333111
```

Позната је висина сваког ученика. Напиши програм који одређује просечну висину ученика сваког одељења.

Улаз: Са стандардног улаза учитава се број n који представља димензију сале ($5 \leq n \leq 10$), затим, квадратна матрица димензије $n \times n$ која садржи висине ученика (реалне бројеве између 1.1 и 1.80) и на крају, у следећем

реду, број m који одређује број дијагонала око главне на којима су распоређени ученици првог одељења ($0 \leq m \leq 3$).

Излаз: На стандардни излаз исписати просечну висину ученика одељења 1, одељења 2 и одељења 3 (сваки број у посебном реду, заокружен на две децимале).

Пример

<i>Улаз</i>	<i>Излаз</i>
7	1.43
1.31 1.58 1.13 1.49 1.70 1.17 1.36	1.45
1.39 1.78 1.42 1.38 1.42 1.54 1.68	1.48
1.28 1.10 1.42 1.52 1.79 1.19 1.33	
1.18 1.44 1.33 1.23 1.55 1.33 1.59	
1.24 1.79 1.17 1.33 1.76 1.19 1.44	
1.63 1.50 1.41 1.11 1.56 1.61 1.80	
1.51 1.76 1.48 1.33 1.74 1.30 1.55	
2	

Задатак: Врхови планине

Планинска експедиција се кретала правоугаоним тереном и у правилним интервалима мерила је надморску висину. Претпостављамо да су прикупљене висине у матрици димензија $m \times n$. Када се налазе у некој тачки, планинари могу да гледају на исток, запад, север и југ, али и на североисток, северозапад, југоисток и југозапад. У сваком од тих 8 смерова планинари могу да виде неколико врхова. Тачка се сматра видљивом, ако се испред ње налазе само тачке које имају строго мању надморску висину од ње (или ако испред ње нема других тачака).

Улаз: Опис улазних података.

Излаз: Опис излазних података.

Пример

<i>Улаз</i>	<i>Излаз</i>
4 5	11
3 8 5 4 2	
4 1 7 6 5	
2 1 4 8 9	
3 3 7 5 2	
1 2	

Задатак: Турнир

Током једне кошаркашке сезоне неки тимови су се састајали више пута. Ако је познат резултат сваког њиховог сусрета, напиши програм који одређује укупан резултат свих њихових сусрета након те сезоне.

Улаз: У првом реду стандардног улаза уноси се укупан број одиграних утакмица n ($1 \leq n \leq 100$). У следећих n редова уносе се по 4 ненегативна цела броја p, q, r, s који описују једну утакмицу, тако да су p и q ($1 \leq p, q \leq 10$) редни бројеви кошаркашких тимова који су одиграли ту утакмицу с резултатом $r : s$ ($0 \leq r, s \leq 150$) тј. тим p је дао r кошева, док је тим q дао s кошева. Након тога се уноси природни број m ($1 \leq m \leq 20$), а потом m парова природних бројева p и q ($1 \leq p, q \leq 10$).

Излаз: За сваки пар тимова p и q , у посебној линији испиши колико укупно кошева је тим p дао тиму q и колико је тим q дао тиму p , раздвојене двотачком.

5.4. ВИШЕДИМЕНЗИОНИ НИЗОВИ И МАТРИЦЕ

Пример

Улаз	Израз
5	38:79
1 2 15 70	138:132
2 3 90 88	
1 2 23 9	
3 1 88 86	
2 3 42 50	
2	
1 2	
3 2	

Задатак: Потез у игри 2048

На интернету је веома популарна игра [2048](#). На квадратој табли неке плочице садрже степене броја 2, а нека поља су празна. Играч игра потез тако што се одлучи за један смер (налево, надесно, наниже или навише) и затим помера све плочице преко празних поља у том смеру, колико год је то могуће. При том се две плочице на којима пише исти број, а које су које су суседне (или које постају суседне након померања плочица по празним пољима) спајају у једну плочицу на којој је дупло већи број (спајање се врши у правцу кретања). Плочице настале спајањем се не спајају даље током истог потеза. Слободно играјте мало игрицу да бисте схватили правила.

Улаз: Са стандардног улаза се уноси број n ($1 \leq n \leq 10$), а затим и матрица димензије $n \times n$ на којој су празна поља означена вредношћу 0, а плочице степенима двојке. У наредном реду се налази један карактер који одређује смер померања (L - налево, D - надесно, V - навише, N - наниже).

Израз: На стандардни излаз исписати матрицу која представља стање табле за игру након одиграног потеза.

Пример 1

Улаз	Израз
4	0 0 0 0
0 0 0 2	0 0 0 0
0 0 2 0	0 0 0 0
0 2 0 0	2 2 2 2
2 0 0 0	
N	

Пример 2

Улаз	Израз
4	0 0 0 4
0 0 2 2	0 0 0 4
0 2 0 2	0 0 0 4
2 0 0 2	0 0 4 4
2 2 2 2	
D	

Пример 3

Улаз	Израз
4	1 2 4 8
1 2 4 8	1 2 4 0
1 2 4 0	1 2 4 0
0 1 2 4	1 2 8 0
1 2 4 4	
L	

Пример 4

Улаз	Израз
4	2 4 4 4
0 0 0 2	0 0 2 4
0 0 2 2	0 0 0 0
0 2 2 2	0 0 0 0
2 2 2 2	
V	

Задатак: Сортирање по просеку

Наставник је записао табелу са закључним оценама ученика. Напиши програм који сортира ученике по просечној оцени, нерастуће. Ако два ученика имају исту просечну оценоу они треба да остану у редоследу у ком су били на почетку.

Улаз: Са стандардног улаза се учитава број ученика u ($5 \leq u \leq 50000$), затим број оцена o ($5 \leq o \leq 100$) и након тога у u наредних линија оценоу за сваког ученика (o оцена раздвојене размацама).

Израз: На стандардном излазу исписати сортиране оценоу (у истом формату у којем су и унете).

Пример

Улаз	Изназ
6 5	5 5 5 4 5
3 5 5 4 2	5 4 5 5 5
5 4 2 2 5	3 5 5 4 2
5 5 5 4 5	5 4 2 2 5
3 3 3 2 1	4 2 5 1 3
4 2 5 1 3	3 3 3 2 1
5 4 5 5 5	

5.4.3 Разуђени низови

Задатак: Електронски дневник

Наставница жели да оцене својих ученика препише у дневник, али је приметила да се редослед ученика у свесци и дневнику не поклапа. Напиши програм који исписује све оцене у жељеном редоследу.

Улаз: Прва линија стандардног улаза садржи број ученика n (природан број између 1 и 30). Наредних n линија садрже оцене ученика (свака линија садржи између два и десет бројева од 1 до 5, раздвојене размацама). Наредних n линија описују редослед у коме је потребно исписати оцене (у свакој линији налази се један број од 1 до n и бројеви у свих n линија су различити).

Изназ: На стандардни излаз исписати све оцене, а затим и просек оцена заокружен на две децимале у редоследу који је задат на улазу.

Пример

Улаз	Изназ
4	3 3 4 3.33
5 4 5	3 5 4 2 2 3.20
4 4 4 2	5 4 5 4.67
3 3 4	4 4 4 2 3.50
3 5 4 2 2	
3	
4	
1	
2	

Решење

Пошто сваки ученик има различити број оцена, а све оцене морамо истовремено складиштити због каснијег исписа, употребићемо разуђен низ.

У језику C++ можемо да употребимо вектор вектора (тип `vector<vector<int>>>`). Вектор вектора алоцирамо након читавања броја ученика, а онда за сваког ученика читавамо број оцена и тада алоцирамо одговарајући вектор који чува оцене тог ученика (то можемо урадити методом `resize`).

Просек низа оцена можемо рачунати на исти начин као у задатку [Средине](#).

```
double prosek(const vector<int>& ocene) {
    return accumulate(begin(ocene), end(ocene), 0.0) / ocene.size();
}
```

```
int main() {
    int brojUcenika;
    cin >> brojUcenika;
    vector<vector<int>> ocene(brojUcenika);
    for (int i = 0; i < brojUcenika; i++) {
        int brojOcena;
        cin >> brojOcena;
        ocene[i].resize(brojOcena);
        for (int j = 0; j < brojOcena; j++)
            cin >> ocene[i][j];
    }
}
```

```
for (int i = 0; i < brojUcenika; i++) {
    int rbr;
    cin >> rbr;
    for (int ocena : ocene[rbr-1])
        cout << ocena << " ";
    cout << fixed << showpoint << setprecision(2)
        << prosek(ocene[rbr-1]) << endl;
}
return 0;
}
```

Ова збирка се бави елементарним алгоритмима и погодна је за почетни ниво учења програмирања у гимназијама, стручним школама и на факултетима, као и почетни ниво припреме за такмичења из програмирања.

Збирка представља штампани извод шире збирке доступне онлајн, на порталу petlja.org. У онлајн издању збирке омогућено је аутоматско тестирање решења за сваки задатак (подаци се у свим задацима учитавају са стандардног улаза и исписују се на стандардни излаз).

По завршетку збирке, читалац може наставити да се бави програмирањем на различите начине. Један од могућих наставака је изучавање сложености алгоритама и налажење ефикасних решења проблема. Тим темама је посвећен други том ове збирке, доступан на порталу petlja.org

