Садржај

Предговор

Материјал који је пред вама писан је као збирка за предмет „Увод у програмирање“ са прве године смера Информатика на Математичком факултету у Београду.

Ово је радна верзија материјала и у наредном периоду сигурно ће се мењати и дотеривати. Сви коментари и сугестије биће веома добродошли.

октобар 2024

Аутор

Филип Марић

Глава 1

Аритметички изрази

У меморији рачунара подаци се чувају у склопу *променљивих*. Свака променљива има придружен тип. Иако језик C++ подржава већи број типова за представљање целих и реалних бројева, чијим се избором прави баланс између заузећа меморије и распона вредности који се могу представити, у наставку ћемо за целе бројеве користити увек тип `int`, а за реалне бројеве увек тип `double`. Целобројне константе (на пример, `3`, `123`, `-14`) су типа `int`, а реалне (на пример, `1.0`, `-3.14`) су типа `double`.

Учитавање вредности променљиве (било целобројне, било реалне) са стандардног улаза врши се наредбом облика:

```
cin >> x;
```

Испис израза (било целобројног, било реалног) на стандардни излаз врши се наредбом облика:

```
cout << izraz << endl;
```

Приказ реалног броја `x` заокругженог на две децимале на стандардни излаз се може постићи наредбом

```
cout << fixed << showpoint << setprecision(2) << x << endl;
```

Навођењем `fixed` постижемо да се број прикаже у тзв. фиксном зарезу (а не у тзв. покретном зарезу, тј. уз коришћење неког експонента броја 10, чиме би се скратио запис). Навођењем `showpoint` се постиже то да се децимале увек наводе (чак и када је вредност иза децималог зареза тј. тачке нула). Навођењем `setprecision` подешава се жељени број децимала.

Основне *аритметичке операције* су сабирање (оператор `+`), одузимање (оператор `-`), множење (оператор `*`) и дељење (оператор `/`).

Када су оба операнда целобројног типа (било променљиве, било константе) врши се целобројно дељење (што понекад може довести до нежељених резултата). Чим је бар један операнд реалног типа, врши се реално дељење. Вредност `x` целобројног типа `int` се може експлицитно конвертовати у тип `double` коришћењем експлицитне конверзије облика `(double)x`, док се вредност `x` типа `double` може конвертовати у тип `int` коришћењем експлицитне конверзије облика `(int)x`.

У израчунавањима се користе разне функције дефинисане у заглављу `<cmath>`. Наведимо неке:

- `sqrt` - израчунава квадратни корен
- `pow` - израчунава степен (и изложилац и основа могу бити реални бројеви)
- `sin`, `cos`, `tan`, `cot` - тригонометријске функције
- `asin`, `acos`, `atan`, `acot` - инверзне тригонометријске функције
- `exp`, `log`, `log2`, `log10` - експоненцијална функција за основу e и логаритамске функције за основе редом e, 2 и 10
- `floor`, `ceil`, `round` - заокруживање наниже, навише и на најближи цео број

Све ове функције примају аргументе типа `double` и враћају резултат типа `double`.

1.1 Основне аритметичке операције над реалним и целим бројевима

Задатак: Кифла и јогурт

Написати програм који одређује кусур који треба да буде враћен купцу који је купио одређени број кифли и одређени број чаша јогурта.

Опис улаза

Са стандардног улаза се уноси цена једне кифле (цео број), број купљених кифли, затим цена једне чаше јогурта (цео број), број купљених чаша јогурта и износ новца који је купац дао (претпоставити да је сигурно већи од износа који је потребно платити).

Опис излаза

На стандардни излаз исписати износ који је потребно да буде враћен купцу.

Пример

Улаз	*Излаз*
20 4 45 2 500	330

Решење

До решења се лако долази ако се од уплаћеног износа одузме укупна цена свих производа која се израчунава на основу формуле $n_k \cdot c_k + n_j \cdot c_k$, где су n_k и n_j број продатих кифли и чаша јогурта, а c_k и c_j цена једне кифле и једне чаше јогурта.

```
#include <iostream>

using namespace std;

int main() {
  // izracunavamo koliko kostaju kifle i jogurt
  int cena_kifle, broj_kifli, cena_jogurta, broj_jogurta;
  cin >> cena_kifle >> broj_kifli;
  cin >> cena_jogurta >> broj_jogurta;
  int ukupan_iznos = cena_kifle * broj_kifli + cena_jogurta * broj_jogurta;
  // ucitavamo koliko novca je kupac uplatio
  int placen_iznos;
  cin >> placen_iznos;
  // izracunavamo i ispisujemo kusur
  int kusur = placen_iznos - ukupan_iznos;
  cout << kusur << endl;
  return 0;
}
```

Задатак: Концентрација мешавине

У хемији концентрација неког раствара представља однос масе растворене супстанце и запремине раствора. Написати програм који одређује концентрацију након мешања два раствора исте супстанце чије су концентрације и запремине познате.

Напомена: Ако први раствор има концентрацију C_1 и запремину V_1, а други концентрацију C_2 и запремину V_2, маса растворене супстанце у првом раствору је C_1V_1; а у другом C_2V_2, па је концентрација резултујућег раствора, тј. однос укупне масе и укупне запремине једнака

$$C = \frac{C_1V_1 + C_2V_2}{V_1 + V_2}$$

Опис улаза

Са стандардног улаза се уносе 4 реална броја: C_1, V_1, C_2 и V_2. Концентрације су дате у грамима по центиметру кубном, а запремине у центиметрима кубним.

Опис излаза

На стандардни излаз исписати резултујућу концентрацију (у грамима по центиметру кубном). Резултат заокружити на три децимале.

Пример

Улаз	*Излаз*
0.2 4 0.5 2	0.300

Решење

Програм се своди на то да се дата формула запише у програму. Пошто су промењљиве реалног типа, нема проблема са дељењем. Потребно је само водити рачуна о томе да се и именилац и бројилац разломка морају навести у заградама.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
  double C1, V1, C2, V2;
  cin >> C1 >> V1 >> C2 >> V2;
  double C = (C1*V1 + C2*V2) / (V1 + V2);
  cout << fixed << showpoint << setprecision(3) << C << endl;
  return 0;
}
```

Задатак: Кречење

Написати програм који одређује запремину (у литрима) боје потребну да би се окречили зидови и плафон просторије правоугаоног облика. Рачунати да прозори и врата смањују површину сваког зида који се кречи за 15% (ово не важи за плафон).

Опис улаза

Са стандрадног улаза се учитава ширина, дужина и висина просторије (у метрима), као и површина у метрима квадратним која се може окречити помоћу једног литра боје. У питању су позитивни реални бројеви.

Опис излаза

Резултат исписати на стандардни излаз, заокружен на две децимале.

Пример

Улаз	*Излаз*
4 3.2 2.65 10.5	4.31

Решење

Обележимо ширину и дужину просторије са a и b, а висину са h. Површина плафона која се кречи је $a \cdot b$. Површина зидова једнака је производу обима и висине просторије тј. $2(a + b) \cdot h$, међутим, кречи се само 85% те површине (због прозора и врата). Укупна површина која се кречи је зато $a \cdot b + 0,85 \cdot 2(a + b) \cdot h$. Број литара боје се добија дељењем површине која се кречи са површином која се може окречити за један литар боје.

```cpp
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
  double sirina, duzina, visina;
  cin >> sirina >> duzina >> visina;
  double povrsinaPoLitru;
  cin >> povrsinaPoLitru;
  double plafon = sirina * duzina;
  double obim = 2 * (sirina + duzina);
  double zidovi = 0.85 * obim * visina;
  double povrsina = plafon + zidovi;
  double litri = povrsina / povrsinaPoLitru;
  cout << fixed << showpoint << setprecision(2) << litri << endl;
  return 0;
}
```

Задатак: Екранске координате

Програм који омогућава цртање графика математичке функције тај график исцртава одређујући положај неких карактеристичних тачака са тог графика на екрану. Декартов координатни систем у ком се црта график треба да буде постављен тако да се на екрану види део x-осе између координата x_{min} и x_{max} и део y-осе између y_{min} и y_{max}. Са друге стране, библиотека за цртање захтева да се задају координате у екранском координатном систему у ком се координатни почетак налази у горњем левом углу екрана, x-координата расте слева надесно, док y-координата расте одозго наниже, ширина екрана је *width*, а висина екрана је *height*. Написати програм који на основу датих x и y координата тачке на графику одређује координате те тачке у екранском координатном систему.

Опис улаза

- Са стандардног улаза се прво уносе реални бројеви x_{min} и x_{max} (важи $x_{min} < x_{max}$).
- Затим се уносе реални бројеви y_{min} и y_{max} (важи $y_{min} < y_{max}$).
- Затим се уносе позитивни цели бројеви *width* и *height*.
- На крају се уносе реални бројеви x и y.

Опис излаза

На стандардни излаз исписати целе бројеве који представљају координате тачке у координатном систему екрана (заокружити вредности на најближе целобројне и исписати их раздвојене размаком).

Пример

Улаз	*Излаз*

`-1 2.5`
`0.5 3.2`
`800 600`
`0.3 2.7`
	`297 111`

Решење

Функције којима се врши трансформација координата су линеарне. Одредимо прво линеарну функцију која вредност x_{min} пресликава у 0, а вредност x_{max} у *width*.

Та функција је облика $x_e = kx + n$. Коришћењем услова добија се систем једначина $0 = kx_{min} + n$ и $width = kx_{max} + n$. Рењавањем овог система добијају се вредности непознатих

$$k = \frac{width}{x_{max} - x_{min}}$$

и

$$n = \frac{-x_{min} \cdot width}{x_{max} - x_{min}}$$

тј. веза

$$x_e = \frac{(x - x_{min}) \cdot width}{x_{max} - x_{min}}.$$

До ове везе се могло доћи и на следећи начин. Количник $\frac{x - x_{min}}{x_{max} - xmin}$ одређује колико је процентуално тачка x удаљена од крајева екрана (за вредност $x = x_{min}$ добија се вредност 0, а за вредност $x = x_{max}$ добија се вредност 1). Множењем са *width* интервал $[0, 1]$ се пресликава на жељени интервал $[0, width]$.

На сличан начин се добија и веза за y-координате. Пошто су кооридинатни системи оријентисани другачије, за $y = y_{min}$ потребно је да се добије вредност *height*, а за $y = y_{max}$ потребно је да се добије вредност 0.

$$ye = \frac{(y - y_{max}) \cdot height}{y_{min} - y_{max}}.$$

Приликом записа ових формула, потребно је имениоце навести у загради. Заокруживање на најближу целобројну вредност се може извршити помоћу функције `round`. Она враћа резултат реалног типа, па приликом његовог додељивања целобројној променљивој није згорег експлицитно нагласити да желимо да се изврши конверзија реалног у целобројни тип.

```
#include <iostream>
#include <cmath>

using namespace std;

int main() {
  double xmin, xmax;
  cin >> xmin >> xmax;
  double ymin, ymax;
  cin >> ymin >> ymax;
  int width, height;
  cin >> width >> height;
  double x, y;
  cin >> x >> y;
  int xe = (int)round((x - xmin) / (xmax - xmin) * width);
  int ye = (int)round((y - ymax) / (ymin - ymax) * height);
  cout << xe << " " << ye << endl;
  return 0;
}
```

Задатак: Површина пирамиде

Написати програм који израчунава број црепова потребних за покривање крова облика правилне четворостране пирамиде.

Опис улаза

Са стандардног улаза се учитава дужина основе крова у метрима (позитиван реалан број), квадратног облика и висина пирамиде у метрима (позитиван реалан број) и број црепова потребних да се покрије један квадратни метар (позитиван цео број).

Опис излаза

На стандардни излаз исписати потребан број црепова (резултат заокружити навише).

Пример

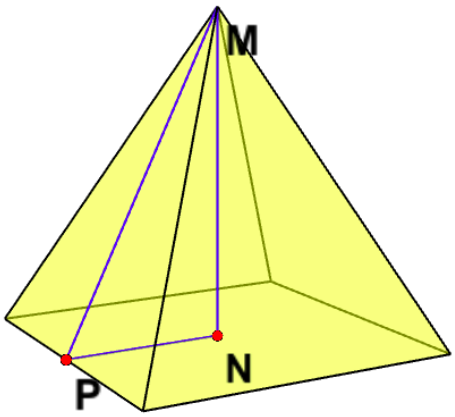
Улаз	*Излаз*
2 1.5 72	520

Решење

Површина крова је површина омотача пирамиде, који се састоји од 4 једнакокрака троугла. Површина сваког од њих се може израчунати множењем основице дужине a одговарајућом висином дужине h. Висине троуглова су бочне висине (апотеме) пирамиде. Оне се могу израчунати применом Питагорине теореме на правоугли троугао који спаја врх пирамиде, подножје висине и средиште странице квадратне основе.

Важи

$$h = \sqrt{H^2 + \left(\frac{a}{2}\right)^2}$$



Када се израчуна висина h, површина једног троугла P_t се може израчунати као $P_t = ah/2$, а површина целог омотача P као $P = 4P_t$.

Број црепова се израчунава множењем површине (у квадратним метрима) и броја црепова потребних за поплочавање једног квадратног метра. Заокруживање навише можемо извршити функцијом `ceil`. Она враћа резултат реалног типа, па га није згорег експлицитно конвертовати у целобројни тип пре доделе целобројној промењливој.

```
#include <iostream>
#include <cmath>

using namespace std;

int main() {
  // osnovica i visina piramide
  double a, H;
  cin >> a >> H;
  // broj crepova potrebnih da se pokrije 1m^2
  int crepova_po_m2;
  cin >> crepova_po_m2;

  // bocna visina
  double h = sqrt(H*H + (a/2)*(a/2));
  // povrsina trougla
  double Pt = a * h / 2;
  // povrsina piramide
```

```
  double P = 4 * Pt;
  // ukupan broj crepova (zaokruzen navise)
  int broj_crepova = (int)ceil(P * crepova_po_m2);

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

Задатак: Просечна оцена

Написати програм који на основу 5 унетих школских оцена израчунава и исписује просек оцена заокружен на две децимале.

Опис улаза

Са стандардног улаза се уноси 5 целих бројева између 1 и 5. Свака оцена је задата у посебном реду.

Опис излаза

На стандардни излаз исписати просек учитаних оцена.

Пример

Улаз	*Излаз*
4 3 4 5 3	3.80

Решење

Просечна оцена се може израчунати на основу формуле

$$o_p = \frac{o_1 + o_2 + o_3 + o_4 + o_5}{5}$$

Ако се оцене представљају целобројним типом података, треба бити обазрив приликом имплементације ове формуле. Наиме, ако се и именилац представи целим бројем `5`, тада су и дељеник и делилац целобројни и дошло би до целобројног, а не реалног дељења. Зато је потребно именилац написати у облику реалног броја `5.0`.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
  int ocena1, ocena2, ocena3, ocena4, ocena5;
  cin >> ocena1 >> ocena2 >> ocena3 >> ocena4 >> ocena5;
  double prosek = (ocena1 + ocena2 + ocena3 + ocena4 + ocena5) / 5.0;
  cout << fixed << showpoint << setprecision(2) << prosek << endl;
  return 0;
}
```

Задатак: Температура (Целзијус, Келвин, Фаренхајт)

Написати програм који на основу унете температуре у степенима Целзијуса израчунава и исписује температуру у степенима Келвина и степенима Фаренхајта.

Опис улаза

Са стандардног улаза се уноси један реалан број (већи или једнак $-273,15$) који представља темепературу у степенима Целзијуса.

Опис излаза

У први ред стандардног излаза исписати температуру у степенима Келвина, а у други у степенима Фаренхајта, заокружене на две децимале.

Пример

Улаз	*Излаз*
`37,1`	`310.25` `98.60`

Решење

Тражене температуре се могу израчунати лако на основу следећих веза:

$$\begin{aligned} K &= C + 273,15 \\ F &= C \cdot \frac{9}{5} + 32 \end{aligned}$$

Приликом уноса друге формуле у програм потребно је бити обазрив. Наиме, ако се унесе израз `9/5` вршиће се целобројно дељење (јер су и дељеник и делилац целобројни). Да би се извршило реално дељење довољно је да бар један од њих буде реалан број (а не смета да то буду и оба). Најједноставнији начин да се то постигне је да се напише `9.0` или `5.0`. Када је бројевна константа написана са децималама, подразумева се да је она реалног типа, па стога има смисла реалне констенте увек записивати са децималама (чак и када то није математички неопходно).

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
  double C;
  cin >> C;
  double K = C + 273.15;
  cout << fixed << showpoint << setprecision(2) << K << endl;
  double F =  C * (9.0 / 5.0) + 32.0;
  cout << fixed << showpoint << setprecision(2) << F << endl;
  return 0;
}
```

Задатак: F-скор

У вештачкој интелигенцији се често разматрају алгоритми класификације и потребно је оценити њихов квалитет. На пример, алгоритам који класификује слике кучића од осталих слика добија 12 слика кучића и 10 слика мачића и пријављује 8 кучића, међутим, од тих 8 њих 5 су стварно кучићи, док су њих 3 мачићи. *Прецизност* овог алгоритма је $5/8$ тј. $62,5\%$, јер се на 8 слика кучића налазило само 5 кучића. Прецизност сама за себе није довољна мера квалитета алгоритма (на пример, алгоритам који би исправно препознао само 2 слике кучића би имао прецизност 100%, али би био лош, јер би пропустио да пријави кучиће на 10 слика). *Одзив* алгоритма је $5/12$ тј. око $41,67\%$, јер је од 12 слика кучића исправно пријавио само 5. Ни одзив сам за себе није довољна мера квалитета (јер би алгоритам који би пријавио свих 12 слика кучића и још 8 слика мачића имао одзив 100%, али би му прецизност била ниска). Стога се формулише тзв. F-скор који представља хармонијску средину прецизности и одзива. Написати програм који на основу броја слика кучића, броја слика мачића, броја слика које је алгоритам класификовао као кучиће и броја слика које је алгоритам класификовао као кучиће на којима се стварно налазе кучићи одређује прецизност, одзив и F-скор тог алгоритма.

Опис улаза

Са стандардног улаза се учитавају редом:

- број слика на којима се налазе кучићи (ненегативан цео број мањи од 10^4)

- број слика на којима се налазе мачићи (ненегативан цео број мањи од 10^4)
- број слика које је алгоритам класификовао као кучиће
- број слика које је алгоритам класификовао као кучиће на којима се стварно налазе кучићи

Опис излаза

На стандардни излаз исписати редом:

- прецизност алгоритма
- одзив алгоритма
- F-скор алгоритма

Све мере треба да буду приказане у процентима, заокруженим на две децимале.

Пример

Улаз	*Излаз*
12	62.50%
10	41.67%
8	50.00%
5	

Решење

Статистике се израчунавају директном применом дефиниција прецизности, одзива и F-скора. Прецизност P је однос броја слика које су тачно класификоване као кучићи (на њима се заиста налазе кучићи) и броја слика које су класификоване као кучићи. Одзив O је однос броја слика које су класификоване као кучићи и укупног броја слика на којима се налазе кучићи. Пошто су бројеви слика цели бројеви, а прецизност и одзив реални, потребно је обезбедити да се пре дељења изврши конверзија било дељеника било делиоца (а може и оба) у реални тип, јер би дељење у супротном било целобројно и добио би се погрешан резултат. F-скор се израчунава на основу формуле

$F = \frac{2}{\frac{1}{P}+\frac{1}{O}}$.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
  int ukupno_kucica, ukupno_macica;
  cin >> ukupno_kucica >> ukupno_macica;
  int klasifikovano_kucica, tacno_klasifikovano_kucica;
  cin >> klasifikovano_kucica >> tacno_klasifikovano_kucica;
  double preciznost = (double)tacno_klasifikovano_kucica / (double)klasifikovano_kucica;
  double odziv = (double)tacno_klasifikovano_kucica / (double) ukupno_kucica;
  double fskor = 2.0 / (1.0 / preciznost + 1.0 / odziv);
  cout << fixed << showpoint << setprecision(2)
       << 100*preciznost << "%" << endl
       << 100*odziv << "%" << endl
       << 100*fskor << "%" << endl;
  return 0;
}
```

Задатак: Површина лоптастог резервоара дате запремине

Киселина се складишти у великим резервоарима облика лопте. Потребно је организовати премазивање резервоара заштитним премазом, при чему се потребна количина заштитног премаза одређује на основу површине коју треба премазати. Написати програм који на основу познате маса и густине киселине складиштене у резервоару напуњеном до врха, израчунава површину тог резервоара.

Напомена: веза између запремине, масе и густине је $m = \rho \cdot V$, запремина лопте се израчунава формулом $V = (4/3)r^3\pi$, где је r полупречник лопте, а површина формулом $P = 4r^2\pi$.

Опис улаза

Са стандардног улаза се учитава маса киселине у тонама и густина киселине у килограмима по метру кубном (у питању су позитивни реални бројеви).

Опис излаза

На стандардни излаз исписати површину резервоара у метрима квадратним.

Пример

Улаз	*Излаз*
`2.3`	`5.63`
`1830`	

Решење

На основу познате масе и густине киселине можемо одредити запремину лопте коју киселина заузима. Потребно је само бити обазрив и ускладити јединице – пошто желимо да израчунамо запремину у m^3, а густина је дата у kg/m^3, потребно је масу превести у kg тако што ћемо дату масу у t помножити са 1000. На основу познате запремине можемо одредити полупречник лопте. Наиме из $V = \frac{4}{3}r^3\pi$, важи

$$r = \sqrt[3]{\frac{3V}{4\pi}}$$

У програмском језику $C{+}{+}$ трећи корен можемо израчунати функцијом `pow`, тако што кореновање сведемо на степеновање (важи да је $\sqrt[3]{x} = x^{\frac{1}{3}}$). Морамо бити пажљиви да приликом навођења експонента $\frac{1}{3}$ употребимо реалне бројеве (када бисмо навели израз `1/3`, дошло би до целобројног, а не реалног дељења). Константу π имамо на располагању као `M_PI`, али да би она могла да се користи, на почетку програма је потребно навести `#define _USE_MATH_DEFINES`.

Када је познат полупречник лопте, површина се лако израчунава по формули $P = 4r^2\pi$. Уместо коришћења функције `pow`, квадрирање се може израчунати и свођењем на множење (важи $r^2 = r \cdot r$).

```
#include <iostream>
#include <iomanip>
#include <cmath>

#define _USE_MATH_DEFINES

using namespace std;

int main() {
  // masa (u tonama) i gustina (u kilogramima po metru kubnom)
  double m, rho;
  cin >> m >> rho;
  // zapremina (u metrima kubnim)
  double V = (m * 1000) / rho;
  // V = 4/3 r^3 pi
  double r = pow((3.0 * V) / (4.0 * M_PI), 1.0/3.0);
  // P = 4 r^2 pi
  double P = 4 * r * r * M_PI;
  cout << fixed << showpoint << setprecision(2) << P << endl;
  return 0;
}
```

Задатак: Лоптица бачена увис

Лоптица је бачена почетном брзином v_0 са балкона који се налази на висини H_0 вертикално навише и пада на земљу под дејством гравитације. Написати програм који израчунава време потребно да лоптица падне на земљу. *Напомена*: висина на којој се лоптица налази након времена t једнака је $H = H_0 + v_0 t - gt^2/2$. Програм треба да одреди позитивно време t за које је $H = 0$.

Опис улаза

Са стандардног улаза се учитава позитивни реални бројеви v_0 (брзина у метрима у секунди) и H_0 (висина балкона у метрима).

Опис излаза

На стандардни излаз исписати број секунди потребан да лопта падне на земљу, заокружен на једну децималу.

Пример

Улаз	*Излаз*
10 100	5.6

Решење

Време се одређује као позитивно решење квадратне једначине

$$0 = H_0 + v_0 t - \frac{gt^2}{2}$$

Коефицијенти ове једначине су $A = -g/2$, $B = v_0$, $C = H_0$. На основу познате формуле за решавање квадратне једначине одређујемо њено веће (једино позитивно) решење.

$$t = \frac{-B - \sqrt{B^2 - 4AC}}{2A}$$

Пошто је $A < 0$ и $C > 0$, важи да је $-4AC > 0$, па је дискриминанта увек позитивна и важи да је $\sqrt{B^2 - 4AC} > B$. Зато је $-B + \sqrt{B^2 - 4AC} > 0$, а пошто је $2A < 0$, решење у ком испред дискриминанте стоји знак $+$ је негативно. Наведено решење у ком испред дискриминанте стоји знак $-$ је позитивно, зато што је $B > 0$, па су и именилац и бројилац разломка негативни.

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

int main() {
  const double g = 9.81;
  // pocetna brzina
  double v0; cin >> v0;
  // visina balkona
  double H0; cin >> H0;

  // koeficijenti kvadratne jednacine
  // H_0 + v_0t - gt^2/2 = 0
  double A = -g / 2.0;
  double B = v0;
  double C = H0;
  // vreme odredjujemo resavanje kvadratne jednacine
  double t = (-B - sqrt(B*B - 4*A*C)) / (2 * A);
  cout << fixed << showpoint << setprecision(1) << t << endl;
  return 0;
}
```

Задатак: Коси хитац

Коси хитац испаљен почетном брзином v_0 под углом θ путује време $t = \frac{2v_0 \sin\theta}{g}$. При том достиже максималну висину $H_{max} = \frac{v_0^2 \sin^2\theta}{2g}$ и има домет $R = \frac{v_0^2 \sin(2\theta)}{g}$. Написати програм који за унету почетну брзину v_0 (у метрима у секунди) и угао θ (у степенима) израчунава време, максималну висину и домет косог хица.

Опис улаза

Са стандардног улаза се учитавају два реална броја $v_0 > 0$ и $\theta \in (0, 180)$.

Опис излаза

На стандардни излаз исписати вредности t, H_{max} и R, заокружене на две децимале (t је изражено у секундама, а H_{max} и R у метрима).

Пример

Улаз	*Излаз*
`10`	`0.94`
`27.42`	`1.08`
	`8.33`

Решење

Програм захтева да се исправно запишу све наведене формуле. За рачунање синуса угла можемо користити функцију `sin` из заглавља `<cmath>`. Да би се синус исправно израчунао, потребно је угао превести из степена у радијане (дељењем са 180 и множењем са π). Приликом записа формула за време и домет, нису неопходне заграде. Са друге стране, у формули за H_{max} је неопходно да се именилац $2g$ наведе у загради. Треба обратити пажњу и на то да запис $\sin^2 x$ уобичајен у математици заправо представља $(\sin(x))^2$.

```
#include <iostream>
#include <iomanip>
#define _USE_MATH_DEFINES
#include <cmath>

using namespace std;

int main() {
  // ucitavamo pocetnu brzinu i ugao u stepenima
  const double g = 9.81;
  double v0, theta_stepeni;
  cin >> v0 >> theta_stepeni;
  // izrazavamo ugao u radijanima
  double theta = theta_stepeni / 180 * M_PI;

  // vreme
  double t = 2 * v0 * sin(theta) / g;
  // maksimalna visina
  double Hmax = (v0*v0 * sin(theta)*sin(theta)) / (2*g);
  // domet
  double R = v0*v0*sin(2*theta) / g;

  cout << fixed << showpoint << setprecision(2)
       << t << endl
       << Hmax << endl
       << R << endl;
  return 0;
}
```

Задатак: Менхетн растојање

Мехнетн (део Њујорка) је познат по својој правоугаоној мрежи улица и авенија. Написати програм који одређује најкраћи пут који такси треба да пређе од једне до друге раскрснице на Менхетну. Претпоставити да је размак између било две улице 80 метара, а између било које две авеније 274 метра.

Опис улаза

Са стандардног улаза се уносе координате прве раскрснице (број авеније и број улице раздвојени размаком), а затим и координате друге раскрснице. Авеније су означене бројевима од 1 до 9, а улице бројевима од 20 до 120.

Опис излаза

На стандардни излаз исписати растојање у метрима које такси треба да пређе.

Пример

Улаз	*Излаз*
7 48	2936
3 25	

Решење

Пошто је мрежа улица и авенија правоугаона, растојање је исто без обзира на то када таксиста скреће, под претпоставком да се у сваком тренутку и у хоризонталном и у вертикалном смеру креће ка одредишту (а не од одредишта). Стога је довољно да израчунамо дужину пута у ком се таксиста креће прво улицом у којој се на почетку налази ка авенији у коју треба да стигне, па затим том авенијом до улице до које треба да стигне. Ако се на почетку налази у авенији a_1, а треба да дође до авеније a_2, прећи ће пут $|a_1 - a_2| \cdot D_a$, где је D_a растојање између две суседне авеније. Слично, ако се на почетку налази у улици u_1, а треба да дође до улице u_2, прећи ће пут $|u_1 - u_2| \cdot D_u$, где је D_u растојање између две суседне улице. Стога је тражено растојање једнако

$$|a_1 - a_2| \cdot D_a + |u_1 - u_2| \cdot D_u.$$

```
#include <iostream>

using namespace std;

int main() {
  const int D_U = 80, D_A = 274;
  int a1, u1, a2, u2;
  cin >> a1 >> u1 >> a2 >> u2;
  int d = abs(a1 - a2) * D_A + abs(u1 - u2) * D_U;
  cout << d << endl;
  return 0;
}
```

1.2 Целобројна аритметика

Задаци у овом поглављу интензивно користе особине целобројног дељења: израчунавање целобројног количника и остатка. Дељењем целих бројева a и b ($b \neq 0$) добијају се количник q и остатак r ако и само ако је $a = qb + r$ и $0 \leq r < b$. У програмском језику C++, целобројно дељење се врши када год су оба операнда оператора `/` цели бројеви. Остатак при дељењу се израчунава оператором `%`.

Задатак: Подела производа

Укупно n производа треба поделити у k једнаких група. Написати програм који одређује колико ће производа бити у свакој групи и колико ће производа бити нераспоређено.

Опис улаза

Са стандардног улаза се учитава природан број n ($1 \leq n \leq 10^6$), а затим и природан број k ($1 \leq k \leq 10^6$).

Опис излаза

На стандардни излаз прво исписати број производа у свакој групи, а затим и број нераспоређених производа.

Пример

Улаз	*Излаз*
100	14
7	2

Објашњење

Формирањем 7 група од по 14 производа распоређује се 98 производа, а 2 производа остаје нераспоређено.

Решење

Број производа у свакој групи се добија целобројним дељењем бројева n и k (једнак је $\lfloor n/k \rfloor$), док је број нераспоређених производа остатак при дељењу бројева n и k.

```
#include <iostream>

using namespace std;

int main() {
  int n, k;
  cin >> n >> k;
  cout << n / k << endl;
  cout << n % k << endl;
  return 0;
}
```

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 $\frac{c}{b} < 1$.

Опис улаза

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 ($a \geq b$).

Опис излаза

Прва и једина линија стандардног излаза треба да садржи мешовити запис разломка, прецизније природан број, бројилац и именилац мешовитог броја међусобно раздвојени по једном празнином (бланко знаком).

Пример

Улаз	*Излаз*
23 8	2 7 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 бројиоца 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.

```
#include <iostream>

using namespace std;

int main() {
  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;
  return 0;
}
```

Задатак: Фестивал

Спортисти на дресовима имају бројеве 1, 2, 3 итд. У свечаном дефилеу корачају у m колона. Написати програм који одређује у којој ће се врсти и у колони налазити спортиста са редним бројем n (врсте и колоне се броје од 1). На пример, ако је $m = 15$, спортисти су распоређени на следећи начин.

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15   врста 1
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30   врста 2
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45   врста 3
46 47 48 49 ...                                врста 4
```

Опис улаза

Са стандардног улаза се учитава број m ($1 \leq m \leq 20$) и n ($1 \leq n \leq 1000$).

Опис излаза

На стандардни излаз исписати редни број врсте и колоне спортисте који носи број n, раздвојене размаком.

Пример 1

Улаз	*Излаз*
15	1 11
11	

Пример 2

Улаз	*Излаз*
15	2 1
16	

Пример 3

Улаз	*Излаз*
15	7 13
103	

Решење

Када би се бројање вршило од нуле, довољно би било пронаћи целобројни количник и остатак броја n при дељењу са k. Пошто се бројање врши од 1, можемо да одузмемо 1 од броја n, затим да одредимо врсту и колону које се броје од 0 и на те бројеве да додамо 1 (да бисмо прешли на бројање од 1).

```cpp
#include <iostream>

using namespace std;

int main() {
  int m, n;
  cin >> m >> n;
  int v = (n - 1) / m + 1;
  int k = (n - 1) % m + 1;
  cout << v << " " << k << endl;
  return 0;
}
```

Задатак: Дан у месецу

Ако се зна који је дан у недељи био први дан текућег месеца, напиши програм који одређује који дан је данас.

Опис улаза

Прва линија стандардног улаза садржи ознаку дана у недељи првог дана текућег месеца (1 - понедељак, 2 - уторак, 3 - среда, 4 - четвртак, 5 - петак, 6 - субота, 7 - недеља), а друга линија садржи редни број текућег дана у текућем месецу.

Опис излаза

На стандардни излаз исписати ознаку данашњег дана.

Пример 1

Улаз	*Излаз*
1	4
4	

Објашњење

Први дан је био понедељак, па је четврти дан четвртак.

Пример 2

Улаз

```
1
8
```

Излаз

```
1
```

Објашњење

Први дан је био понедељак, па је и осми дан понедељак.

Пример 3

Улаз

```
3
17
```

Излаз

```
5
```

Објашњење

Први дан је била среда, па је седамнаести дан петак.

Решење

На ознаку првог дана у месецу додаћемо колико је дана прошло до данашњег. Тај број протеклих дана се добија тако што се од редног броја данашњег дана одузме 1. Пошто у обзир треба узети и периодично понављање дана, треба пронаћи остатак при дељењу са 7. Пошто се дани броје од 1 до 7, а остаци при дељењу са 7 су од 0 до 6, примењујемо "трик" да пре израчунавања остатка одузмемо 1, а након израчунавања додамо 1.

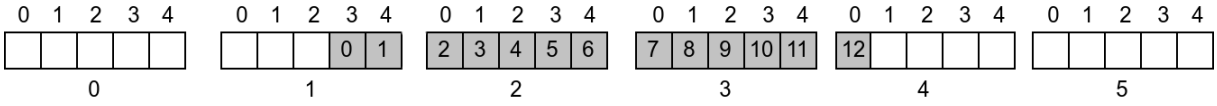
```
#include <iostream>

using namespace std;

int main() {
  int prvi;
  cin >> prvi;
  int danas;
  cin >> danas;
  cout << (prvi + (danas - 1) - 1) % 7 + 1 << endl;
  return 0;
}
```

Задатак: Програмери у возу

Програмери из једне компаније су кренули возом у посету сајму. Купили су карте тако да седе на узастопним седиштима. Пошто их је пуно, седеће у неколико вагона, као што је приказано на слици.



Написати програм који на основу броја седишта у вагонима, броја вагона у коме седи први програмер и броја седишта на ком он седи, одређује број вагона и број седишта на ком седи n-ти програмер по реду (вагони, седишта и програмери се броје од 0).

У примеру на слици први програмер седи у вагону број 1, на седишту број 3, па се зато, на пример, програмер са бројем 9 налази у вагону број 3 на седишту број 2.

Опис улаза

Са стандардног улаза се уносе редом:

- број k (позитиван цео број који одређује број седишта у сваком вагону),
- број v_0 (ненегативан цео број који одређује број вагона у ком седи почетни програмер),
- број i ($0 \leq i < k$, ненегативан цео број који одређује број седишта на ком седи први програмер),

- број n (ненегативан цео број који одређује редни број програмера чији положај желимо да одредимо).

Опис излаза

На стандардни излаз исписати број b вагона у ком се налази тражени програмер и редни број j ($0 \leq j < k$) седишта на ком он седи.

Пример 1

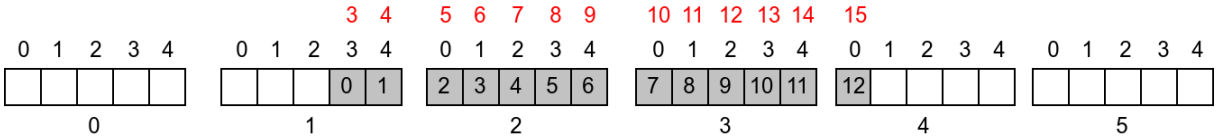
Улаз	*Излаз*
5	3
1	2
3	
9	

Пример 2

Улаз	*Излаз*
5	1
1	4
3	
1	

Решење

Почетни програмер седи у вагону v_0 на седишту s_0. Претпоставимо за почетак да су седишта нумерисана редом од седишта s_0 и да се бројање седишта наставља и након преласка у наредни вагон, као што је приказано на наредној слици:



Тада би програмер са редним бројем n седео на седишту број $s = s_0 + n$. Лако се примећује да у таквом бројању почетна седишта у сваком вагону добијају бројеве дељиве са k, наредна седишта добијају бројеве који дају остатак 1 при дељењу са k итд. тј. да се стварни број седишта може лако добити израчунавањем остатка при дељењу са k. Слично, редни број вагона се може добити целобројним дељењем са k, при чему то нису стварни бројеви вагона већ померај у односу на вагон v_0 (за седишта у вагону v_0 добиће се количник 0, за седишта у наредном вагону количник 1 итд.). Зато се тражени број седишта може израчунати као $s_n = (s_0 + n) \bmod k$, а тражени број вагона се може израчунати као $v_n = v_0 + \lfloor (s_0 + n)/k \rfloor$.

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

int main() {
    // broj sedista u vagonu
    int k;
    cin >> k;

    // polozaj pocetnog programera
    int vagon0, sediste0;
    cin >> vagon0 >> sediste0;

    // redni broj programera koji nas zanima
    int n;
    cin >> n;

    // izračunavanje položaja programera sa rednim brojem n
    int sediste = sediste0 + n;
    int vagonn = vagon0 + sediste / k;
    int sedisten = sediste % k;

    cout << vagonn << endl;
    cout << sedisten << endl;

    return 0;
}
```

Задатак: Центар правоугаоника

Правоугаоници се у рачунарској графици задају или тако што се задају координате центра, ширина и висина или тако што се задају координате горњег левог и доњег десно темена. Написати програм који на основу једне репрезентације одређује другу.

Опис улаза

Са стандардног улаза се учитавају координате центра једног правоугаоника (цели бројеви), његова ширина и висина (позитивни цели бројеви).

Затим се учитавају координате горњег левог и координате доњег десног темена другог правоугаоника.

Опис излаза

На стандардни излаз исписати координате горњег левог и доњег десног темена првог правоугаоника. Затим исписати координате центра, ширину и висину другог правоугаоника.

Напомена: ако су ширина или висина парни бројеви, координате центра упадају између два пиксела. У том случају центром сматрамо онај од њих који има мању координату.

Пример

Улаз	*Излаз*
`100 100 10 15`	`96 93 105 107`
`100 45 200 100`	`150 72 101 56`

Решење

Размотримо прво одређивање темена на основу познатих координата центра. Да су координате реалне, задатак би био једноставнији. Координате горњег левог темена (x_1, y_1) би се могле лако одредити као $x_1 = c_x - w/2$, $y_1 = c_y - w/2$, а доњег десног темена (x_2, y_2) као $x_2 = c_x + h/2$, $y_2 = c_y + h/2$. Међутим, пошто су координате целобројне, треба бити мало пажљивији. Ако је ширина непарна, тада су и лево и десно теме удаљена за $(w-1)/2$ од средишњег пиксела. Ако је ширина парна, онда је лево теме удаљено за $w/2-1$ пиксел, а десно за $w/2$ пиксела. Може се приметити да је у случају непарног w вредност $\lfloor (w-1)/2 \rfloor$ једнака $(w-1)/2$, а да је у случају парног w та вредност једнака $w/2-1$. Слично, вредност $\lfloor w/2 \rfloor$ је у случају непарног w једнака $(w-1)/2$, а у случају парног w једнака је $w/2$. Дакле, без обзира на парност броја w, x-координата левог темена се може израчунати као $x_1 = c_x + \lfloor (w-1)/2 \rfloor$, а десног темена као $x_2 = c_y + \lfloor w/2 \rfloor$. Координате y темена се израчунавају потпуно аналогно.

Одређивање центра, ширине и висине на основу координата темена је једноставније. Ширина је очигледно једнака $w = x_2 - x_1 + 1$, а висина $h = y_2 - y_1 + 1$. Ако би координате биле реални бројеви, тада би се координате центра могле израчунати као аритметичка средина координата темена $c_x = (x_1 + x_2)/2$, $c_y = (y_1 + y_2)/2$. Сличне формуле се користе и у случају целобројних координата, једино је (по условима задатка) потребно употребити заокруживање наниже. $c_x = \lfloor (x_1 + x_2)/2 \rfloor$, $c_y = \lfloor (y_1 + y_2)/2 \rfloor$.

Заокруживање количника наниже се, наравно, у случају када се ради са целим бројевима врши применом целобројног дељења.

```
#include <iostream>

using namespace std;

void centarUTemena() {
  int cx, cy, w, h;
  cin >> cx >> cy >> w >> h;
  int x1 = cx - (w-1) / 2;
  int y1 = cy - (h-1) / 2;
  int x2 = cx + w / 2;
  int y2 = cy + h / 2;
  cout << x1 << " " << y1 << " " << x2 << " " << y2 << endl;
}

void temenaUCentar() {
  int x1, y1, x2, y2;
  cin >> x1 >> y1 >> x2 >> y2;
```

```
  int cx = (x1+x2) / 2;
  int cy = (y1+y2) / 2;
  int w = x2 - x1 + 1;
  int h = y2 - y1 + 1;
  cout << cx << " " << cy << " " << w << " " << h << endl;
}

int main() {
  centarUTemena();
  temenaUCentar();
  return 0;
}
```

1.3 Позициони запис

Задатак: ПИН апликације од ПИН-а телефона

Студент треба да одреди четвороцифрени ПИН за нову апликацију, али није сигуран да ће га добро запамтити. Зато је смислио да тај ПИН израчуна на основу њему добро познатог ПИН-а телефона. Сабраће ПИН свог телефона, са бројем који добије када том ПИН-у замени прву и последњу цифру и са бројем који добије када том ПИН-у замени две средишње цифре. Нови ПИН ће бити последње 4 цифре тако добијеног збира. На пример, ако је ПИН телефона 1234, сабраће бројеве 1234 + 4231 + 1324 и добиће збир 6789, који је уједно нови ПИН.

Опис улаза

Са стандардног улаза се уноси четвороцифрени ПИН телефона (може да има и водеће нуле).

Опис излаза

На стандардни излаз исписати четвороцифрени ПИН апликације (укључујући и водеће нуле ако их има).

Напомена: приказ водећих нула у броју `p` се у језику C++ може постићи наредбом

```
cout << setw(4) << setfill('0') << p << endl;
```

Пример 1

Улаз	*Излаз*
1234	6789

Пример 2

Улаз	*Излаз*
9999	9997

Објашњење

Ако је ПИН телефона 9999, тада ће добити збир 9999 + 9999 + 9999 = 29997, па је нови ПИН 9997. Написати програм који на основу унетог четвороцифреног ПИН-а телефона одређује нови четвороцифрени ПИН који ће се користити за апликацију.

Решење

Опис главног решења

У овом блоку се описује главно решење задатка.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
  int pinTelefona;
  cin >> pinTelefona;

  // odredjujemo cifre PIN-a telefona
```

```
  int c0 = pinTelefona % 10;
  int c1 = (pinTelefona / 10) % 10;
  int c2 = (pinTelefona / 100) % 10;
  int c3 = (pinTelefona / 1000) % 10;

  // broj dobijen zamenom cifre jedinica i hiljada
  int zamenaSpoljasnjih = 1000*c0 + 100*c2 + 10*c1 + c3;
  // broj dobijen zamenom cifre desetica i stotica
  int zamenaUnutrasnjih = 1000*c3 + 100*c1 + 10*c2 + c0;

  // PIN aplikacije su poslednje 4 cifre zbira
  int pinAplikacije = (pinTelefona + zamenaSpoljasnjih + zamenaUnutrasnjih) % 10000;

  // ispisujemo cetvorocifreni PIN aplikacije sa eventualnim vodecim nulama
  cout << setw(4) << setfill('0') << pinAplikacije << endl;

  return 0;
}
```

Задатак: Контролна цифра

Идентификациони бројеви (нпр. ISBN, JMBG, бројеви банковних рачуна и кредитних картица) се обично штите од грешака увођењем тзв. контролне цифре. Последња цифра броја се одређује тако да се нека израчуната статистика свих цифара броја буде дељива неким бројем. На пример, четвороцифрени број $ABCD$ можемо заштитити контролном цифром X тако што ћемо X одредити тако да у добијеном броју $ABCDX$ важи да је $A + 2B + 3C + 4D + X$ дељиво бројем 10. Написати програм који за унети четвороцифрени број $ABCD$ одређује најмању вредност конторлне цифре X.

Опис улаза

Са стандардног улаза се учитава четвороцифрени број $ABCD$ (који евентуално може да има и водеће нуле).

Опис излаза

На стандардни излаз исписати контролну цифру X.

Пример 1

Улаз *Излаз*

```
1234      0
```

Објашњење

Важи да је $1 + 2 \cdot 2 + 3 \cdot 3 + 4 \cdot 4 = 30$, који је већ дељив са 10. Зато је најмањи број X који се може додати на овај број да би он постао дељив са 10 број $X = 0$.

Пример 2

Улаз

```
8352
```

Излаз

```
3
```

Објашњење

Важи да је $8 + 2 \cdot 3 + 3 \cdot 5 + 4 \cdot 2 = 37$. Најмањи број X који се може додати на овај број да би он постао дељив са 10 је 3.

Решење

Учитавамо број и одређујемо му појединачне цифре A, B, C и D. Након тога можемо израчунати вредност израза $A + 2B + 3C + 4D$. На њега треба додати неку вредност X тако да збир буде дељив са 10 тј. да му је последња цифра нула. Ако је последња цифра збира $A + 2B + 3C + 4D$ нека цифра k, тада се може додати број $X = 10 - k$. Ово је и коначно решење у свим случајевима осим када је $k = 0$, јер је тада уместо

$X = 10$ могуће додати $X = 0$. Зато је пре исписа коначног резултата потребно још одредити остатак при дељењу X са 10 (или гранањем обрадити овај специјални случај).

```
#include <iostream>

using namespace std;

int main() {
  int broj;
  cin >> broj;
  // odredjujemo pojedinacne cifre broja
  int A = (broj / 1000) % 10;
  int B = (broj / 100) % 10;
  int C = (broj / 10) % 10;
  int D = (broj / 1) % 10;
  // odredjujemo kontrolnu cifru - najmanju cifru X tako da
  // A+2B+3C+4D+X bude deljivo sa 10
  int X = (10 - (A+2*B+3*C+4*D) % 10) % 10;
  cout << X << endl;
  return 0;
}
```

Задатак: Трећи угао троугла

Написати програм који на основу величине два унета угла троугла (задата у степенима, минутама и секундама) израчунава величину трећег угла.

Опис улаза

Са стандрадног улаза се учитава број степени, минута и секунди првог угла. За сваки угао је дат број степени (ненегативан цео број између 0 и 179), број минута (ненегативан цео број између 0 и 59) и број секунди (ненегативан цео број између 0 и 59). Сваки угао је већи од 0 степени. Збир унетих углова је строго мањи од 180 степени.

Опис излаза

На стандардни излаз исписати величину трећег угла у степенима (између 0 и 179), минутама (између 0 и 59) и секундама (између 0 и 59).

Пример

Улаз	*Излаз*
43 17 23 56 19 55	80 22 42

Решење

Збир углова у троуглу је 180 степени. Зато се трећи угао израчунава тако што се од 180 степени одузму први и други угао. Аритметика са угловима се најлакше изводи ако се сви углови представе само секундама. Превођење угла датог у степенима, минутама и секундама само у секунде и обратно се своди на операције над бројевима записаним у основи 60.

Ако знамо број степени *st*, мнута *min* и секунди *sek*, број секунди можемо израчунати коришћењем чињенице да у једном степену има 60 минута, а у једном минуту има 60 секунди. Број секунди је зато једнак $st \cdot 60^2 + min \cdot 60 + sek$. Алтернативно, можемо се прво ослободити степени и израчунати укупан број минута као $st \cdot 60 + min$, а затим све превести у секунде множењем претходно добијеног броја минута са 60 и додајући број секунди. Ово одговара Хорнеровој схеми $(st \cdot 60 + min) \cdot 60 + sek$.

Обратно, целобројним дељењем са 60, можемо добити угао представљен помоћу целог броја минута (то је целобројни количник) и преосталог броја секунди (то је целобројни остатак). Даљим целобројним дељењем овог целог броја минута се може добити број степени и преостали број минута. Степени се, дакле, добијају целобројним дељењем угла у секундама са 60^2, минути као остатак при дељењу са 60 количника са 60, а секунде као остатак при дељењу са 60.

```cpp
#include <iostream>

using namespace std;

int main() {
  int stepeni1, minuti1, sekunde1;
  cin >> stepeni1 >> minuti1 >> sekunde1;
  int ugao1 = ((stepeni1 * 60) + minuti1) * 60 + sekunde1;

  int stepeni2, minuti2, sekunde2;
  cin >> stepeni2 >> minuti2 >> sekunde2;
  int ugao2 = ((stepeni2 * 60) + minuti2) * 60 + sekunde2;

  int ugao3 = 180*60*60 - ugao1 - ugao2;
  int sekunde3 = ugao3 % 60;
  int minuti3 = (ugao3 / 60) % 60;
  int stepeni3 = ugao3 / (60 * 60);

  cout << stepeni3 << " " << minuti3 << " " << sekunde3 << endl;
  return 0;
}
```

Види другачија решења овог задатка.

Задатак: Трајање рачунарског процеса

Познато је време почетка извршавања рачунарског процеса (дато у сатима, минутама, секундама и милисекундама) и његово трајање у милисекундама. Одредити време завршетка изршавања тог процеса.

Опис улаза

Са стандардног улаза се учитава време почетка у облику 4 броја раздвојена размаком који представљају:

- сат (цео број између 0 и 23),
- минут (цео број између 0 и 59),
- секунд (цео број између 0 и 59) и
- милисекунд (цео број између 0 и 999).

Из наредног реда се учитава тајање процеса у милисекундама (ненегативан цео број између 0 и 10^6).

Опис излаза

На стандардни излаз исписати време завршетка у истом облику у ком је задато време почетка.

Пример

Улаз	*Излаз*
`15 38 17 125` `130435`	`15 40 27 560`

Решење

Задатак се најлакше решава ако се време почетка претвори у милисекунде (од протекле поноћи), затим се израчуна време завршетка сабирајући време почетка и трајање (оба у милисекундама) и на крају се то време претвори у сате, минуте, секунде и милисекунде. Користе се уобичајене технике конверзије позиционог записа (у питању је мешовита основа $24, 60, 60, 1000$).

```cpp
#include <iostream>

using namespace std;

int main() {
  // ucitavamo vreme pocetka
  int sat, minut, sekund, milisekund;
  cin >> sat >> minut >> sekund >> milisekund;
```

```
  // pretvaramo vreme pocetka u milisekunde
  int pocetak = ((sat*60 + minut)*60 + sekund)*1000 + milisekund;
  // ucitavamo trajanje
  int trajanje;
  cin >> trajanje;
  // izracunavamo vreme zarvrsetka u milisekundama
  int kraj = pocetak + trajanje;
  // pretvaramo vreme zavrsetka u sate, minute, sekunde i milisekunde
  milisekund = kraj % 1000;
  sekund = (kraj / 1000) % 60;
  minut = (kraj / (1000 * 60)) % 60;
  sat = (kraj / (1000 * 60 * 60)) % 24;
  // ispisujemo vreme zavrsetka
  cout << sat << " " << minut << " " << sekund << " " << milisekund << endl;
  return 0;
}
```

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.

Опис улаза

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.

Опис излаза

На стандрадни излаз се исписује један ред у коме су три броја (сат, минут и секунд) раздвојена двотачкама.

Пример 1

Улаз	*Излаз*
23 59 59 0 0 1	0:0:2

Пример 2

Улаз	*Излаз*
0 0 1 23 59 59	23:59:58

Решење

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). Када су оба времена изражена само у секундама, потребно је пронаћи разлику између њих по модулу броја секунди у једном дану. На крају, добијену разлику у секундама потребно је превести у време у сатима, минутима и секундама. Илустрације ради, претварање из сати, минута и секунди у секунде и обратно можемо издвојити у посебне функције (које се онда могу користити у већем броју програма).

Потребно је да се израчуна разлика времена завршетка и времена почетка путовања по модулу броја секунди у дану. Разлику бројева a и b по модулу n могуће је израчунати као $(a - b + n) \bmod n$.

```
#include <iostream>

using namespace std;

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;
```

```
  m = (S / 60) % 60;
  h = (S / (60*60)) % 24;
}

int main() {
  int hPocetak, mPocetak, sPocetak;
  cin >> hPocetak >> mPocetak >> sPocetak;
  int hKraj, mKraj, sKraj;
  cin >> hKraj >> mKraj >> sKraj;

  int SPocetak = uSekunde(hPocetak, mPocetak, sPocetak);
  int SKraj    = uSekunde(hKraj, mKraj, sKraj);

  const int brojSekundiUDanu = uSekunde(24, 0, 0);

  int hTrajanje, mTrajanje, sTrajanje;
  odSekundi((SKraj - SPocetak + brojSekundiUDanu) % brojSekundiUDanu,
            hTrajanje, mTrajanje, sTrajanje);

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

  return 0;
}
```

Веома једноставан начин да се израчуна разлика по модулу (који додуше подразумева коришћење гранања) је да се испита да ли је време доласка мање од времена поласка и ако јесте, да се пре одузимања увећа за 24 часа (изражена бројем секунди).

```
int SPocetak = uSekunde(hPocetak, mPocetak, sPocetak);
int SKraj    = uSekunde(hKraj, mKraj, sKraj);

if (SPocetak > SKraj)
  SKraj += uSekunde(24, 0, 0);

int hTrajanje, mTrajanje, sTrajanje;
odSekundi(SKraj - SPocetak, hTrajanje, mTrajanje, sTrajanje);
```

Задатак: Јарди и метри

У САД се за мерење дужине користе миље, ланци, јарди, стопе и инчи. Једна стопа има 12 инча, један јард има 3 стопе, један ланац има 22 јарда, а једна миља има 80 ланаца. Један инч износи $2,54\,\text{cm}$. Написати програм који врши прерачунавање између ових и метричких јединица.

Опис улаза

У првом реду стандардног улаза се наводи дужина изражена у броју миља, ланаца, јарди, стопа и инча (5 ненегативних целих бројева раздвојених са по једним размаком).

У другом реду се наводи дужина изражена у броју километара, метара, центиметара и милиметара (четири ненегативна цела броја раздвојена са по једним размаком).

Опис излаза

На стандардни излаз исписати прву дужину изражену у метричким јединицама и другу дужину изражену у империјалним јединицама. Вршити заокруживање на најближи цео број инча тј. милиметара.

Пример

Улаз	*Излаз*
`1 35 17 2 9`	`2 329 81 5`
`2 520 37 2`	`1 45 6 0 11`

Решење

Прво је потребно да конвертујемо дужину из империјалних у метричке јединице. Први корак ће бити да дужину изражену у миљама, ланцима, јардима, стопама и инчима изразимо само у инчима. Најједноставнији начин да се то уради је Хорнерова схема.

- Прво желимо да се ослободимо миља, тако што ћемо дужину у миљама претворити у ланце. Зато број миља множимо бројем 80 (јер једна миља има 80 ланаца) и на то додајемо дати број ланаца.
- Ланаца се ослобађамо тако што тако добијени број ланаца множимо са 22 (јер један ланац има 22 јарда) и на то додајемо дати број јарди.
- Јарди се ослобађамо тако што тако добијени број јарди множимо са 3 (јер један јард има 3 стопе) и на то додајемо дати број стопа.
- На крају, стопа се ослобађамо тако добијени број стопа множимо са 12 (јер једна стопа има 12 инча) и на то додајемо дати број инча.

Тако добијену дужину у инчима можемо изразити у милиметрима множењем са $25,4$ (јер један инч има $25,4$ милиметра) и заокруживањем на најближи цео број.

Сада је потребно дужину дату у милиметрима претворити у километре, метре, центиметре и милиметре.

- Пошто један центиметар има 10 милиметара, дужину изражену у целом броју центиметара можемо добити целобројним дељењем дужине дате у милиметрима са 10, а преостали број милиметара ће бити остатак при дељењу са 10.
- Пошто у једном метру има 100 центиметара, дужину изражену у метрима можемо добити целобројним дељењем дужине изражене у целом броју центиметара, а преостали број центиметара можемо добити израчунавањем остатка при том дељењу. Дакле, дужина изражена у целом броју метара се добија целобројним дељењем дужине у милиметрима са $10 \cdot 100$ (што је број милиметара у метру).
- Пошто у једном километру има 1000 метара, дужину изражену у целом броју километара можемо добити целобројним дељењем дужине изражене у метрима са 1000, а преостали број метара можемо добити као остатак у том дељењу. Дакле, број километара се добија целобројним дељењем дужине у милиметрима са $10 \cdot 100 \cdot 1000$ (што је број милиметара у километрима).

Превођење метричких у империјалне јединице врши се аналогно.

```
#include <iostream>
#include <cmath>

using namespace std;

int main() {
  int milja, lanaca, jardi, stopa, inca;
  cin >> milja >> lanaca >> jardi >> stopa >> inca;
  int duzina1_in =
    ((((milja * 80) + lanaca) * 22 + jardi) * 3 + stopa) * 12 + inca;
  int duzina1_mm = (int)(round(duzina1_in * 25.4));

  int mm = duzina1_mm % 10;
  int cm = (duzina1_mm / 10) % 100;
  int m  = (duzina1_mm / (10*100)) % 1000;
  int km = duzina1_mm / (10*100*1000);

  cout << km << " " << m << " " << cm << " " << mm << endl;

  cin >> km >> m >> cm >> mm ;
  int duzina2_mm =
    (((km * 1000) + m) * 100 + cm) * 10 + mm;
  int duzina2_in = (int)round(duzina2_mm / 25.4);
  inca   = duzina2_in % 12;
  stopa  = (duzina2_in / 12) % 3;
  jardi  = (duzina2_in / (12 * 3)) % 22;
  lanaca = (duzina2_in / (12 * 3 * 22)) % 80;
  milja  = duzina2_in / (12 * 3 * 22 * 80);
```

```
  cout << milja << " " << lanaca << " " << jardi << " " << stopa << " " << inca << endl;

  return 0;
}
```

Елегантније решење се може добити ако се користе помоћне променљиве.

```
#include <iostream>
#include <cmath>

using namespace std;

int main() {
  int milja, lanaca, jardi, stopa, inca;
  cin >> milja >> lanaca >> jardi >> stopa >> inca;
  int duzina1_in = ((((milja * 80) + lanaca) * 22 + jardi) * 3 + stopa) * 12 + inca;
  int duzina1_mm = (int)(round(duzina1_in * 25.4));

  int mm = duzina1_mm % 10;  int duzina1_cm = duzina1_mm / 10;
  int cm = duzina1_cm % 100; int duzina1_m  = duzina1_cm / 100;
  int m =  duzina1_m % 1000; int duzina1_km = duzina1_m  / 1000;
  int km = duzina1_km;

  cout << km << " " << m << " " << cm << " " << mm << endl;

  cin >> km >> m >> cm >> mm ;
  int duzina2_mm = (((km * 1000) + m)*100 + cm)*10 + mm;
  int duzina2_in = (int)round(duzina2_mm / 25.4);

  inca   = duzina2_in % 12; int duzina2_st = duzina2_in / 12;
  stopa  = duzina2_st % 3;  int duzina2_jd = duzina2_st / 3;
  jardi  = duzina2_jd % 22; int duzina2_ln = duzina2_jd / 22;
  lanaca = duzina2_ln % 80; int duzina2_ml = duzina2_ln / 80;
  milja  = duzina2_ml;

  cout << milja << " " << lanaca << " " << jardi << " " << stopa << " " << inca << endl;

  return 0;
}
```

Глава 2

Гранање

Гранање вршимо на основу логичких услова који су типа `bool` и могу имати вредност `true` или `false`. Елементарни логички услови се граде применом релацијских оператора на изразе састављене од променљивих и константи. Релацијски оператори су `<` (мање), `<=` (мање једнако), `>` (веће), `>=` (веће једнако), `==` (једнако) и `!=` (различито). Могу се примењивати на бројевне типове (`int`, `double`, …), на карактере и ниске карактера (тип `string`). Логички услови се комбинују применом логичких оператора `&&` (конјункција), `||` (дисјункција), `!` (негација).

Основна наредба гранања је наредба `if`.

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

Наредба може бити или појединачна наредба или блок наредби наведених између витичастих заграда (око једне наредбе није обавезно навести витичасте заграде). Прва наредба се извршава ако је услов испуњен, а друга ако услов није испуњен. Део `else` се може изоставити.

Наредбе `if` се могу угнежђавати (у телу једне наредбе `if` налази се друга наредба `if`). Чест облик угнежђавања је и тзв. конструкција `else-if`.

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

Уместо наредбе гранања некада је погодније искористити условни израз (израз гранања) који се формира употребом оператора `?:` и 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.

Гранање на основу дискретног скупа вредности се може остварити и наредбом `switch-case`, чији је општи облик:

```
switch (izraz) {
  case vrednost1:
     naredbe1
     break;
```

```
  case vrednost2:
     naredbe2
     break;
  ...
  default:
     naredben
     break;
}
```

Израчунава се вредност наведеног израза и затим се тражи ознака `case` у коме је наведена баш та вредност и извршавају се наредбе кренувши од те ознаке, све до наредбе `break` (или до краја наредбе `switch`, ако се не наведе `break`). Ако таква ознака не постоји, извршавају се наредбе наведене иза ознаке `default`.

У наставку ћемо приказати неколико задатака који илуструју употребу гранања приликом решавања проблема. Груписаћемо их по неким типичним начинима на које се ова наредба употребљава. Наглашавамо да ови сценарији употребе нису исцрпни и да програмер увек има слободу да наредбе организује на начин који доводи до решења проблема.

2.1 Једноставно гранање

У наставку ћемо показати неколико веома једноставних задатака у којима се илуструје употреба појединачних наредби гранања, са простим условим добијеним применом релацијских оператора.

Задатак: Корен

Написати програм који исписује корен унетог ненегативног реалног броја заокружен на три децимале. Ако је унос неисправан (ако се унесе негативан број), пријавити грешку.

Опис улаза

Са стандардног улаза се уноси реалан број (он може бити позитиван, негативан или нула).

Опис излаза

Ако је унети број ненегативан, на стандардни излаз исписати његов корен заокружен на три децимале, а ако је негативан, исписати поруку `ne postoji`.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
`4`	`2.000`	`-4`	`ne postoji`

Решење

Након учитавања броја x гранањем на основу услова $x \geq 0$ проверавамо да ли је учитани број ненегативан и ако јесте исписујемо његов корен, а ако није поруку да реалан корен не постоји.

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

int main() {
  double x;
  cin >> x;
  if (x >= 0)
    cout << fixed << showpoint << setprecision(3) << sqrt(x) << endl;
  else
    cout << "ne postoji" << endl;
  return 0;
}
```

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.

Опис улаза

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

Опис излаза

На стандардном излазу приказати реч `da` ако унети број може бити збир година Пере, Мике, Лазе и Ане, а ако не може приказати реч `ne`.

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 акко 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`.

```
#include <iostream>

using namespace std;

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

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

```
#include <iostream>

using namespace std;

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

Задатак: Троцифрен Армстронгов

Троцифрени број је Армстронгов ако је једнак збиру кубова својих цифара. Написати програм који за унети број проверава да ли је Армстронгов.

Опис улаза

Са стандардног улаза се учитава природан број.

Опис излаза

На стандардни излаз исписати `da` ако је унети број троцифрен Армстронгов број, односно `ne` у супротном.

Пример

Улаз	*Излаз*
`370`	`da`

Решење

Наредбом гранања можемо прво проверити да ли је број троцифрен, тако што ћемо проверити да ли је већи или једнак од 100 и мањи или једнак од 999 (нагласимо да за разлику од математике, скраћени запис `100 <= n <= 999` није коректан). Ако број јесте троцифрен, издвајамо му цифру јединица, десетица и стотина, а затим проверавамо да ли је збир њихових кубова једнак броју `n`.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  if (100 <= n && n <= 999) {
    int c0 = n % 10;
    int c1 = (n / 10) % 10;
    int c2 = (n / 100) % 10;
    if (c0*c0*c0 + c1*c1*c1 + c2*c2*c2 == n)
      cout << "da" << endl;
    else
      cout << "ne" << endl;
  } else {
    cout << "ne" << endl;
  }
  return 0;
}
```

2.2 Логички оператори

Циљ овог поглавља је да се увежба употреба логичких оператора у циљу добијања сложенијих услова који се наводе у склопу наредбе гранања.

Задатак: Постоји ли троугао датих дужина страница

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

Опис улаза

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

Опис излаза

Једна линија стандарног излаза која садржи реч `da` ако постоји троугао, иначе садржи реч `ne`.

Пример

Улаз	*Излаз*

`4.3`
`5.4`
`6.7`
	`da`

Решење

Први услов који мора да важи је да су дужине свих страница позитивни бројеви.

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, тада постоји троугао са датим страницама. У супротном такав троугао не постоји. 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 $a - c < b$).

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

```
#include <iostream>

using namespace std;

int main() {
  double a, b, c;
  cin >> a >> b >> c;
  // proveravamo da li postoji trougao sa duzinama stranica a, b i c
  if (a > 0 && b > 0 && c > 0 &&
      a < b + c && b < a + c && c < a + b)
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

Задатак: Растуће цифре

Написати програм који проверава да ли су цифре четвороцифреног броја строго растуће.

Опис улаза

Са стандардног улаза се уноси четвороцифрени број (цео број између 1000 и 9999).

Опис излаза

На стандардни излаз исписати `da` ако цифре унетог броја јесу растуће или `ne` ако нису.

Пример 1

Улаз	*Излаз*
1234	da

Пример 2

Улаз	*Излаз*
3558	ne

Решење

Одређујемо све 4 цифре броја $c_3c_2c_1c_0$ и затим проверавамо да ли важи да је $c_3 < c_2 < c_1 < c_0$. За разлику од математике у програму морамо то урадити тако што ћемо употребити конјункцију три услова.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  // odredjujemo cifre broja
  int c0 = (n / 1) % 10;
  int c1 = (n / 10) % 10;
  int c2 = (n / 100) % 10;
  int c3 = (n / 1000) % 10;
```

```
  if (c3 < c2 && c2 < c1 && c1 < c0)
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

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).

Опис улаза

Са стандардног улаза се учитавају 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.

Опис излаза

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

Пример 1

Улаз	*Излаз*
`5 3` `1 7`	`da`

Пример 2

Улаз	*Излаз*
`5 3` `1 8`	`ne`

Пример 3

Улаз	*Излаз*
`4 4` `4 6`	`da`

Решење

Претпоставимо да се прва краљица налази на пољу (x_1, y_1), а друга на (x_2, y_2).

Провера да ли се даме налазе у истој врсти или истој колони је веома једноставна (довољно је проверити да ли важи да је $x_1 = x_2$ или је $y_1 = y_2$). Две даме се налазе на истој дијагонали ако и само ако је део дијагонале између њих хипотенуза једног једнакокраког правоуглог троугла (коме су катете паралелне ивицама табле). Тада је растојање између краљица по x-оси и по y-оси једнако. Та два растојања су редом $|x_1 - x_2|$ и $|y_1 - y_2|$ и краљице су на истој дијагонали ако и само ако су она међусобно једнака. У језику C++ апсолутну вредност можемо израчунати библиотечком функцијом `abs`, декларисаном у заглављу `<cmath>` у варијанти за реалне бројеве и у заглављу `<cstdlib>` у варијанти за целе бројеве.

```
#include <iostream>

using namespace std;

int main() {
  // ucitavamo koordinate polja na kojima su dame
  int x1, y1, x2, y2;
  cin >> x1 >> y1 >> x2 >> y2;

  // proveravamo da li se dame napadaju
  if (x1 == x2 ||                     // u istoj su vrsti
      y1 == y2 ||                     // u istoj su koloni
      abs(x1 - x2) == abs(y1 - y2))   // na istoj su dijagonali
    cout << "da" << endl;
  else
    cout << "ne" << endl;

  return 0;
}
```

Задатак: Двократно радно време

Продавница ради двократно: од 8 до 12h и од 16 до 20h. За унето време написати да ли је продавница отворена.

Опис улаза

Са стандардног улаза се учитава тренутно време (број сати између 0 и 23 и број минута између 0 и 59).

Опис излаза

На стандардни излаз исписати текст `otvoreno` ако је продавница тренутно отворена или `zatvoreno` ако је продавница тренутно затворена.

Пример 1

Улаз	*Излаз*
`12 01`	`zatvoreno`

Пример 2

Улаз	*Излаз*
`19 59`	`otvoreno`

Решење

Продавница ради ако је купац дошао између 8 и 12 часова или између 16 и 20 часова. Проверу да ли је дошао између 8 и 12 часова вршимо тако што проверавамо да ли је његово време доласка веће или једнако од 8 часова (у 8.00 је продавница већ отворена) и да ли је његово време одласка строго мање од 12 часова (у 12.00 је продавница затворена). Аналогно се проверава и поподневна смена.

```
#include <iostream>

using namespace std;

int main() {
  int sati, minuta;
  cin >> sati >> minuta;
  if ((8 <= sati && sati < 12) ||
      (16 <= sati && sati < 20))
    cout << "otvoreno" << endl;
  else
    cout << "zatvoreno" << endl;
  return 0;
}
```

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.

Опис улаза

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

Опис излаза

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

Пример 1

Улаз	*Излаз*
`5`	`ne moze`

Пример 2

Улаз	*Излаз*
`6`	`moze`

Пример 3

Улаз	*Излаз*
`13`	`ne moze`

Решење

Једно решење можемо засновати на томе да су радови дозвољени ако и само ако сат припада интервалу $[6, 13)$ или $[17, 22)$.

```
#include <iostream>

using namespace std;

int main() {
  int sat;
  cin >> sat;
  if ((6 <= sat && sat < 13) ||
      (17 <= sat && sat < 22))
```

```
    cout << "moze" << endl;
  else
    cout << "ne moze" << endl;
  return 0;
}
```

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.

```
#include <iostream>

using namespace std;

int main() {
  int sat;
  cin >> sat;
  if (sat < 6 || (13 <= sat && sat < 17) || (sat >= 22))
    cout << "ne moze" << endl;
  else
    cout << "moze" << endl;
  return 0;
}
```

Задатак: Осигурање

Осигуравајућа компанија жели да одреди да ли особа има право на исплату накнаде на основу осигурања.

Да би се накнада могла исплатити, захтев мора бити или због штете настале на основу природних узрока (нпр. штета од олује) или због неког другог облика штете. Ако је у питању други облик штете, потребно је да је осигураник претходно уплатио све премије и да је поднео захтев у дозвољеном року. Ако је у питању штета на основу природних узрока, онда особа мора да има уплаћено осигурање које покрива природне катастрофе или да живи у означеној "ризичној" зони. Ако живи у "ризичној" зони, износ захтева не сме да прелази 100000 динара. Да би се накнада штете исплатила, особа не сме имати никакве активне истраге о превари, без обзира на врсту захтева.

Опис улаза

Са стандардног улаза се прво уноси основа за надокнаду штете (`priroda` означава да је штета настала на основу природних узрока).

Након тога се уноси број уплаћених рата премије осигурања (постоји укупно 12 месечних рата и сматра се да је осигураник уплатио све премије ако је уплатио свих 12 рата).

Након тога се уноси дан у текућем месецу (број од 1 до 31) у ком је поднет захтев за осигурање. Сматра се да је захтев поднет у року ако је поднет најкасније до 15. у месецу (укључујући и тај дан).

Након тога се уноси податак о томе да ли осигурање покрива природне катастрофе (1 означава да покрива, а 0 да не покрива).

Након тога се уноси редни број зоне у којој живи осигураник (редни број од 1 до 10). Ризичне зоне су зона 1 и 5.

Након тога се уноси износ захтева за одштету (цео број од 100 до 200000).

На крају се уноси и број истрага о превари које се воде против осигураника (ненегативан цео број).

Опис излаза

На стандардни излаз исписати `da` ако је на основу описаних правила могуће исплатити накнаду штете, односно `ne` ако није.

Пример

Улаз	*Излаз*
priroda 12 13 1 5 85000 1	ne

Решење

Главни изазов у задатку је да се на основу прилично комплексне спецификације направи логички услов који проверава да ли је могуће добити накнаду штетете.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  string osnova;
  cin >> osnova;
  int uplaceno_rata;
  cin >> uplaceno_rata;
  int dan_u_mesecu;
  cin >> dan_u_mesecu;
  bool osiguranje_pokriva_katastrofe;
  cin >> osiguranje_pokriva_katastrofe;
  int zona;
  cin >> zona;
  int iznos;
  cin >> iznos;
  int broj_istraga;
  cin >> broj_istraga;
  if (((osnova != "priroda" &&
        uplaceno_rata >= 12 && dan_u_mesecu <= 15) ||
       (osnova == "priroda" &&
        (osiguranje_pokriva_katastrofe ||
         ((zona == 1 || zona == 5) && iznos <= 100000)))) &&
      broj_istraga == 0)
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

Уместо да задатак решавамо јединственим изразом, гранање можемо извршити на некоико нивоа. Прво можемо елиминисати оне против којих се води неки поступак за превару (јер они сигурно не могу добити накнаду штете). Затим можемо извршити гранање на основу основа накнаде штете (природних или других узрока), а затим у оквиру сваког основа извршити проверу услова специфичних за тај основ.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  string osnova;
  cin >> osnova;
  int uplaceno_rata;
```

```
  cin >> uplaceno_rata;
  int dan_u_mesecu;
  cin >> dan_u_mesecu;
  bool osiguranje_pokriva_katastrofe;
  cin >> osiguranje_pokriva_katastrofe;
  int zona;
  cin >> zona;
  int iznos;
  cin >> iznos;
  int broj_istraga;
  cin >> broj_istraga;
  if (broj_istraga > 0)
    cout << "ne" << endl;
  else {
    if (osnova != "priroda") {
      if (uplaceno_rata >= 12 && dan_u_mesecu <= 15)
        cout << "da" << endl;
      else
        cout << "ne" << endl;
    } else {
      if (osiguranje_pokriva_katastrofe ||
          ((zona == 1 || zona == 5) && iznos <= 100000))
        cout << "da" << endl;
      else
        cout << "ne" << endl;
    }
  }
  return 0;
}
```

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 четвртом квадранту, тачке на негативном делу 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 четвртом квадранту, а да координатни почетак припада свим квадрантима.

Опис улаза

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 $B(x_2, y_2)$

Опис излаза

На стандардном излазу у једној линији приказати реч `da` ако тачке припадају истом квадранту у супротном приказати реч `ne`.

Пример

Улаз	*Излаз*
12 -45 15 23	ne

Решење

Да тачке на осама нису обухваћене задатком или да је формулација била таква да осе не припадају ниједном квадранту могло би се рећи да су две тачке $A(x_1, y_1), B(x_2, y_2)$ у истом квадранту ако су њихове x координате истог знака (истовремено стриктно позитивне или истовремено стриктно негативне) и ако су њихове y

координате истог знака.

Можемо приметити да тачке припадају истом квадранту ако су истовремено испуњени следећи услови:

- њихове x координате су обе позитивне, или обе негативне, или је нека од њих једнака 0
- њихове y координате су обе позитивне, или обе негативне, или је нека од њих једнака 0

Дакле, задатак се може решити испитивањем истинитосне вредности следећег израза.

$$(x_1 = 0 \ \vee \ x_2 = 0 \ \vee \ (x_1 > 0 \ \wedge \ x_2 > 0) \ \vee \ (x_1 < 0 \ \wedge \ x_2 < 0)) \wedge$$
$$(y_1 = 0 \ \vee \ y_2 = 0 \ \vee \ (y_1 > 0 \ \wedge \ y_2 > 0) \ \vee \ (y_1 < 0 \ \wedge \ y_2 < 0))$$

```
#include <iostream>

using namespace std;

int main() {
   int x1, y1, x2, y2;
   cin >> x1 >> y1 >> x2 >> y2;

   if ((x1 == 0 || x2 == 0 || (x1 > 0 && x2 > 0) || (x1 < 0 && x2 < 0)) &&
       (y1 == 0 || y2 == 0 || (y1 > 0 && y2 > 0) || (y1 < 0 && y2 < 0)))
     cout << "da" << endl;
   else
     cout << "ne" << endl;

   return 0;
}
```

Овај израз се може мало поједноставити на разне начине.

На пример, можемо приметити да су тачке у истом квадранту ако су њихове x координате истог знака или је једна од њих 0, и ако су y координате истог знака или је једна од њих 0. Два броја `a` и `b` су истог знака или је један од њих 0 ако важи да је

$$a = 0 \ \vee \ b = 0 \ \vee \ (a > 0) \Leftrightarrow (b > 0)$$

што доводи до израза

$$(x_1 = 0 \ \vee \ x_2 = 0 \ \vee \ (x_1 > 0) = (x_2 > 0)) \wedge$$
$$(y_1 = 0 \ \vee \ y_2 = 0 \ \vee \ (y_1 > 0) = (y_2 > 0))$$

Напоменимо да услов

$$a \geq 0 \ \Leftrightarrow b \geq 0$$

није одговарајући (на пример, ако a има вредност 0, а b има вредност -1, његова вредност је нетачно, што је супротно од онога што би требало да буде).

```
#include <iostream>

using namespace std;

int main() {
   int x1, y1, x2, y2;
   cin >> x1 >> y1 >> x2 >> y2;
```

```
  if ((x1 == 0 || x2 == 0 || (x1 > 0) == (x2 > 0)) &&
      (y1 == 0 || y2 == 0 || (y1 > 0) == (y2 > 0)))
    cout << "da" << endl;
  else
    cout << "ne" << endl;

  return 0;
}
```

Можемо приметити да је полазни услов еквивалентан услову да су оба броја ненегативна или оба броја непозитивна.

$$(a \geq 0 \ \wedge \ b \geq 0) \ \vee \ (a \leq 0 \ \wedge \ b \leq 0)$$

Према томе задатак можемо решити испитивањем истинитосне вредности израза

$$((x_1 \geq 0 \ \wedge \ x_2 \geq 0) \ \vee \ (x_1 \leq 0 \ \wedge \ x_2 \leq 0)) \wedge \\ ((y_1 \geq 0 \ \wedge \ y_2 \geq 0) \ \vee \ (y_1 \leq 0 \ \wedge \ y_2 \leq 0))$$

```
#include <iostream>

using namespace std;

int main() {
  int x1, y1, x2, y2;
  cin >> x1 >> y1 >> x2 >> y2;

  if (((x1 >= 0 && x2 >= 0) || (x1 <= 0 && x2 <= 0)) &&
      ((y1 >= 0 && y2 >= 0) || (y1 <= 0 && y2 <= 0)))
    cout << "da" << endl;
  else
    cout << "ne" << endl;

  return 0;
}
```

Приметимо даље и да тачке припадају истом квадаранту ако њихове x координате нису различитог знака и њихове y координате нису различитог знака. Бројеви `a` и `b` су различитог знака ако је први позитиван а други негативан или први негативан или је први негативан тј. ако важи

$$(a > 0 \ \wedge \ b < 0) \ \vee \ (a < 0 \ \wedge \ b > 0)$$

На основу овога задатак можемо решити и испитивањем вредности израза

$$\neg((x_1 > 0 \ \wedge \ x_2 < 0) \ \vee \ (x_1 < 0 \ \wedge \ x_2 > 0)) \wedge \\ \neg((y_1 > 0 \ \wedge \ y_2 < 0) \ \vee \ (x_1 < 0 \ \wedge \ x_2 > 0))$$

Напоменимо и да то што бројеви нису различитог знака не значи да су истог знака (јер је и вредност 0 укључена).

Још један начин да се провера изврши је да се производ бројева упореди са нулом. Ако њихов производ није негативан тј. ако је већи или једнак нули ($x_1 \cdot x_2 \geq 0$) координате нису различитог знака. Слично извршимо проверу за y координте. Дакле, задатак се може решити следећим изразом.

$$x_1 \cdot x_2 \geq 0 \ \wedge \ y_1 \cdot y_2 \geq 0$$

Напоменимо да иако је веома кратко и елегантно, ово решење се може користити ако смо сигурни да производ координата неће довести до прекорачења (што је случај у нашем задатку).

```
#include <iostream>

using namespace std;

int main() {
   int x1, y1, x2, y2;
   cin >> x1 >> y1 >> x2 >> y2;

   if (x1 * x2 >= 0 && y1 * y2 >= 0)
     cout << "da" << endl;
   else
     cout << "ne" << endl;

   return 0;
}
```

Приметимо да се у много случајева исти услов користи и да се провере координате x и да се провере координате y, тако да има смисла тај услов издвојити у посебну функцију (на пример, дефинисати функцију која проверава да ли су две тачке на правој са исте стране тачке 0, при чему се подразумева да она припада и левој и десној полуправој којима је почетак).

```
#include <iostream>

using namespace std;

bool razlicitog_znaka(int a, int b) {
  return (a > 0 && b < 0) || (a < 0 && b > 0);
}

int main() {
   int x1, y1, x2, y2;
   cin >> x1 >> y1 >> x2 >> y2;

   if (!razlicitog_znaka(x1, x2) &&
       !razlicitog_znaka(y1, y2))
     cout << "da" << endl;
   else
     cout << "ne" << endl;

   return 0;
}
```

Задатак: Непознати број - скочко

Шифра за улаз у просторију је непознати троцифрени број. Добили смо тајну информацију да је приликом уноса броја 682 тачно једна цифра тачна и налази се на правој позицији, а да приликом уноса броја 738 ниједна цифра није тачна. Ово нам помаже да елиминишемо неке троцифрене бројеве. Написати програм који за унети троцифрени број проверава да ли може бити шифра за улаз.

Опис улаза

Са стандардног улаза се уноси један троцифрени број.

Опис излаза

На стандардни излаз исписати `da` ако унети број може односно `ne` ако не може да буде шифра за улаз.

Пример 1

Улаз	*Излаз*
547	ne

Објашњење

Када би ово била комбинација, при уносу броја 738 би једна цифра била тачна (додуше не на правом месту).

Пример 2

Улаз

```
649
```

Излаз

```
da
```

Решење

Нека су c_2, c_1 и c_0 цифра стотина, десетица и јединица броја n.

Пошто у броју 738 није погођена ниједна цифра, свака од цифара c_2, c_1 и c_0 је различита и од 7 и од 8 и од 3.

Пошто је у броју 682 тачно једна цифра погођена и то на правом месту, важи или да је $c_2 = 6$, а остале цифре нису ни 2, ни 6, ни 8, или је $c_1 = 8$, а остале цифре нису ни 2, ни 6, ни 8, или је $c_0 = 2$, а остале цифре нису ни 2, ни 6, ни 8.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  // odredjujemo cifre broja n
  int c0 = n % 10;
  int c1 = (n / 10) % 10;
  int c2 = (n / 100) % 10;

  // za 682 tacno jedna je cifra tacna i na pravom je mestu
  bool uslov682 =
    (c2 == 6 &&
     c1 != 2 && c1 != 6 && c1 != 8 &&
     c0 != 2 && c0 != 6 && c0 != 8) ||
    (c1 == 8 &&
     c2 != 2 && c2 != 6 && c2 != 8 &&
     c0 != 2 && c0 != 6 && c0 != 8) ||
    (c0 == 2 &&
     c2 != 2 && c2 != 6 && c2 != 8 &&
     c1 != 2 && c1 != 6 && c1 != 8);

  // za 738 nista nije korektno
  bool uslov738 =
    c2 != 7 && c2 != 3 && c2 != 8 &&
    c1 != 7 && c1 != 3 && c1 != 8 &&
    c0 != 7 && c0 != 3 && c0 != 8;

  if (uslov682 && uslov738)
    cout << "da" << endl;
  else
    cout << "ne" << endl;

  return 0;
}
```

2.3 Гранање на основу дискретне вредности

Честа је ситуација да се наредбе извршавају у односу на то коју вредност из неког малог скупа вредности тренутно има нека променљива. Иако је то типичан сценарио употребе наредбе `switch`, није неуобичајено да се грање изврши и помоћу наредбе `if`.

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).

Опис улаза

Са стандардног улаза учитавају се два броја:

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

Опис излаза

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

Пример 1

Улаз	*Излаз*
11 2014	30

Пример 2

Улаз	*Излаз*
2 2014	28

Пример 3

Улаз	*Излаз*
2 2016	29

Пример 4

Улаз	*Излаз*
2 2000	29

Пример 5

Улаз	*Излаз*
2 2100	28

Пример 6

Улаз	*Излаз*
4 2019	30

Пример 7

Улаз	*Излаз*
8 2018	31

Решење

У овом задатку вршимо гранање на основу могућих (дискретних) вредности промењиве `mesec`. 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. Година је преступна ако је дељива са 4 и није дељива са 100, или је дељива са 400.

```
#include <iostream>

using namespace std;

// 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() {
  // ucitavamo 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`. Све месеце са истим бројем дана можемо груписати у исту грану.

```
#include <iostream>

using namespace std;

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

  // 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;

  // ispisujemo rezultat
  cout << brojDana << endl;

  return 0;
}
```

Још један начин да се задатак реши је да се направи низ бројева који представљају број дана у сваком од 12 месеци и да се након учитавања месеца тражени податак прочита са одговарајућег места у низу (пошто индекси у низу почињу од 0, а бројеви месеца од 1, онда на прво месту у низу можемо уписати вештачки неки број, на пример, 0). Једино треба бити обазрив и накнадно проверити да ли је у питању месец фебруар преступне године (ако јесте број дана треба увећати за један тј. поставити на 29).

```
#include <iostream>
```

```
using namespace std;

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

  // broj dana u svakom mesecu
  int brojDanaUMesecu[] =
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

  // citamo broj dana
  int brojDana = brojDanaUMesecu[mesec];
  // posebno obradjujemo februar prestupnih godina
  if (mesec == 2 && prestupna(godina))
    brojDana++;

  // ispisujemo rezultat
  cout << brojDana << endl;

  return 0;
}
```

2.4 Гранање на основу припадности интервалима

Честа је ситуација да се наредбе извршавају у односу на то у ком се интервалу (од неколико понуђених, дисјунктних интервала) налази тренутна вредност неке променљиве. У овим ситуацијама се гранање често врши коришћењем конструкције `else-if`.

Задатак: Атлетичари

У зависности од трка које трче, атлетичаре делимо на краткопругаше, средњепругаше и дугопругаше. Краткопругаши (или спринтери) трче трке дужине највише 400 метара, средњепругаши трче дуже трке од њих, али највише дужине једне миље, док дугопругаши трче трке дуже од тога. Написати програм којим се на основу дужине трке у којој атлетичар учествује одређује категорија којој он припада.

Опис улаза

Дужина трке у метрима - цео број од 60 до 42195.

Опис излаза

На стандардни излаз исписати једну од следећих речи: `kratko`, `srednje`, `dugo`.

Пример 1

Улаз	*Излаз*
`400`	`kratko`

Пример 2

Улаз	*Излаз*
`1500`	`srednje`

Решење

Из формулације задатка произилази да имамо три категорије, које су одређене следећим интервалима:

- ако дужина у метрима припадају интервалу $[60, 400]$, тркач је краткопругаш;
- ако дужина у метрима припада интервалу $(400, 1609]$, тркач је средњепругаш (једна миља има 1609 метара);

- ако дужина у метрима припада интервалу $(1609, 42195]$, тркач је дугопругаш.

Одавде произилази кôд са три међусобно независна услова, којима се у произвољном редоследу проверава припадност једном од три интервала $[60, 400]$, $(400, 1609]$ и $(1609, 42195]$. Пошто знамо да је унети број метара увек у интервалу $[60, 42195]$, довољно је испитивати припадност интервалима $(-\infty, 400]$, $(400, 1609]$ и $(1609, +\infty)$.

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

int main() {
  int m; // Duzina trke u metrima
  cin >> m;

  if (m <= 400)
    cout << "kratko" << endl;
  if (m > 400 && m <= 1609)
    cout << "srednje" << endl;
  if (m > 1609)
    cout << "dugo" << endl;

  return 0;
}
```

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

- ако број метара није већи од 400, тркач је краткопругаш;
- у супротном (број метара јесте већи од 400): ако је број метара није већи од 1609 (припада другом интервалу), тркач је средњепругаш;
- у супротном (број метара је већи од 1609), тркач је дугопругаш.

Овим се избегавају провере које нису заиста неопходне.

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

int main() {
  int m; // Duzina trke u metrima
  cin >> m;

  if (m <= 400)
    cout << "kratko" << endl;
  else if (m <= 1609)
    cout << "srednje" << endl;
  else
    cout << "dugo" << endl;

  return 0;
}
```

Аналогно претходном решењу, можемо проверавати припадност броја метара идући здесна од интервала $(1609, \infty)$ наниже.

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

int main() {
  int m; // Duzina trke u metrima
  cin >> m;

  if (m > 1609)
    cout << "dugo" << endl;
```

```
  else if (m > 400)
    cout << "srednje" << endl;
  else
    cout << "kratko" << endl;

  return 0;
}
```

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.

Опис улаза

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

Опис излаза

На стандардни излаз приказати успех ученика (реч `odlican`, `vrlodobar`, `dobar`, `dovoljan` или `nedovoljan`).

Пример

Улаз	*Излаз*
`3.75`	`vrlodobar`

Решење

Гранање на основу припадности интервалима

Одређивање успеха врши се на основу припадности датог просека интервалима $[1, 2{,}5]$, $(2{,}5, 3{,}5]$ итд.

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.

```
#include <iostream>

using namespace std;

int main() {
  double prosek;
  cin >> prosek;
  if (prosek >= 4.5)
    cout << "odlican" << endl;
  if (prosek >= 3.5 && prosek < 4.5)
    cout << "vrlodobar" << 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;
  return 0;
}
```

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 врлодобар 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.

```
#include <iostream>

using namespace std;

int main() {
  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;
  return 0;
}
```

Свођење на целобројне вредности успеха и коришћење низа

Постоји начин да се сваки интервал преслика у вредност која га представља: интервал одличног успеха $[4,50, 5,00)$ у вредност 5, интервал врлодоброг успеха $[3,50, 4,50)$ у вредност 4 и тако даље. Да би се то урадило, довољно је просек заокружити на њему најближи цео број. У језику C++ то је могуће урадити коришћењем библиотечке функције `round` декларисане у заглављу `<cmath>`. Међутим, проблем настаје око вредности које су тачно на граници (3,5, 4,5 и слично), јер се функција заокруживања у разним језицима различито понаша у тим граничним случајевима и потребно је веома прецизно познавати њено понашање. Зато је боље употребити технику који каже да се заокруживање на најближи цео број тако да се граничне вредности заокруже навише може добити тако што се броју дода $0,5$ и резултат заокружи наниже тј. израчуна се $\lfloor x + 0,5 \rfloor$. У језику C++ то је могуће урадити коришћењем библиотечке функције `floor` ili `trunc`, које су декларисане у заглављу `<cmath>`. Прва одређује најближи цео број мањи или једнак датом, а друга само одсеца децимале, тако да се вредности ове две функције поклапају за ненегативне бројеве, а разликују код негативних.

Када одредимо цео број који кодира успех, назив можемо прочитати из низа.

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

int main() {
  double prosek;
  cin >> prosek;
  int uspeh = floor(prosek + 0.5);
  string uspesi[] = {"nedovoljan", "dovoljan", "dobar", "vrlodobar", "odlican"};
  cout << uspesi[uspeh - 1] << endl;
```

```
  return 0;
}
```

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.

Опис улаза

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

Опис излаза

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

Решење

Гранање на основу припадности интервалима

Једно решење се може засновати на гранању и оцена се може одредити на основу тога ком од интервала $[0, 50]$, $[51, 60]$, $[61, 70]$, $[71, 80]$, $[81, 90]$ или $[91, 100]$ припада број поена. Задатак може да се реши и употребом конструкције `else-if` и поређењем броја поена редом са бројевима 51, 61, 71, 81 и 91 и проверавањем да ли је број поена строго мањи од њих.

```
// poeni osvojeni na ispitu
int poeni;
cin >> poeni;

// ocena na ispitu
int ocena;
if (poeni < 51)
  ocena = 5;
else if (poeni < 61)
  ocena = 6;
else if (poeni < 71)
  ocena = 7;
else if (poeni < 81)
  ocena = 8;
else if (poeni < 91)
  ocena = 9;
else
  ocena = 10;

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

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 $10 \cdot (o - 1) \leq p - 1$. 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: Годишње доба

Рећи ћемо да пролеће почиње 20. марта (укључујући и тај дан) и траје до 21. јуна (без тог дана). Лето почиње 21. јуна (укључујући тај дан) и траје до 23. септембра (без тог дана). Јесен почиње 23. септембра (укључујући и тај дан) и траје до 21. децембра (без тог дана). Осталих дана је зима. Напиши програм који на основу унетог датума одређује годишње доба.

Опис улаза

Са стандардног улаза се уноси дан, а затим и месец (сваки број у посебном реду). Број 1 означава јануар, 2 фебруар, 3 март итд. Сматрати да је унети датум исправан.

Опис излаза

На стандардни излаз исписати слово `p` (пролеће), `l` (лето), `j` (јесен) или `z` (зима).

Пример 1

Улаз	*Излаз*
`1` `2`	`z`

Објашњење

Унет је први фебруар и он је током зиме.

Пример 2

Улаз

```
20
3
```

Излаз

```
p
```

Објашњење

Унет је 20. март и сматрамо да је он први дан пролећа.

Пример 3

Улаз

```
19
3
```

Излаз

```
z
```

Пример 4

Улаз

```
21
6
```

Излаз

```
l
```

Решење

Задатак најлакше можемо решити тако што датум представимо у облику троцифреног или четвороцифреног броја (који добијамо тако што месец помножмо са 100 и додамо дан) и затим употребимо гранање на основу припадности интервалима (помоћу конструкције `else-if`).

```
#include <iostream>

using namespace std;

int main() {
    int dan, mesec;
    cin >> dan >> mesec;
    int mes_dan = 100 * mesec + dan;
    if (mes_dan < 320)
        cout << 'z' << endl;
    else if (mes_dan < 621)
        cout << 'p' << endl;
    else if (mes_dan < 923)
        cout << 'l' << endl;
    else if (mes_dan < 1221)
      cout << 'j' << endl;
    else
      cout << 'z' << endl;

    return 0;
}
```

Задатак је могуће решити и угнежђеним гранањем. Прво испитујемо месец. Ако откријемо да се ради о неком месецу који се простире кроз два годишња доба, посебно испитујемо дан.

```
#include <iostream>

using namespace std;

int main() {
  int dan, mesec;
  cin >> dan >> mesec;
  if (mesec < 3)
    cout << 'z' << endl;
  else if (mesec == 3) {
    if (dan < 20)
      cout << 'z' << endl;
    else
      cout << 'p' << endl;
  } else if (mesec < 6)
    cout << 'p' << endl;
  else if (mesec == 6) {
    if (dan < 21)
      cout << 'p' << endl;
    else
      cout << 'l' << endl;
  } else if (mesec < 9)
    cout << 'l' << endl;
  else if (mesec == 9) {
    if (dan < 23)
      cout << 'l' << endl;
```

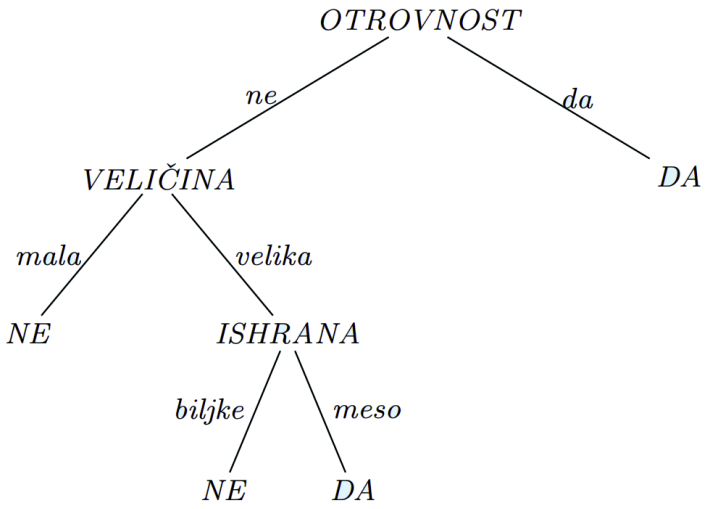
```
    else
      cout << 'j' << endl;
  } else if (mesec < 12)
    cout << 'j' << endl;
  else if (mesec == 12) {
    if (dan < 21)
      cout << 'j' << endl;
    else
      cout << 'z' << endl;
  }
  return 0;
}
```

2.5 Хијерархија услова

Као што је у почетку већ речено, а у неким примерима већ показано, чест је случај да се до решења долази угнежђавањем наредби `if`. У зависности од одговора на прво питање постављају се нова питања и тако све док се не добије довољно информација да се дође до коначног одговора.

Задатак: Дрво одлучивања

Написати програм који пита корисника о неким карактеристикама животиња и на основу постављених питања одређује да ли је животиња опасна по човека или није. Питања треба да буду одређена дрветом одлучивања приказаном на следећој слици.



Опис улаза

Са стандардног улаза корисник уноси одговоре на постављена питања (`da`, `ne`, `mala`, `velika`, `meso`, `biljke`).

Опис излаза

На стандардни излаз исписати текст постављених питања (у истом облику као што је приказано у примеру) и након тога одговор на питање да ли је животиња опасна по човека или не.

Пример 1

Улаз	*Излаз*
`da`	`OTROVNOST:` `DA`

Објашњење

Програм прво корисника пита за отровност исписујући текст `OTROVNOST:`. Када корисник унесе одговор `da`, програм закључује да животиња јесте опасна и исписује одговор `DA`.

Пример 2

Улаз

```
ne
velika
biljke
```

Излаз

```
OTROVNOST:
VELICINA:
ISHRANA:
NE
```

Објашњење

Програм прво корисника пита за отровност исписујући текст `OTROVNOST:`. Када корисник унесе одговор `ne`, програм наставља са питањима исписујући текст `VELICINA:`. Када корисник унесе одговор `velika`, програм пита за исхрану исписујући текст `ISHRANA:`. Након што корисник одговори да животиња једе биљке, програм закључује да животиња није опасна и исписује `ne`.

Решење

Структура гранања мора да одговара приказаном дрвету одлучивања. Програм прво пита за отровност животиње. Ако добије одговор `da`, може да закључи да је животиња опасна, испише одговор `DA` и заврши са радом. У супротном, ако добије одговор `ne`, онда пита за величину животиње. Ако добије одговор `mala`, може да закључи да животиња није опасна, испише одговор `NE` и заврши са радом. У супротном, ако добије одговор `velika`, програм пита за исхрану животиња и у зависности од тога да ли добије одговор `biljke` или `meso`, исписује одговор `NE` или `DA`.

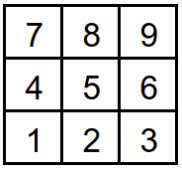
```cpp
#include <iostream>
#include <string>

using namespace std;

int main() {
  cout << "OTROVNOST:" << endl;
  string otrovnost;
  cin >> otrovnost;
  if (otrovnost == "da")
    cout << "DA" << endl;
  else if (otrovnost == "ne") {
    cout << "VELICINA:" << endl;
    string velicina;
    cin >> velicina;
    if (velicina == "mala")
      cout << "NE" << endl;
    else if (velicina == "velika") {
      cout << "ISHRANA:" << endl;
      string ishrana;
      cin >> ishrana;
      if (ishrana == "biljke")
        cout << "NE" << endl;
      else if (ishrana == "meso")
        cout << "DA" << endl;
    }
  }
  return 0;
}
```

Задатак: Икс-окс

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 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:



Опис улаза

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-координату пиксела на који је кликнуто мишем.

Опис излаза

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

Пример 1		**Пример 2**		**Пример 3**		**Пример 4**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
1 1	1	120 280	8	100 201	7	101 300	8

Решење

Пошто је у овом задатку број случајева релативно мали, можемо га решити 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.

```
#include <iostream>

using namespace std;

int main() {
  // 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;

  return 0;
}
```

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 $1 \leq y \leq 100$.

```
#include <iostream>

using namespace std;

int main() {
  // 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;
  if (201 <= x && x <= 300 && 1 <= y && y <= 100)
    kvadrat = 3;
  if (1 <= x && x <= 100 && 101 <= y && y <= 200)
    kvadrat = 4;
  if (101 <= x && x <= 200 && 101 <= y && y <= 200)
    kvadrat = 5;
  if (201 <= x && x <= 300 && 101 <= y && y <= 200)
    kvadrat = 6;
  if (1 <= x && x <= 100 && 201 <= y && y <= 300)
    kvadrat = 7;
  if (101 <= x && x <= 200 && 201 <= y && y <= 300)
    kvadrat = 8;
  if (201 <= x && x <= 300 && 201 <= y && y <= 300)
    kvadrat = 9;

  // ispisujemo resenje
  cout << kvadrat << endl;
  return 0;
}
```

Целобројно дељење

Најелегантнији начин да одредимо квадрат у коме се налази пиксел је да посебно одредимо редни број врсте v, а затим и редни број колоне k (оба редна броја бројимо од нуле до два и то почевши од доњег левог квадрта, па надесно, тј. навише). Тада редни број квадрата можемо добити као $3v + k + 1$ (јер се квадрати броје од 1). Редни број врсте и колоне можемо одредити целобројним дељењем. Ако дужину странице означимо са $a = 100$, тада су x-координате пиксела који припадају колони 0 између 1 и a, координате пиксела који припадају колони 1 су између $a + 1$ и $2a$, а колони 3 су између $2a + 1$ и $3a$. Ако пиксел има x-координату једнаку x, редни број колоне је највећи број k такав да је $k \cdot a + 1 \leq x$, тј. да је $k \leq \frac{x-1}{a}$. Важи да је $k = \lfloor \frac{x-1}{a} \rfloor$. Слично, ако је y-координата пиксела једнака y, број врсте је $v = \lfloor \frac{y-1}{a} \rfloor$.

```
#include <iostream>

using namespace std;

int main() {
  const int a = 100; // dimenzija kvadrata
  int x, y;          // koordinate piksela
  cin >> x >> y;
  // redni broj vrste i kolone u kojoj se nalazi piksel
  int k = (x - 1) / a, v = (y - 1) / a;
  // redni broj kvadrata
  int kvadrat = 3 * v + k + 1;
  cout << kvadrat << endl;
  return 0;
}
```

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.

Опис улаза

У првом реду коефицијент a, у другом коефицијент b.

Опис излаза

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

Пример 1

Улаз	*Излаз*
1	-1.00
1	

Пример 2

Улаз	*Излаз*
0	BESKONACNO RESENJA
0	

Пример 3

Улаз	*Излаз*
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.

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

int main() {
  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;
  }

  return 0;
}
```

Задатак: Квадратна једначина

Написати програм који на основу унетих коефицијената квадратне једначине $ax^2 + bx + c = 0$ испитује број и природу решења ове јединачине и исписује их.

Опис улаза

Са стандардног улаза се уносе три цела броја a, b и c.

Опис излаза

У првом реду исписати број различитих реалних решења једначине. Ако их има бесконачно много, исписати `inf`. Ако једначина има коначно много различитих реалних решења, у другом реду исписати та различита реална решења, уређена по величини (по потреби их заокружити на 5 децимала).

Пример 1

Улаз	*Излаз*
`1 -2 1`	`1` `1`

Пример 2

Улаз	*Излаз*
`0 1 1`	`1` `-1`

Пример 3

Улаз	*Излаз*
`1 -1 -1`	`2` `-0.61803 1.61803`

Решење

Први услов који треба проверити је да ли је $a = 0$, јер ако јесте, у питању је линеарна, а не квадратна једначина.

Ако установимо да је једначина линеарна, тј. облика $bx + c = 0$, број и природа решења зависи од тога да ли је $b = 0$. Ако јесте, онда је у питању једначина $0 \cdot x + c = 0$. Ако је $c = 0$, сваки реални број x је њено решење, а ако је $c \neq 0$, једначина нема решења. Ако је $b \neq 0$, једначина има јединствено реално решење $-c/b$.

Ако је $a \neq 0$, тада је у питању квадратна једначина чији број решења зависи од дискриминанте $D = b^2 - 4ac$. Ако је она позитивна, једначина има два реална решења, ако је једнака нули има јединствено реално решење, а ако је негативна, једначина има два конјуговано комплексна решења и нема реалних решења.

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

int main() {
  int a, b, c;
  cin >> a >> b >> c;
  if (a == 0) {
    // jednacina je linearna, oblika bx+c=0
    if (b == 0) {
```

```
      // jednacina je oblika c=0
      if (c == 0)
        // jednacina je oblika 0=0 i svaki realni broj joj je resenje
        cout << "inf" << endl;
      else {
        // jednacina nema resenja
        cout << 0 << endl;
      }
    } else {
      // za b != 0 jednacina bx+c=0 ima jedinstveno resenje -c/b
      cout << 1 << endl;
      cout << fixed << setprecision(5) << -c/b << endl;
    }
  } else {
    // jednacina je kvadratna

    // diskriminanta
    int D = b*b - 4*a*c;
    if (D > 0) {
      // postoje dva razlicita realna resenja
      cout << 2 << endl;
      double x1 = (-b - sqrt(D)) / (2.0 * a);
      double x2 = (-b + sqrt(D)) / (2.0 * a);
      // pazimo da ih ispisemo u rastucem redosledu
      if (a > 0)
        cout << fixed << setprecision(5) << x1 << " " << x2 << endl;
      else
        cout << fixed << setprecision(5) << x2 << " " << x1 << endl;
    } else if (D < 0) {
      // resenja su konjugovano kompleksna i nema realnih resenja
      cout << 0 << endl;
    } else {
      // postoji dvostruko realno resenje
      cout << 1 << endl;
      cout << (-b + sqrt(D)) / (2.0 * a) << endl;
    }
  }
  return 0;
}
```

2.6 Лексикографско поређење торки

Један од услова који се често гранањем испитује је лексикографско поређење торки исте дужине. Торка $(a_1, a_2, \ldots, a_k)$ и $(b_1, b_2, \ldots, b_k)$ се пореде тако што се прво упореде вредности a_1 и b_1. Ако је нека од њих већа од друге, већа је и одговарајућа торка. Ако је $a_1 = b_1$, онда се прелази на поређење $(a_2, \ldots, a_k)$, по истом принципу (пореде се a_2 и b_2 итд.).

Задатак: Предње седиште

Полиција врши контролу саобраћаја. Написати програм којим се испитује да ли дете које тренутно седи на предњем седишту аутомобила има по закону право да седи на предњем седишту (по закону на предњем седишту могу седети искључиво особе које су навршиле 12 година старости).

Опис улаза

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

Опис излаза

Исписати на стандардном излазу DA ако особа сме да седи на предњем седишту или NE ако особа не сме да седи на предњем седишту. Ако је данашњи датум тачно 12 година након датума рођења, сматра се да је особа напунила 12 година (без обзира на тачно време рођења).

Пример 1

Улаз	*Излаз*
1 5 1992 1 5 2004	DA

Пример 2

Улаз	*Излаз*
24 11 2012 5 10 2024	NE

Решење

У задатку се захтева поређење два датума – датума који је тачно 12 година након године рођења и задатог датума. Ако се датуми представе уређеним тројкама облика (g, m, d) тада се поређење датума своди на лексикографско поређење ових уређених тројки. Приметимо да број дана у месецу и преступне године не утичу никако на коректност овог поступка. Постоје различити начини да се лексикографско поређење торки изврши.

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 <, >, <=, >=,

```
#include <iostream>
#include <tuple>

using namespace std;

int main() {
  int d1, m1, g1; // datum rodjenja
  int d2, m2, g2; // datum u kom se ispituje punoletstvo
  cin >> d1 >> m1 >> g1
      >> d2 >> m2 >> g2;

  if (make_tuple(g2, m2, d2) >= make_tuple(g1 + 12, m1, d1))
    cout << "DA" << endl;
  else
    cout << "NE" << endl;

  return 0;
}
```

Лексикографско поређење помоћу једног израза

Особа рођена на дан (d_r, m_r, g_r) ће моћи да седи на предњем седишту (имаће пуних 12 година) на дан (d, m, g):

- ако је $g_r + 12 < g$, или;
- ако је $g_r + 12 = g$, али и $m_r < m$, или;
- ако је $g_r + 12 = g$ и $m_r = m$, али и $d_r \leq d$.

Дакле, на основу овога могуће је формирати сложени логички израз који одређује да ли особа може да седи на предњем седишту.

```
#include <iostream>

using namespace std;
```

```
int main() {
  int d1, m1, g1; // datum rodjenja
  int d2, m2, g2; // danasnji datum
  cin >> d1 >> m1 >> g1
      >> d2 >> m2 >> g2;
  if ((g2 > g1 + 12) ||
      (g2 == g1 + 12 && m2 > m1) ||
      (g2 == g1 + 12 && m2 == m1 && d2 >= d1))
    cout << "DA" << endl;
  else
    cout << "NE" << endl;
  return 0;
}
```

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.

```
#include <iostream>

using namespace std;

int main() {
  int d1, m1, g1; // datum rodjenja
  int d2, m2, g2; // danasnji datum
  cin >> d1 >> m1 >> g1
      >> d2 >> m2 >> g2;
  bool napunio12;
  if (g2 > g1 + 12)
    napunio12 = true;
  else if (g2 < g1 + 12)
    napunio12 = false;
  else { // g1 == g2
    if (m2 > m1)
      napunio12 = true;
    else if (m2 < m1)
      napunio12 = false;
    else { // m1 == m2
      if (d2 >= d1)
        napunio12 = true;
      else
        napunio12 = false;
      // napunio12 = d2 >= d1;
    }
  }

  if (napunio12)
    cout << "DA" << endl;
  else
    cout << "NE" << endl;

  return 0;
```

```
}
```

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.

```
#include <iostream>

using namespace std;

// 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() {
  int d1, m1, g1; // datum rodjenja
  int d2, m2, g2; // danasnji datum
  cin >> d1 >> m1 >> g1
      >> d2 >> m2 >> g2;

  if (porediDatume(d2, m2, g2, d1, m1, g1 + 12) >= 0)
    cout << "DA" << endl;
  else
    cout << "NE" << endl;
  return 0;
}
```

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.

Опис улаза

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.

Опис излаза

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

Поређење се може реализовати помоћу библиотечке подршке за торке и њихово лексикографско поређење (које је описано у задатку Предње седиште).

```
#include <iostream>
#include <tuple>

using namespace std;

// 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 make_tuple(ukupnoA, progA) > make_tuple(ukupnoB, progB);
}

int main() {
  // broj poena prvog takmicara iz matematike i programiranja
  int mat1, prog1;
  // broj poena drugog takmicara
  int mat2, prog2;

  // Ucitavamo podatke
  cin >> mat1 >> prog1;
  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;
}
```

Сваки такмичар је представљен уређеним паром бројева (поенима из математике и програмирања) и потребно је одредти 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` акко је први бољи од другог у овој релацији. Различити могући начини имплементације лексикографског поређења описани су у задатку Предње седиште.

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.

```
#include <iostream>

using namespace std;

// 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.

```
#include <iostream>

using namespace std;

// 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);
}

int main() {
  int mat1, prog1; // broj poena prvog takmicara
  int mat2, prog2; // broj poena drugog takmicara

  // Ucitavamo podatke
  cin >> mat1 >> prog1;
  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;
}
```

Задатак: Верзије софтвера

Свака верзија софтвера се означава се 3 броја. На пример 2.13.5 је пета подподверзија тринаесте подверзије друге верзије софтвера. Написати програм који проверава да ли тренутно инсталирана верзија софтвера на рачунару задовољава потребе корисника.

Опис улаза

Са стандардног улаза се уноси верзија софтвера коју потребно имати на рачунару и верзија софтвера која је тренутно присутна (по три природна броја раздвојена размацима).

Опис излаза

На стандардни излаз исписати `da` ако је тренутна верзија софтвера једнака потребној или је новија од тога, тј. `ne` у супротном.

Пример 1

Улаз	*Излаз*
`2 5 3` `2 6 1`	`da`

Пример 2

Улаз	*Излаз*
`2 5 3` `2 4 14`	`ne`

Решење

Пошто не знамо колико цифара имају ознаке, није једноставно три броја претворити у један. Зато ћемо верзије поредити лексикографски. Прво се пореди први број, па ако је он једнак пореди се други број, па ако је и он једнак, пореди се трећи број.

Можемо написати један логички израз којим се проверава да ли је инсталирана верзија у реду.

```
#include <iostream>

using namespace std;

int main() {
```

```
  int potrebnoA, potrebnoB, potrebnoC;
  cin >> potrebnoA >> potrebnoB >> potrebnoC;
  int instaliranoA, instaliranoB, instaliranoC;
  cin >> instaliranoA >> instaliranoB >> instaliranoC;
  if (instaliranoA > potrebnoA ||
      (instaliranoA == potrebnoA && instaliranoB > potrebnoB) ||
      (instaliranoA == potrebnoA && instaliranoB == potrebnoB && instaliranoC >= potrebnoC))
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

Уместо јединственог логичког израза можемо надовезати испитивање више једноставних услова.

```
#include <iostream>

using namespace std;

int main() {
  int potrebnoA, potrebnoB, potrebnoC;
  cin >> potrebnoA >> potrebnoB >> potrebnoC;
  int instaliranoA, instaliranoB, instaliranoC;
  cin >> instaliranoA >> instaliranoB >> instaliranoC;
  bool OK;
  if (instaliranoA > potrebnoA)
    OK = true;
  else if (instaliranoA < potrebnoA)
    OK = false;
  else if (instaliranoB > potrebnoB)
    OK = true;
  else if (instaliranoB < potrebnoB)
    OK = false;
  else if (instaliranoC >= potrebnoC)
    OK = true;
  else
    OK = false;

  if (OK)
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

2.7 Додатни примери задатака са гранањем

До сада смо описали неке честе сценарије употребе наредби гранања, међутим, постоје многи задаци који се решавају наредбама гранања које не потпадају ни у једну од наведених категорија. Илуструјмо неке од њих.

Задатак: Сутрашњи датум

Напиши програм који за унети датум одређује датум наредног дана.

Опис улаза

Са стандардног улаза се уносе три позитивна цела броја (сваки у засебном реду) која представљају дан, месец и годину једног исправног датума.

Опис излаза

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	*Излаз*
1 1 2016	2.1.2016.

Пример 2

Улаз	*Излаз*
28 2 2016	29.2.2016.

Пример 3

Улаз	*Излаз*
28 2 1900	1.3.1900.

Пример 4

Улаз	*Излаз*
31 12 2017	1.1.2018.

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.

```
#include <iostream>

using namespace std;

// 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;
}

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

int main() {
  // 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;
      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;

  return 0;
}
```

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 кôд, али се губи информација о данашњем дану након одређивања сутрашњег, што је лоше.

```
int main() {
  // ucitavamo 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;

  return 0;
}
```

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).

Опис улаза

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).

Опис излаза

Једна линија стандарног 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	*Излаз*
35 23 92 37 52 0	`tupougli`

Решење

Три угла могу бити углови троугла ако ако су сви већи од нуле и ако је њихов збир 180 степени. Ако услов постојања троугла није испуњен (негација претходног услова тражи да је неки од углова мањи или једнак нули или ако је збир углова у троуглу различит од 180 степени) треба на стандардном излазу исписати реч `ne`. У супротном потребно је извршти анализу величине углова како би утврдили врсту троугла (ако постоји угао већи од 90 степени троугао је тупоугли, ако постоји угао од 90 степени троугао је правоугли, а иначе је оштроугли).

Пошто су углови троугла задати у облику степени и минута, ради лакшег извођења аритметичких операција и релација са њима пожељно је све их превести у углове задате само у минутима.

Услов да постоји угао већи од 90 степени можемо изразити тако што проверимо да ли је било који од три угла већи од 90 степени (везавши услове оператором дисјункције тј. логичким везником *или*).

```
#include <iostream>

using namespace std;

// pretvara ugao dat u stepenima(s) i minutima (m) u
// ugao samo u minutima
int uMinute(int s, int m) {
  return s * 60  + m;
}

int main() {
  // ucitavamo uglove u stepenima i minutima
  int ugao1_s, ugao1_m, ugao2_s, ugao2_m, ugao3_s, ugao3_m;
  cin >> ugao1_s >> ugao1_m;
  cin >> ugao2_s >> ugao2_m;
  cin >> 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);
```

```
  // uglovi trougla ne smeju biti 0 i zbir im mora biti 180 stepeni
  if (ugao1 == uMinute(0, 0) || ugao2 == uMinute(0, 0) || ugao3 == uMinute(0, 0) ||
      ugao1 + ugao2 + ugao3 != uMinute(180, 0))
    cout << "ne" << endl;
  else { // jesu uglovi trougla
    // prav ugao u minutima
    int pravUgao = uMinute(90, 0);
    // ako je bar jedan ugao tup, trougao je tupougli
    if (ugao1 > pravUgao || ugao2 > pravUgao || ugao3 > pravUgao)
      cout << "tupougli" << endl;
    // u suprotnom, ako je bar jedan ugao prav, trougao je pravougli
    else if (ugao1 == pravUgao || ugao2 == pravUgao || ugao3 == pravUgao)
      cout << "pravougli" << endl;
    // u suprotnom nema pravih ni tupih uglova, pa je trougao ostrougli
    else
      cout << "ostrougli" << endl;
  }

  return 0;
}
```

Уместо да проверавамо све углове, можем да пронађемо највећи од три угла и само њега поредимо са 90 степени (сви бројеви су мањи од неке вредности ако и само ако је највећи од њих мањи од те вредности).

```
#include <iostream>
#include <algorithm>

using namespace std;

// pretvara ugao dat u stepenima(s) i minutima (m) u
// ugao samo u minutima
int uMinute(int s, int m) {
  return s * 60  + m;
}

int main() {
  // ucitavamo uglove u stepenima i minutima
  int ugao1_s, ugao1_m, ugao2_s, ugao2_m, ugao3_s, ugao3_m;
  cin >> ugao1_s >> ugao1_m;
  cin >> ugao2_s >> ugao2_m;
  cin >> 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);

  // uglovi trougla ne smeju biti 0 i zbir im mora biti 180 stepeni
  if (ugao1 == uMinute(0, 0) || ugao2 == uMinute(0, 0) ||
      ugao3 == uMinute(0, 0) ||
      ugao1 + ugao2 + ugao3 != uMinute(180, 0))
    cout << "ne" << endl;
  else { // jesu uglovi trougla
    // 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)
```

```
      cout << "tupougli" << endl;
    // ako je najveci ugao prav, trougao je pravougli
    else if (maxUgao == pravUgao)
      cout << "pravougli" << endl;
    // u suprotnom je najveci ugao ostar, pa su svi uglovi ostri
    else
      cout << "ostrougli" << endl;
  }

  return 0;
}
```

Задатак: Тенис

Напиши програм који на основу броја освојених гемова два играча одређује исход сета у тенису (сет није последњи и при резултату 6:6 се игра тај-брејк).

Опис улаза

Са стандардног улаза се уносе два природна броја (раздвојена размаком) који представљају број освојених гемова сваког играча редом.

Опис излаза

Ако је први играч освојио сет на стандардни излаз исписати поруку `pobedio prvi`. Ако је други играч освојио сет исписати `pobedio drugi`. Ако унети резултат неисправан исписати `neispravno`. Ако сет још није завршен исписати `nije zavrseno`.

Пример 1

Улаз	*Излаз*
`7 5`	`pobedio prvi`

Пример 2

Улаз	*Излаз*
`3 7`	`neispravno`

Решење

Треба да проверимо да ли је први играч победио другог, а затим и да ли је други играч победио првог. Пошто су ове две провере заправо идентичне, погодно је дефинисати функцију којом проверавамо да ли је играч који је освојио a гемова победио играча који је освојио b гемова, а онда да је позовемо тако што проследимо број освојених гемова првог и другог играча, а затим број гемова другог и првог играча. Дефиниција те функције је једноставна (играч са освојених a гемова је победио ако и само ако је освојио 6 гемова, док је онај други освојио највише 4 гема или је освојио 7 гемова, док је онај други освојио 5 или 6 гемова).

Провера исправности резултата је мало компликованија. И њу ћемо имплементирати у посебној функцији. Једноставности ради, можемо уредити два броја освојених гемова, тако да знамо који је од та два броја мањи, а који је већи. Најлакши начин да се то уради је да се употребе библиотечке функције за одређивање минимума и максимума два броја. У језику C++ то су функције `min` и `max` декларисане у заглављу `<algorithm>`. Нико не може да има више од 7 освојених гемова, тако да ако је већи број већи од 7 функција одмах може да врати `false` (чиме се констатује да је резултат неисправан). Ако је већи освојио 7 гемова, мањи је морао да освоји или 6 или 5 (ако се то десило функција може да врати `true`, а у супротном вредност `false`). У супротном су оба играча освојила највише 6 гемова и у том случају је сваки резултат исправан (ако је мањи освојио највише 4 гема, сет је завршен, а ако је освојио 5 или 6, још се игра), и функција може да врати `true`.

```
#include <iostream>
#include <algorithm>

using namespace std;

// da li je igrac a pobedio igraca b
bool pobedio(int a, int b) {
  return (a == 6 && b <= 4) || // rezultati 6:4, 6:3, 6:2, 6:1 i 6:0
         (a == 7 && (b == 5 || b == 6)); // rezultati 7:6 i 7:5
}

// da li je trenutni rezultat a:b ispravan?
```

```
bool ispravno(int a, int b) {
  // odredjujemo veci i manji broj osvojenih gemova
  int veci = max(a, b), manji = min(a, b);
  // niko ne sme imati vise od 7 osvojenih gemova
  if (veci > 7) return false;
  // ako je neko osvojio 7 gemova, onaj drugi mora imati 5 ili 6
  if (veci == 7) return manji == 5 || manji == 6;
  // znamo da je veci osvojio <= 6 gemova, pa je toliko osvojio i
  // manji - sve takve kombinacije rezultata su ispravne
  return true;
}

int main() {
  int prvi, drugi;
  cin >> prvi >> drugi;
  if (pobedio(prvi, drugi))
    cout << "pobedio prvi" << endl;
  else if (pobedio(drugi, prvi))
    cout << "pobedio drugi" << endl;
  else if (ispravno(prvi, drugi))
    cout << "nije zavrseno" << endl;
  else
    cout << "neispravno" << endl;
  return 0;
}
```

Још једна могућност је да дефинишемо функцију која проверава да ли је сет незавршен. Опет је погодно сортирати два броја освојених бодова и затим проверити да ли је већи број мањи од 6 или је већи број једнак 6, а мањи број је већи или једнак 5.

Пошто победа првог или другог играча осигурава да је резултат исправан, исправност експлицитно треба проверавати тек када установимо да нико још није победио. Дакле, у главном програму проверавамо редом да ли је први победио, у супротном да ли је други победио, у супротном да ли је резултат неисправан и у супротном закључујемо да сет још није завршен. За ово је потребно употребити конструкцију `else-if` (ако бисмо услове испитивали независно могли бисмо, на пример, добити истовремено и поруку да је први победио и да је резултат исправан).

```
#include <iostream>
#include <algorithm>

using namespace std;

// da li je igrac a pobedio igraca b
bool pobedio(int a, int b) {
  return a == 6 && b <= 4 || // rezultati 6:4, 6:3, 6:2, 6:1 i 6:0
         a == 7 && (b == 5 || b == 6); // rezultati 7:6 i 7:5
}

// proveravamo da li se pri rezultatu a:b jos igra set
bool josSeIgra(int a, int b) {
  // odredjujemo veci i manji broj osvojenih gemova
  int veci = max(a, b), manji = min(a, b);
  // proveravamo da li se jos igra
  return veci < 6 || (veci == 6 && manji >= 5);
}

int main() {
  int prvi, drugi;
  cin >> prvi >> drugi;
  if (pobedio(prvi, drugi))
```

```
    cout << "pobedio prvi" << endl;
  else if (pobedio(drugi, prvi))
    cout << "pobedio drugi" << endl;
  else if (josSeIgra(prvi, drugi))
    cout << "nije zavrseno" << endl;
  else
    cout << "neispravno" << endl;
  return 0;
}
```

Наравно, задатак је могуће решити и без дефинисања помоћних функција.

```
#include <iostream>

using namespace std;

int main() {
  int prvi, drugi;
  cin >> prvi >> drugi;

  if (prvi > 7 || drugi > 7 ||
      (prvi == 7 && drugi < 5) || (drugi == 7 && prvi < 5))
    cout << "neispravno" << endl;
  else if (prvi > drugi && ((prvi == 6 && drugi < 5) || prvi == 7))
    cout << "pobedio prvi" << endl;
  else if (drugi > prvi && ((drugi == 6 && prvi < 5) || drugi == 7))
    cout << "pobedio drugi" << endl;
  else
    cout << "nije zavrseno" << endl;
  return 0;
}
```

2.8 Минимум и максимум два броја

Многи задаци се могу решити ако се употребе функције за одређивање минимума и/или максимума два броја (у језику C++ то су функције `min` и `max` декларисане у заглављу `<algorithm>`). На тај начин се често добијају програми који су много једноставнији него када се користи експлицитно гранање (на овај начин, гранање је скривено унутар самих функција `min` и `max`).

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).

Опис улаза

Са стандардног улаза се учитавају 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.

Опис излаза

На стандардни излаз исписати један цео број који представља најмањи број потеза потребан краљу да са полазног стигне до долазног поља.

Пример 1

Улаз	*Излаз*
`1 2` `3 4`	`2`

Пример 2

Улаз	*Излаз*
`8 3` `2 5`	`6`

Решење

Нека су (x_1, y_1) координате полазног поља, а (x_2, y_2) координате долазног поља. Растојање по првој координати једнако је $|x_2 - x_1|$, а по другој координати једнако је $|y_1 - y_2|$. Да би краљ стигао на жељено поље, оба ова растојања морају бити смањена на нулу. У сваком кораку (било хоризонталном, било вертикалном, било дијагоналном) свака од координата се може променити највише за један (у дијагоналном потезу истовремено се мењају обе координате, али свака за по један). Зато се у сваком од потеза хоризонтално и вертикално растојање могу смањити за један. У почетку краљ може ићи дијагонално истовремено смањујући оба растојања за по један, све док једно од растојања не постане једнако нули (док не стигне у врсту или колону у којој се налази циљно поље). Након тога може се кретати хоризонтално тј. вертикално до циља.

Дакле, број потеза потребан да се стигне до циљаног поља је једнак већем од растојања по координатама x и y тј. једнако је вредности $\max(|x_2 - x_1|, |y_2 - y_1|)$. Максимум два броја се може израчунати било гранањем, било библиотечком функцијом. Апсолутну вредност је могуће израчунати коришћењем функције `abs` која је декларисана у заглављу `<cmath>`[1].

```
#include <iostream>
#include <cmath>
#include <algorithm>

using namespace std;

int main() {
  // ucitavamo koordinate dva polja
  int x1, y1, x2, y2;
  cin >> x1 >> y1 >> x2 >> y2;
  // odredjujemo i ispisujemo najmanji broj poteza kralja
  cout << max(abs(x1 - x2), abs(y1 - y2)) << endl;
  return 0;
}
```

Задатак: Такси промо-кôд

Корисник је добио промо-кôд са којим плаћа вожњу таксијем 500 динара мање, али цена коју плаћа не сме бити нижа од цене поласка која је 300 динара. Написати програм који за унету цену вожње без попуста одређује колико корисник треба да плати након урачунавања овог попуста.

Опис улаза

Са стандардног улаза се уноси цена без попуста.

Опис излаза

На стандардни излаз се исписује цена са попустом.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
1300	800	600	300

Решење

Нека је цена вожње без попуста c. Ако је вредност $c - 500$ мања од 300 динара, цена после попуста је 300, а у супротном је цена после попуста $c - 500$. Дакле, цена после попуста је максимум вредности 300 и $c - 500$.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int cena_bez_popusta;
  cin >> cena_bez_popusta;
  cout << max(300, cena_bez_popusta - 500) << endl;
```

[1] Наслеђе програмског језика C је то да је варијанта ове функције за целе бројеве декларисана у заглављу `<cstdlib>`.

```
  return 0;
}
```

Задатак се може решити и експлицитним гранањем.

```
#include <iostream>

using namespace std;

int main() {
  int cena_bez_popusta;
  cin >> cena_bez_popusta;
  if (cena_bez_popusta > 800)
    cout << cena_bez_popusta - 500 << endl;
  else
    cout << 300 << endl;
  return 0;
}
```

Задатак: Темена правоугаоника

Написати програм који одређује координате горњег левог и доњег десног угла правоугаоника чије су странице паралелне са координатним осама и његову ширину и дужину ако су познате координате нека његова два наспрамна темена (не зна се која). Напомена: у рачунарској графици y-координате расту од врха ка дну екрана, па горња темена имају мање y-координате него доња.

Опис улаза

Са стандардног улаза се уносе координате једног темена правоугаоника (два цела броја раздвојена размаком), а затим и координате њему наспрамног темена.

Опис излаза

На стандардни излаз исписати координате горњег левог, а затим координате доњег десног темена тог правоугаоника.

Пример

Улаз	*Излаз*
100 50	40 50
40 70	100 70

Решење

Једно од два наспрамна темена се налази на левој, а једно на десно ивици правоугаоника. Самим тим су познате x-координате тих ивица. Мања од те две координате је x-координата леве, а већа је x-координата десне ивице правоугаоника. Слично, мања од две y-координате наспрамних темена је y-координата горње, а већа од њих је y-координата доње ивице правоугаоника. Горње лево теме се налази у пресеку леве и горње ивице правоугаоника, а доње десно у пресеку доње и десне ивице правоугаоника.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int x1, y1, x2, y2;
  cin >> x1 >> y1 >> x2 >> y2;
  cout << min(x1, x2) << " " << min(y1, y2) << endl;
  cout << max(x1, x2) << " " << max(y1, y2) << endl;
  return 0;
}
```

Задатак: Разврставање студената

У једној Excel табели налазе се подаци о m студената првог и n студената другог тока, а у другој о p студената првог и q студената другог тока. Колико је најмање података потребно преместити из једне у другу табелу да би се у једној табели налазили само подаци о студентима првог, а у другој само о студентима другог тока?

Опис улаза

Са стандардног улаза се учитавају ненегативни цели бројеви m, n, p и q.

Опис излаза

На стандардни излаз исписати најмањи број података које треба преместити.

Пример

Улаз	*Излаз*
123 58 49 83	107

Објашњење

Најбоље је да податке о 58 студената другог тока пребацимо у другу, а о 49 студената првог тока у прву табелу.

Решење

Можемо или да све студенте првог тока пребацимо у прву, а студенте другог тока у другу табелу или да све студенте другог тока пребацимо у прву, а студенте првог тока у другу табелу. У првом случају треба да пребацимо n студената из прве у другу табелу и p студената из друге у прву табелу. У другом случају треба да пребацимо m студената из прве у другу табелу и q студената из друге у прву табелу. Зато је најмањи број премештања минимум бројева $n + p$ и $m + q$.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int m, n, p, q;
  cin >> m >> n >> p >> q;
  cout << min(n+p, m+q) << endl;
  return 0;
}
```

Уместо функције за одређивање минимума, задатак се може решити и експлицитном применом гранања.

```
#include <iostream>

using namespace std;

int main() {
  int m, n, p, q;
  cin >> m >> n >> p >> q;
  if (n+p < m+q)
    cout << n+p << endl;
  else
    cout << m+q << endl;
  return 0;
}
```

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).

Опис улаза

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 $a_2 < b_2$.

Опис излаза

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 `presek ne postoji` ако се интервали не секу и у трећем реду се исписује дужина уније.

Пример 1

Улаз	*Излаз*
1	1.00 4.00
3	2.00 3.00
2	3.00
4	

Пример 2

Улаз	*Излаз*
0.5	-2.50 4.50
4.5	presek ne postoji
-2.5	5.00
-1.5	

Решење

Почетак покривача два интервала је мањи од два лева краја та два интервала (тј. $a_{pokrivac} = \min(a_1, a_2)$), а крај покривача је већи од два десна краја (тј. $b_{pokrivac} = \max(b_1, b_2)$). Напоменимо и да покривач увек постоји.

Почетак евентуалног пресека је већи од два лева краја (тј. $a_{presek} = \max(a_1, a_2)$), а крај евентуалног пресека је мањи од два десна краја (тј. $b_{presek} = \min(b_1, b_2)$). Пресек постоји ако и само ако је $b_{presek} > a_{presek}$.

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).

```
#include <iostream>
#include <iomanip>
#include <algorithm>

using namespace std;

int main() {
  // krajnje tacke intervala [a1, b1] i intervala [a2, b2]
  double a1, b1, a2, b2;
  cin >> a1 >> b1 >> a2 >> b2;

  // 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 presek 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 << "presek ne postoji" << endl;
    duzinaPreseka = 0.0;
  }

  // odredjujemo i ispisujemo duzinu unije intervala
  double duzina1 = b1 - a1, duzina2 = b2 - a2;
  double duzinaUnije = duzina1 + duzina2 - duzinaPreseka;
  cout << fixed << showpoint << setprecision(2)
```

```
      << duzinaUnije << endl;

  return 0;
}
```

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.

Опис улаза

Са стандардног улаза учитавају се два цела броја a и b ($-10^5 \leq a \leq b \leq 10^5$).

Опис излаза

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`	`5`	`2` `5`	`3`	`-3` `-1`	`0`

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.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  // 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;
  return 0;
}
```

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 $\max(b, 0) - \max(a, 0)$.

Задатак: Пресек правоугаоника

Одредити површину пресека два правоугаоника чије су странице паралелне координатним осама. Правоугаоници су задати са по два наспрамна темена.

Опис улаза

Са стандардног улаза учитава се 8 целих бројева iz intervala $[-100, 100]$.

- x_1^1, y_1^1 - једно теме првог правоугаоника
- x_2^1, y_2^1 - њему наспрамно теме
- x_1^2, y_1^2 - једно теме другог правоугаоника
- x_2^2, y_2^2 - њему наспрамно теме

Координате сваке тачке су наведене у посебном реду, раздвојене са тачно једним размаком.

Опис излаза

На стандардни излаз исписати један цео број - површину пресека та два правоугаоника.

Пример

Улаз	*Излаз*
-5 -5 2 2 0 0 3 3	4

Решење

Пресек два правоугаоника чије су ивице паралелне координатним осама је или нови такав правоугаоник (укључујући и дегенерисане случајеве дужи и тачке) или празан скуп. Пројекција правоугаоника на било коју од координатних оса је дуж (интервал реалне праве) чије координате темена су одговарајуће координате сваког од наспрамних темена правоугаоника. Кључна опаска за решење овог задатка је да се дужина странице пресечног правоугаоника може добити израчунавањем дужине пресека дужи добијених пројекцијом правоугаоника на одговарајућу координатну осу (пројекција странице пресечног правоугаоника је пресек пројекија страница полазних правоугаоника). Зато је кључни елемент решења функција која израчунава дужину пресека две дужи на координатним осама (два интервала реалне праве) чије је решење приказано у задатку Интервали.

```
#include <iostream>
#include <algorithm>

using namespace std;

// duzina preseka dva intervala [a1, b1] i [a2, b2]
// pri cemu ai ne mora da bude uvek manje i jednako od bi
int presekIntervala(int a1, int b1, int a2, int b2) {
  // levi i desni kraj prvog intervala
  int l1 = min(a1, b1), d1 = max(a1, b1);
  // levi i desni kraj drugog intervala
  int l2 = min(a2, b2), d2 = max(a2, b2);
  // drugi levi kraj
  int l = max(l1, l2);
  // prvi desni kraj
  int d = min(d1, d2);
  // presek je interval [l, d] - prazan ako je l > d
  return max(d-l, 0);
}

int main() {
  // koordinate dva naspramna temena prvog pravougaonika
  int x11, y11, x12, y12;
  cin >> x11 >> y11 >> x12 >> y12;
  // koordinate dva naspramna temena drugog pravougaonika
  int x21, y21, x22, y22;
  cin >> x21 >> y21 >> x22 >> y22;

  // racunamo povrsinu preseka dva pravougaonika
  int ax = presekIntervala(x11, x12, x21, x22);
  int by = presekIntervala(y11, y12, y21, y22);
  cout << ax * by << endl;

  return 0;
}
```

Задатак: Део квадра у првом октанту

Дат је квадар чије су ивице паралелне координатним осама. Одредити запремину дела тог квадра који припада првом октанту (део простора у коме су све координате позитивне).

Опис улаза

Са стандардног улаза учитава се 6 целих бројева који представљају редом 3 координате једног темена, затим 3 координате другог темена исте просторне дијагонале квадра.

Опис излаза

На стандардни излаз исписује се један цео број - запремина дела квадра у првом октанту.

Пример 1

Улаз	*Излаз*
-2 4 1 3 2 7	36

Пример 2

Улаз	*Излаз*
9 2 -5 1 6 -2	0

Решење

Задатак можемо решити тако што одредимо дужину дела једне горње ивице, дужину дела једне десне ивице и дужину дела једне предње ивице квадра, који припадају првом октанту (запремина дела квадра у првом октанту је онда производ те три дужине). То се своди на проналажење дела дужи на x-оси, тј. y-оси тј. z-оси који припадају позитивном делу те осе, што је приказано 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, односно координате z_1 и z_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, z1, x2, y2, z2;
  cin >> x1 >> y1 >> z1 >> x2 >> y2 >> z2;

  int ax = pozitivanDeoDuzi(x1, x2);
  int by = pozitivanDeoDuzi(y1, y2);
  int cz = pozitivanDeoDuzi(z1, z2);
  cout << ax * by * cz << endl;
  return 0;
}
```

Глава 3

Итерација

3.1 Основни итеративни алгоритми над малим серијама елемената

Задатак: Три трансакције

Са стандардног улаза се уносе подаци о три новчане трансакције. Позитивни износи представљају уплате, а негативни исплате. Написати програм који одређује укупан уплаћени и укупан исплаћени износ.

Опис улаза

Са стандардног улаза се учитавају 3 реална броја из интервала $[-1\,000, +1\,000]$, различитих од нуле.

Опис излаза

У први ред стандардног излаза исписати укупан износ уплата, а у други ред укупан износ исплата, заокружен на две децимале.

Пример

Улаз	*Излаз*
384.25	384.25
-14.2	-24.60
-10.4	

Решење

Задатак је најједноставније решити итеративним поступком тако што ћемо увести променљиве које чувају збир уплата и збир исплата. Ове променљиве ћемо иницијализовати на нулу, а затим ћемо учитавати једну по једну трансакцију, проверавати да ли је у питању уплата или исплата и ако је уплата, увећаваћемо вредност променљиве која чува збир уплата, а ако је исплата увећаваћемо вредност променљиве која чува збир исплата. Након обраде свих трансакција, исписаћемо вредности ових променљивих.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
  double zbirUplata = 0.0, zbirIsplata = 0.0;

  double transakcija;

  cin >> transakcija;
  if (transakcija > 0.0)
    zbirUplata += transakcija;
  else
    zbirIsplata += (-transakcija);
```

```
  cin >> transakcija;
  if (transakcija > 0.0)
    zbirUplata += transakcija;
  else
    zbirIsplata += (-transakcija);

  cin >> transakcija;
  if (transakcija > 0.0)
    zbirUplata += transakcija;
  else
    zbirIsplata += (-transakcija);

  cout << fixed << showpoint << setprecision(2)
       << zbirUplata << endl
       << zbirIsplata << endl;

  return 0;
}
```

Задатак: Разлика између најмање и највеће цифре

Написати програм који исписује разлику између најмање и највеће цифре унетог четвороцифреног броја.

Опис улаза

Са стандардног улаза се уноси четвороцифрени број.

Опис излаза

На стандардни излаз исписати тражену разлику.

Пример

Улаз	*Излаз*
4312	3

Решење

Цифу броја n тежине 10^k можемо одредити као $\lfloor n \rfloor 10^i \bmod 10$. Када одредимо све 4 цифре броја, одредићемо најмању и највећу. Опишимо поступак одређивања најмање (поступак одређивања највеће тече аналогно). Претпостављамо да је цифра јединица најмања тако што променљиву која ће на крају садржати најмању цифру ажурирамо на цифру јединица. Затим је поредимо са цифром десетица и ако је она мања, ажурирамо вредност те променљиве. Затим поредимо вредност те променљиве са цифром стотина и ако је цифра стотина мања, ажурирамо вредност те променљиве. На крају, исто радимо и за цифру хиљада.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;

  // odredjujemo cifre broja
  int c0 = (n / 1) % 10;
  int c1 = (n / 10) % 10;
  int c2 = (n / 100) % 10;
  int c3 = (n / 1000) % 10;

  // odredjujemo najmanju cifru
  int minCifra = c0;
  if (c1 < minCifra)
    minCifra = c1;
  if (c2 < minCifra)
```

```
    minCifra = c2;
  if (c3 < minCifra)
    minCifra = c3;

  // odredjujemo najvecu cifru
  int maksCifra = c0;
  if (c1 > maksCifra)
    maksCifra = c1;
  if (c2 > maksCifra)
    maksCifra = c2;
  if (c3 > maksCifra)
    maksCifra = c3;

  cout << maksCifra - minCifra << endl;
  return 0;
}
```

Цифре броја је могуће одредити и итеративним поступком. У сваком кораку наредну цифру одређујемо као остатак текуће вредности броја при дељењу са 10 (то је цифра јединица), а затим број мењањмо тако што из њега уклањамо цифру јединица целбројним дељењем 10.

Минимум можемо одредити и тако што га иницијализујемо на вредност цифре јединица, а онда га у сваком кораку постављамо на мању од текуће вредности минимума и нове цифре (мању од две вредности можемо израчунати и библиотечком функцијом `min`).

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int n;
  cin >> n;
  int c0 = n % 10; n = n / 10;
  int c1 = n % 10; n = n / 10;
  int c2 = n % 10; n = n / 10;
  int c3 = n % 10; n = n / 10;

  int minCifra = c0;
  minCifra = min(minCifra, c1);
  minCifra = min(minCifra, c2);
  minCifra = min(minCifra, c3);

  int maksCifra = c0;
  maksCifra = max(maksCifra, c1);
  maksCifra = max(maksCifra, c2);
  maksCifra = max(maksCifra, c3);

  cout << maksCifra - minCifra << endl;
  return 0;
}
```

Минимум неколико променљивих можемо израчунати и помоћу израза `min({x1, ..., xk})`.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int n;
```

```
  cin >> n;

  // odredjujemo cifre broja
  int c0 = (n / 1) % 10;
  int c1 = (n / 10) % 10;
  int c2 = (n / 100) % 10;
  int c3 = (n / 1000) % 10;

  int minCifra = min({c0, c1, c2, c3});
  int maksCifra = max({c0, c1, c2, c3});

  cout << maksCifra - minCifra << endl;
  return 0;
}
```

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.

Опис улаза

Са стандардног улаза уносе се три цела броја из интервала од 50 до 5000 који представљају цене у динарима за три купљена производа.

Опис излаза

На стандардни излаз исписати један цео број који представља укупну снижену цену та три производа.

Пример

Улаз	*Излаз*
2499	6099
3599	
899	

Решење

Одређивање збира и минумума

Најједноставнији начин да израчунамо снижену цену је да од збира сва три производа одузмемо цену најјефтинијег од њих и да на то додамо један динар.

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

int main() {
  int cena1, cena2, cena3;
  cin >> cena1 >> cena2 >> cena3;
  int ukupna_cena = cena1 + cena2 + cena3;
  int najmanja_cena = min({cena1, cena2, cena3});
  int snizena_ukupna_cena = ukupna_cena - najmanja_cena + 1;
  cout << snizena_ukupna_cena << endl;
  return 0;
}
```

Минимум три броја можемо одредити и угнежђеним гранањем, међутим, тако се добија компликован кôд, подложан грешкама и тај поступак се тешко уопштава на више од три броја.

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

int min3(int a, int b, int c)
```

```
{
  if (a <= b)
    if (a <= c)
      return a;
    else
      return c;
  else
    if (b <= c)
      return b;
    else
      return c;
}

int main() {
  int cena1, cena2, cena3;
  cin >> cena1 >> cena2 >> cena3;
  int ukupna_cena = cena1 + cena2 + cena3;
  int najmanja_cena = min3(cena1, cena2, cena3);
  int snizena_ukupna_cena = ukupna_cena - najmanja_cena + 1;
  cout << snizena_ukupna_cena << endl;
  return 0;
}
```

Сортирање

Други начин подразумева да сортирамо цене неопадајуће и да израчунамо збир износа једног динара и друге и треће цене по реду у сортираном редоследу.

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

int main() {
  int cena[3];
  cin >> cena[0] >> cena[1] >> cena[2];
  sort(begin(cena), end(cena));
  cout << 1 + cena[1] + cena[2] << endl;
  return 0;
}
```

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.

Опис улаза

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.

Опис излаза

На стандардном излазу исписати тражени редни број најтоплијег дана.

Пример

Улаз	*Излаз*
27 32 28 27 32 31 29	2

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. 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 najtoplijeg 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;
```

Задатак: Трећи угао троугла

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

Треба израчунати угао $180^\circ - (\alpha + \beta)$.

Угао $\alpha + \beta$ можемо израчунати применом алгоритма сабирања позиционих записа. Сабирамо секунде. Ако добијемо збир већи од 59, тада вршимо пренос једног минута (смањујемо број секунди за 60 и памтимо да

минуте треба увећати за 1). Затим сабирамо минуте и добијени пренос. Ако добијемо збир већи од 59, тада вршимо пренос једног степена (смањујемо број минута за 60 и памтимо да степене треба увећати за 1). На крају сабирамо степене урачунавајући и претходни пренос.

Добијени збир одузимамо од 180 степени. Одузимамо прво секунде. Пошто је умањеник константан, могли бисмо га унапред записати у облику $179°59'60''$, који обезбеђује да је могуће извршити одузимање без даљих позајмица, међутим, записаћемо га у облику $180°0'0''$ и применићемо општи алгоритам, који на исти начин ради какав год да су умањеник и умањилац. Крећемо од одузимања секунди. Ако је број секунди умањеника строго мањи од броја секунди умањиоца, позајмићемо један минут тако што ћемо број секунди умањеника повећати за 60, а број минута умањеника смањити за један (па и по цену да тај број постане -1). Када одузмемо секунде, прелазимо на одузимање минута. Поново, ако је број минута умањеника строго мањи од броја минути умањиоца, позајмићемо један степен тако што ћемо број минута умањеника повећати за 60, а број степени умањеника смањити за један. Након одузимања минута, прећи ћемо на одузимање степена. Под претпоставком да је умањеник био већи или једнак од умањиоца, степени ће моћи да се одузму без позајмица.

```
#include <iostream>

using namespace std;

int main() {
  int stepeni1, minuti1, sekunde1;
  cin >> stepeni1 >> minuti1 >> sekunde1;

  int stepeni2, minuti2, sekunde2;
  cin >> stepeni2 >> minuti2 >> sekunde2;

  // primenujemo algoritam sabiranja
  int sekundeZbir = sekunde1 + sekunde2;
  // prenos na narednu poziciju (ako su sekunde vece od 60, prece se 1 minut)
  int prenos = sekundeZbir / 60;
  // osigurava se da sekunde budu u rasponu [0, 60)
  sekundeZbir = sekundeZbir % 60;
  // uracunavamo i prenos sa prethodne pozicije
  int minutiZbir = minuti1 + minuti2 + prenos;
  // prenos na narednu poziciju (ako su minuti veci od 60, prece se 1 stepen)
  prenos = minutiZbir / 60;
  minutiZbir = minutiZbir % 60;
  // uracunavamo i prenos sa prethodne pozicije
  int stepeniZbir = stepeni1 + stepeni2 + prenos;

  // primenujemo algoritam oduzimanja
  int stepeni180 = 180, minuti180 = 0, sekunde180 = 0;
  // ako je potrebno, pozajmljujemo jedan minut
  if (sekunde180 < sekundeZbir) {
    sekunde180 += 60;
    minuti180 -= 1;
  }
  int sekunde3 = sekunde180 - sekundeZbir;

  // ako je potrebno, pozajmljujemo jedan stepen
  if (minuti180 < minutiZbir) {
    minuti180 += 60;
    stepeni180 -= 1;
  }
  int minuti3 = minuti180 - minutiZbir;

  int stepeni3 = stepeni180 - stepeniZbir;

  cout << stepeni3 << " " << minuti3 << " " << sekunde3 << endl;
```

```
  return 0;
}
```

3.2 Врсте петљи и њихова елементарна употреба

Задатак: Бројеви од a до b

Написати програме који исписују све целе бројеве из интервала $[a, b]$. У првом употребити петљу `for`, у другом `while`, а у трећем `do-while`.

Опис улаза

Са стандардног улаза се уносе цели бројеви a и b.

Опис излаза

На стандардни излаз исписати тражене бројеве, свеки у посебном реду.

Пример

Улаз	*Излаз*
3	3
5	4
	5

Решење

Први програм реализујемо петљом `for`.

```
#include <iostream>

using namespace std;

int main() {
  int a, b;
  cin >> a >> b;
  for (int i = a; i <= b; i++)
    cout << i << endl;
  return 0;
}
```

Други програм реализујемо петљом `while`.

```
#include <iostream>

using namespace std;

int main() {
  int a, b;
  cin >> a >> b;

  int i = a;
  while (i <= b) {
    cout << i << endl;
    i++;
  }

  return 0;
}
```

Приликом употребе петље `do-while` морамо бити обазриви, јер се њено тело увек извршава бар једном, што нзачи да ће она исписати број a, чак и када је строго већи од b и када не би требало исписати ни један број. Зато ову петљу морамо оградити и наредбом `if` којом услов проверавамо и пре извршавања петље.

```
#include <iostream>

using namespace std;

int main() {
  int a, b;
  cin >> a >> b;

  int i = a;
  if (i <= b)
    do {
      cout << i << endl;
      i++;
    } while (i <= b);

  return 0;
}
```

Задатак: Бројање у игри жмурке

У игри жмурке деца обично броје по пет $(5, 10, 15, 20, ...)$. Напиши програм који исписује баш те бројеве.

Опис улаза

Са стандардног улаза уноси се број x $(100 \leq x \leq 1000)$ дељив са 5.

Опис излаза

На стандардни излаз исписати бројеве дељиве са 5, почевши од 5 и завршивши са x. Сваки број исписати у посебном реду.

Пример

Улаз	*Излаз*
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`.

```
#include <iostream>

using namespace std;

int main() {
  int x;
  cin >> x;
  for (int i = 5; i <= x; i += 5)
    cout << i << endl;
  return 0;
}
```

Задатак: Степени двојке

Написати програм који исписује све степене двојке мање од дате границе.

Опис улаза

Са стандардног улаза се уноси природан број n ($1 \leq n \leq 2^30$).

Опис излаза

На стандардни излаз исписати све степене броја два који су мањи или једнаки од n, сваки у посебном реду.

Пример

Улаз	*Излаз*
32	1 2 4 8 16 32

Решење

Крећемо од броја 1 и у сваком кораку петље исписујемо текући број и затим га множимо са 2 (да бисмо од текућег добили наредни степен двојке). Пошто ово радимо све док је текући број мањи од n, можемо употребити петљу `while` (она каже *ради ДОК је услов испуњен*).

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  int stepen = 1;
  while (stepen <= n) {
    cout << stepen << endl;
    stepen *= 2;
  }
  return 0;
}
```

У језику C и његовим наследницима је уобичајено да се петља `for` употребљава када год је могуће природно идентификовати иницијализацију, услов и корак петље, што је у овом задатку случај.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  for (int stepen = 1; stepen <= n; stepen *= 2)
    cout << stepen << endl;
  return 0;
}
```

Задатак: Уоквиравање текста

Написати програм који унети текст уоквирује коришћењем карактера `-` и `|`.

Опис улаза

Са стандардног улаза се уноси ниска карактера (која не садржи преласке у нови ред).

Опис излаза

На стандардни излаз исписати уоквирену ниску која је учитана.

Пример

Улаз	*Излаз*

`programiranje`	`---------------`
	`|programiranje|`
	`---------------`

Решење

Пошто знамо дужину текста, знамо и број карактера - које треба исписати у првом и трећем реду (треба исписати два карактера више од дужине текста). Зато је програм најједноставније реализовати коришћењем бројачке петље `for`.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  string tekst;
  cin >> tekst;
  for (int i = 0; i < tekst.size() + 2; i++)
    cout << "-";
  cout << endl;
  cout << "|" << tekst << "|" << endl;
  for (int i = 0; i < tekst.size() + 2; i++)
    cout << "-";
  cout << endl;
  return 0;
}
```

Задатак: Уклањање крајњих нула

Написати програм који украња крајње нуле из учитаног природног броја.

Опис улаза

Са стандардног улаза се учитава позитиван цео број n ($n < 10^9$).

Опис излаза

На стандардни излаз исписати вредност броја која се добија уклањањем крајњих нула из његовог декадног записа.

Пример 1

Улаз	*Излаз*
`123000`	`123`

Пример 2

Улаз	*Излаз*
`1234`	`1234`

Решење

Опис главног решења

У овом блоку се описује главно решење задатка.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  while (n % 10 == 0)
    n /= 10;
  cout << n << endl;
```

```
  return 0;
}
```

Задатак: Цифре броја

Написати програм који исписује цифру по цифру ненегативног целог броја.

Опис улаза

Са стандардног улаза се учитава број n ($0 \leq n \leq 10^9$).

Опис излаза

На стандардни излаз исписати цифре броја n, сваку у посебном реду, од цифре јединице, ка цифрама веће тежине.

Пример

Улаз	*Излаз*
123	3
	2
	1

Решење

Цифру јединице можемо одредити као остатак у целобројном дељењу са 10, а уклонити је израчунавањем целобројног количника при дељењу са 10. Овај поступак треба понављати све док број не постане 0. Посебну пажњу треба обратити на случај $n = 0$. Да бисмо били сигурни да ће се исписати његова јединица цифра 0, уместо петље `while`, програм је пожељно организовати коришћење петље `do-while`.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  do {
    cifra = n % 10;
    n = n / 10;
    cout << cifra << endl;
  } while (n > 0);
   return 0;
}
```

Задатак: Број линија

Написати програм који одређује колико је линија стандардног улаза учитано док се није дошло до краја стандардног улаза. Тестирати програм и учитавањем података из неке датотеке коришћењем редирекције стандардног улаза.

Опис улаза

Са стандардног улаза се учитавају линије текста. Када се програм тестира интерактивно, на крају улаза је потребно унети симбол краја улаза (у систему Linux, он се на тастатури добија тастерима `ctrl+d`, а у систему Windows, он се на тастатури добија тастерима `ctrl+z`).

Опис излаза

На стандардни излаз исписати број учитаних линија.

Пример

Улаз	*Излаз*
zdravo svete	2
123	

Решење

За учитавање линије текста можемо користити функцију `getline`. Ако учитавање успе, њена повратна вредност се имплицитно конвертује у истинитосну вредност тачно, а ако не успе тј. ако се дошло до краја улаза, у вредност нетачно.

Бројање остварујемо итеративним алгоритмом. Бројач иницијално постављамо на нулу и увећавамо га за један у сваком кораку петље (након сваке успешно учитане линије).

```
#include <iostream>

using namespace std;

int main() {
  int brojLinija = 0;
  string linija;
  while (getline(cin, linija))
    brojLinija++;
  cout << brojLinija << endl;
  return 0;
}
```

Задатак: Број речи

Написати програм који броји унете речи све док се не унесе реч `kraj` или се не дође до краја улаза.

Опис улаза

Са стандардног улаза се уносе линије текста које садрже речи раздвојене размацима. Свака реч се састоји од малих слова енглеске абецеде.

Опис излаза

На стандардни излаз исписати број учитаних речи.

Пример 1

Улаз

```
zdravo svima
dobar dan svima
```

Излаз

```
5
```

Пример 2

Улаз

```
ovo je kraj
ovo je pocetak
```

Излаз

```
2
```

Објашњење

Две су речи унете пре него што је унета реч `kraj`.

Решење

Пошто речи не садрже размаке, већ само мала слова, речи можемо учитавати помоћу `cin >> rec`. Повратна вредност ове функције је таква да се имплицитно конвертује у истинитосну вредност тачно ако је учитавање успело тј. нетачно ако учитавање није успело (на пример, ако се дошло до краја стандардног улаза). Стога се петља којом се учитавају све речи може реализовати помоћу

```
while (cin >> rec) {
...
}
```

Када учитавамо реч проверавамо да ли је једнака `kraj` и ако јесте, тада можемо прекинути петљу наредбом `break`.

Бројање речи остварујемо тако што бројач иницијализујемо на нулу и увећавамо га сваки пут када учитамо реч која је различита од речи `kraj`.

```
#include <iostream>

using namespace std;
```

```
int main() {
  int brojReci = 0;
  string rec;
  while (cin >> rec) {
    if (rec == "kraj")
      break;
    brojReci++;
  }
  cout << brojReci << endl;
  return 0;
}
```

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.

Опис улаза

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 $a < b$.

Опис излаза

На стандардном излазу приказати редом тражене бројеве, заокружене на пет децимала, сваки у посебној линији.

Пример

Улаз	*Излаз*
`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 петљу 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 исписује и затим увећава за вредност dx.

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

int main() {
  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;
  }
  return 0;
}
```

Може се приметити да су тражени бројеви $a, a+dx, a+2\cdot dx, ..., a+(n-1)\cdot dx$, па се они могу приказивати као бројеви $a + i \cdot dx$ за све целе бројеве i од 0 до $n - 1$ (кажемо да *пресликавамо* бројеве од 0 до $n - 1$ функцијом $f(i) = a + i \cdot dx$.

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

int main() {
  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;
  return 0;
}
```

3.3 Пресликавање елемената серије

Задатак: Квадрати бројева од 1 до n

Написати програм који исписује све квадрате бројева од 1 до n.

Опис улаза

Са стандардног улаза се уноси природан број n ($1 \leq n \leq 1000$).

Опис излаза

На стандардни излаз исписати све квадрате бројева од 1 до n, сваки у посебном реду.

Пример

Улаз	*Излаз*
3	1
	4
	9

Решење

Помоћу петље `for` набрајамо све бројеве од 1 до n. У телу петље израчунавамо квадрат тренутне бројачке променљиве и исписујемо га.

```
#include <iostream>

using namespace std;

int main() {
```

```
  int n;
  cin >> n;
  for (int i = 0; i < n; i++)
    cout << i * i << endl;
  return 0;
}
```

Задатак: Табела Фаренхајт-Целзијус

Написати програм који за све целобројне вредности x из интервала $[a, b]$ одређује број степени Целзијуса које одговарају x степени Фаренхајта и број степени Фаренхајта који одговарају x степени Целзијуса.

Напомена: Фаренхајтова скала је дефинисана на основу тачке мржњења и топљења одређеног сланог раствора, тако да он мрзне на $0°F$, а кључа на $180°F$ степени Фарехнајта, а Целизијусова скала тако да вода мрзне на $0°C$, а кључа на $100°C$. Касније установљено да вода мрзне на $32°F$, а кључа на 180 степени више тј. на $212°F$, што је довољно за извођење веза између ове две скале.

Опис улаза

Са стандардног улаза се уносе цели бројеви a и b ($-200 \leq a < b \leq 200$).

Опис излаза

На стандардни излаз исписати тражене вредности, заокружене на по две децимале.

Пример

Улаз	*Излаз*
32	32 0.00 89.60
35	33 0.56 91.40
	34 1.11 93.20
	35 1.67 95.00

Решење

Веза између ових скала је линеарна и облика је $F = kC + n$. Важи да је $32 = k \cdot 0 + n$ и $212 = k \cdot 100 + n$. Зато је $n = 32$, а $k = 180/100 = 9/5$, па је $F = 9/5 \cdot C + 32$, а $C = 5/9 \cdot (F - 32)$.

Ове формуле примењујемо на свако x из интервала $[a, b]$ (користимо петљу `for` да бисмо све ове вредности набројали). Треба бити обазрив, јер добијене вредности не морају бити цели бројеви и треба осигурати да се у формулама користи реално дељење.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
  int a, b;
  cin >> a >> b;
  for (int x = a; x <= b; x++) {
    double F = 9.0/5.0*x + 32;
    double C = (x - 32) * 5.0/9.0;
    cout << x << " ";
    cout << fixed << showpoint << setprecision(2)
         << C << " " << F << endl;
  }
  return 0;
}
```

Задатак: Табелирање функције пређеног пута аутомобила

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 по-

требно је израчунати пређени пут аутомобила. *Напомена*: за равномерно убрзано кретање пређени пут након протеклог времена t изражава се са $s = v_0 \cdot t + \frac{a \cdot t^2}{2}$.

Опис улаза

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

Опис излаза

На стадардни излаз исписати серију бројева који представљају пређени пут у задатим тренуцима. Сваки број исписати заокружен на пет децимала.

Пример

Улаз	*Излаз*
1 1 2 0.5	0.00000 0.62500 1.50000 2.62500 4.00000

Решење

Потребно је у петљи обићи све тачке t интервала $[0, T]$. За свако време t израчунавамо пређени пут по формули $s = v_0 \cdot t + \frac{a \cdot t^2}{2}$ и исписујемо га.

Приметимо да се у овом задатку заправо врши табелирање једне (у овом случају квадратне) функције тј. пресликавање серије бројева.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
  // 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;
  }
  return 0;
}
```

Задатак: Табелирање синуса и косинуса

Написати програм који табелира функције $\sin(x)$ и $\cos(x)$ тј. израчунава и исписује њихове вредности у n равномерно распоређених (еквидистантних) тачака интервала $[0, 2\pi]$.

Опис улаза

Са стандардног улаза се учитава број n ($2 \leq n \leq 100$).

Опис излаза

За сваку од n тачака x исписати вредност x, $\sin(x)$ и $\cos(x)$, заокружене на по 5 децимала.

Пример

Улаз	*Излаз*

`5`	`0.00000 0.00000 1.00000`
	`1.57080 1.00000 0.00000`
	`3.14159 0.00000 -1.00000`
	`4.71239 -1.00000 -0.00000`
	`6.28319 -0.00000 1.00000`

Решење

Размак између тачака је $dx = \frac{2\pi}{n-1}$. У петљи пролазимо кроз n тачака интервала тако што обезбеђујемо да се петља изврши тачно n пута, крећемо од вредности $x = 0$ и у сваком кораку x увећавамо за dx. За сваку тачку x израчунавамо $\sin(x)$ и $\cos(x)$ и исписујемо их у траженом формату.

```
#include <iostream>
#include <iomanip>
#define _USE_MATH_DEFINES
#include <cmath>

using namespace std;

int main() {
  int n;
  cin >> n;
  double dx = 2*M_PI / (n-1);
  double x = 0;
  for (int i = 0; i < n; i++) {
    cout << fixed << showpoint << setprecision(5)
         << x << " " << sin(x) << " " << cos(x) <<  endl;
    x += dx;
  }
  return 0;
}
```

Задатак: Цезарова шифра

Један од најстаријих начина шифровања текста је Цезарова шифра у којој се сваки карактер мења карактером који је за k места даље од њега у абецеди (при чему се ово померање врши циклично, тј. након последњих карактера се прелази поново на прве). На пример, ако је $k = 3$, слово `a` се мења словом `d`, `b` словом `e`, `c` словом `f` итд., слово `x` се мења словом `a`, `y` словом `b`, а `z` словом `c`.

Опис улаза

Са стандардног улаза се уноси вредност помераја k ($1 \leq k \leq 25$), а затим једна реч која се састоји само од малих слова енглеске абецеде.

Опис излаза

На стандардни излаз исписати резултат шифровања те речи Цезаровом шифром.

Пример

Улаз	*Излаз*
`3` `abcxyzzdravo`	`defabccgudyr`

Решење

Да бисмо извршили шифровање користићемо операције над ASCII кодовима карактера. Карактер типа `char` се у језику C++ идентификује са његовим ASCII кодом и нема потребе вршити никакву посебну конверзију да би се од карактера добио његов ASCII код (податак типа `char` је заправо број којим се карактер представља). Редни број карактера у абецеди можемо добити тако што од ASCII кода тог карактера одузмемо ASCII код малог слова `a` (који се добија константом `'a'`). Редни број шифрованог карактера добијамо од тог редног

броја увећањем за k и одређивањем остатка при дељењу са 26 (да би се постигло циклично померање тј. да би се последњи карактери абецеде шифровали првим). Након одређивања ASCII кода шифрованог карактера, сам карактер се добија сабирањем са ASCII кодом слова `a` (који се добија константном `'a'`).

У главом програму учитавамо ниску, у петљи пролазимо кроз сваки њен карактер, шифрујемо га и исписујемо.

```
#include <iostream>

using namespace std;

int main() {
  int k;
  cin >> k;
  string rec;
  cin >> rec;
  for (char c : rec) {
    char cc = 'a' + (c - 'a' + k) % 26;
    cout << cc;
  }
  cout << endl;
  return 0;
}
```

3.4 Филтрирање серије, бројање елемената

Задатак: Одлични ученици

Написати програм који исписује имена и презимена свих одличних ученика са задатог списка ученика (ученик је одличан ако има просечну оцену бар $4{,}5$).

Опис улаза

Са стандардног улаза се учитава број ученика n ($1 \leq n \leq 100$). У сваком од n наредних редова налазе се подаци о неком ученику: име, презиме и просечна оцена (раздвојени са по једним размаком).

Опис излаза

На стандардни излаз исписати имена и презимена одличних ученика.

Пример

Улаз	*Излаз*
`4`	`Jovana Dragojevic`
`Jovana Dragojevic 4.92`	`Milena Zivadinovic`
`Selma Cimili 4.45`	
`Petar Jovanovic 3.82`	
`Milena Zivadinovic 5.00`	

Решење

У петљи чије се тело извршава n пута учитавамо име, презиме и просечну оцену. Ако просечна оцена већа или једнака од $4{,}5$, тада исписујемо име и презиме ученика.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    string ime, prezime;
    double prosecna_ocena;
```

```
    cin >> ime >> prezime >> prosecna_ocena;
    if (prosecna_ocena >= 4.50)
      cout << ime << " " << prezime << endl;
  }
  return 0;
}
```

Задатак: Triple-Double од 3 категорије

Кошаркаш Никола Јокић је познат по томе што често остварује "трипл дабл" тј. има двоцифрен број поена, скокова и асистенција. Ако је познат учинак кошаркаша у свакој утакмици сезоне у свакој од ове 3 категорије, написати на колико је утакмица остварен трипл-дабл.

Опис улаза

Са стандардног улаза се у првом реду уноси цео број N $(1 \leq N \leq 200)$, број утакмица у сезони на којима је неки играч играо. У сваком од следећих N редова је по 3 цела броја раздвојена по једним размаком: број поена, скокова, асистенција (ови бројеви нису већи од 200).

Опис излаза

На стандардни излаз исписати један цео број, број утакмица на којима је играч остварио трипл–дабл.

Пример

Улаз	*Излаз*
4 20 9 7 28 14 11 20 9 18 10 10 10	2

Решење

Потребно је пребројати редове улаза у којима су сва три броја двоцифрена. Текући број трипл-даблова чувамо у посебној бројачкој променљивој иницијализованој на нулу. У петљи учитавамо податке о једној по једној утакмици (број поена, скокова и асистенција) и ако су сва три двоцифрена, увећавамо бројач.

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

int main()
{
  int brojUtakmica;
  cin >> brojUtakmica;
  // prebrojavamo broj triple double utakmica
  int brojTriplDabl = 0;
  for (int utakmica = 0; utakmica < brojUtakmica; utakmica++) {
    int poena, skokova, asistencija;
    cin >> poena >> skokova >> asistencija;
    // tripl dabl je postignut ako su svi brojevi dvocifreni
    if (poena >= 10 && skokova >= 10 && asistencija >= 10)
      brojTriplDabl++;
  }
  cout << brojTriplDabl << endl;
  return 0;
}
```

Задатак: Број троцифрених бројева са растућим цифрама

Написати програм који одређује колико у интервалу $[a, b]$ постоји троцифрених бројева чије су цифре строго растуће.

Опис улаза

Са стандардног улаза се учитавају природни бројеви a и b.

Опис излаза

На стандардни излаз исписати тражени број троцифрених бројева.

Пример

Улаз	*Излаз*
1 350	54

Решење

Троцифрени бројеви се налазе у интервалу $[100, 999]$. Пресек тог интервала и интервала $[a, b]$ је интервал $[\max(a, 100), \min(b, 999)]$. У петљи ћемо за сваки број из тог интервала проверавати да ли су му цифре различите. Читљивости програма ради ту проверу можемо извршити у засебној функцији. Одређивање појединачних цифара троцифреног броја можемо извршити на уобичајени начин (целобројном дељењем) и затим их можемо упоредити.

Напомена: ако се провера не изврши у посебној функцији, потребно је обратити пажњу да се бројачка променљива у петљи не сме мењати у телу петље тј. да одређивање цифара не смемо извршити итеративним алгоритмом у ком ће се уклањати једна по једна цифра броја, здесна.

```
#include <iostream>
#include <algorithm>

using namespace std;

bool rastuceCifre(int n) {
  int c0 = n % 10;
  int c1 = (n / 10) % 10;
  int c2 = (n / 100) % 10;
  return c2 < c1 && c1 < c0;
}

int main() {
  int a, b;
  cin >> a >> b;
  int broj = 0;
  for (int n = max(a, 100); n <= min(b, 999); n++)
    if (rastuceCifre(n))
      broj++;
  cout << broj << endl;
  return 0;
}
```

Задатак: Бројање свих самогласника

Написати програм који броји појављивања свих самогласника у тексту који се уноси са стандардног улаза.

Опис улаза

Са стандардног улаза се уноси текст који се састоји само од ACII карактера, све док се не дође до краја стандардног улаза.

Напомена: Приликом интерактивног тестирања, крај стандардног улаза се уноси тастерима `ctrl+d` у оперативном систему Linux, односно `ctrl+z` у оперативном систему Windows. Тестирати програм и редирекцијом стандардног улаза тако да се подаци читају из неке датотеке.

Опис излаза

На стандардни излаз исписати редом број слова `a`, `e`, `i`, `o` и `u`. Не правити разлику између великих и малих слова (бројати их заједно).

Пример

Улаз

```
Ovo je primer teksta. Sa standardnog ulaza se unose linije.
Ovo je jos jedna linija.
I na kraju i treca.
```

Излаз

```
11 9 7 7 3
```

Решење

Учитаваћемо и обрађивати једну по једну линију стандардног улаза, све док не стигнемо до његовог краја. Линије се учитавају функцијом `getline`.

За бројање самогласника користимо 5 бројача (по један за сваки самогласник). Када учитамо линију у петљи пролазимо кроз све њене карактере и проверавамо да ли је у питању самогласник (на пример, наредбом `switch-case-break`) и ако јесте, повећавамо одговарајући бројач. На крају исписујемо вредности свих 5 бројача.

```
#include <iostream>

using namespace std;

int main() {
  int broj_a = 0, broj_e = 0, broj_i = 0, broj_o = 0, broj_u = 0;
  string linija;
  while (getline(cin, linija)) {
    for (char c : linija)
      switch(tolower(c)) {
      case 'a': broj_a++; break;
      case 'e': broj_e++; break;
      case 'i': broj_i++; break;
      case 'o': broj_o++; break;
      case 'u': broj_u++; break;
      }
  }
  cout << broj_a << " " << broj_e << " " << broj_i << " " << broj_o << " " << broj_u << endl;
  return 0;
}
```

Задатак: Број приступа у датом периоду

Познат је дневник приступа неком веб-сајту. За сваки приступ је позната је IP адреса са које се приступало, датум и време приступа. Написати програм који за унете датуме одређује колико је приступа сајту било у периоду између та два датума.

Опис улаза

Први ред стандардног улаза садржи датум почетка периода који нас занима, а други ред датум краја тог периода (датуми су дати у формату `dd/mm/gggg`). Трећи ред садржи број уноса n ($1 \leq n \leq 100$). Наредних n редова садржи IP адресу, датум и време приступа.

Опис излаза

За сваки приступ у датом периоду исписати IP адресу са које се приступило сајту. На крају исписати укупан број приступа у датом периоду.

Пример

Улаз	*Излаз*
01/07/2024 01/10/2024 5 192.168.1.1 12/02/2024 08:24 172.16.254.2 23/08/2023 17:35 203.0.113.15 15/09/2024 19:40 198.51.100.22 05/05/2025 03:00 10.0.0.5 30/09/2025 12:01	203.0.113.15 1

Решење

Основни подзадатак је проверити да ли се дати датум налази између два задата датума (времена се могу потпуно занемарити). У циљу те провере довољно је проверити да ли је почетни датум периода мањи или једнак од датума приступа ког анализирамо и да ли је датум тог приступа мањи или једнак од последњег датума периода. Једноставности ради можемо дефинисати структуру за представљање датума и функцију којом се пореде два задата датума (поређење је лексикографско, прво се пореди година, затим месец и на крају дан).

На почетку бројач иницијализујемо на нулу, затим у петљи учитавамо податке о свим уносима и за сваки унос на описани начин проверавамо да ли је у датом периоду. Ако јесте, исписујемо његову IP адресу (која се, једноставности ради, може представити ниском карактера) и увећавамо бројач. Након петље исписујемо крајњу вредност бројача.

```
#include <iostream>

using namespace std;

struct Datum {
  int dan, mesec, godina;
};

bool manjiIliJednak(Datum d1, Datum d2) {
  if (d1.godina < d2.godina)
    return true;
  if (d1.godina > d2.godina)
    return false;
  if (d1.mesec < d2.mesec)
    return true;
  if (d1.mesec > d2.mesec)
    return false;
  return d1.dan <= d2.dan;
}

int main() {
  char crtica;
  Datum pocetak;
  cin >> pocetak.dan >> crtica >> pocetak.mesec >> crtica >> pocetak.godina;
  Datum kraj;
  cin >> kraj.dan >> crtica >> kraj.mesec >> crtica >> kraj.godina;

  int n;
  cin >> n;
  int brojPristupa = 0;
  for (int i = 0; i < n; i++) {
    string ip;
    cin >> ip;
    Datum d;
    cin >> d.dan >> crtica >> d.mesec >> crtica >> d.godina;
```

```
    int sat, minut;
    cin >> sat >> crtica >> minut;
    if (manjiIliJednak(pocetak, d) && manjiIliJednak(d, kraj)) {
      cout << ip << endl;
      brojPristupa++;
    }
  }

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

Задатак: Triple-Double од 5 категорија

Кошаркаш Никола Јокић је познат по томе што често остварује "трипл дабл" тј. има двоцифрен број поена, скокова и асистенција. Трипл-дабл се може остварити и ако се оствари двоцифрен број блокада или украдених лопти (потребан је двоцифрен учинак у било којој од ових 5 категорија). Ако је познат учинак кошаркаша у свакој утакмици сезоне у свакој од ових 5 категорија, написати на колико је утакмица остварен трипл-дабл.

Опис улаза

Са стандардног улаза се у првом реду уноси цео број N $(1 \leq N \leq 200)$, број утакмица у сезони на којима је неки играч играо. У сваком од следећих N редова је по 5 целих бројева раздвојених по једним размаком: број поена, скокова, асистенција, блокада и украдених лопти редом (ови бројеви нису већи од 200).

Опис излаза

На стандардни излаз исписати један цео број, број утакмица на којима је играч остварио трипл–дабл.

Пример

Улаз	*Излаз*
3 20 9 7 2 1 127 12 11 3 0 0 10 11 14 12	2

Решење

У овом задатку треба пребројати редове улаза у којима су бар три од пет бројева већи од 9. Да бисмо установили да ли неки ред улаза треба бројати, треба у сваком реду пребројати елементе који су већи од 9. То значи да ћемо у овом задатку применити технику пребројавања "на два нивоа": глобално бројимо редове који испуњавају услов, а у сваком реду бројимо елементе веће од 9.

Можемо дефинисати функцију која на основу 5 појединачних статистика испитује да ли је постигнут трипл-дабл учинак.

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

bool TriplDabl(int poeni, int skokovi, int asistencije, int blokade, int ukradeneLopte) {
  int brojDvocifrenih = 0;
  if (poeni >= 10)
    brojDvocifrenih++;
  if (skokovi >= 10)
    brojDvocifrenih++;
  if (asistencije >= 10)
    brojDvocifrenih++;
  if (blokade >= 10)
    brojDvocifrenih++;
  if (ukradeneLopte >= 10)
    brojDvocifrenih++;
  return brojDvocifrenih >= 10;
}
```

```
int main()
{
  int brojUtakmica;
  cin >> brojUtakmica;
  // prebrojavamo broj triple double utakmica
  int brojTriplDabl = 0;
  for (int utakmica = 0; utakmica < brojUtakmica; utakmica++) {
    int poeni, skokovi, asistencije, blokade, ukradeneLopte;
    cin >> poeni >> skokovi >> asistencije >> blokade >> ukradeneLopte;
    if (TriplDabl(poeni, skokovi, asistencije, blokade, ukradeneLopte))
      brojTriplDabl++;
  }
  cout << brojTriplDabl << endl;
  return 0;
}
```

Једноставније решење је ако употребимо двоструку петљу: спољна петља ће ићи по утакмицама (тј. редовима улаза), а унутрашња по категоријама које се прате на свакој утакмици. Бројач трипл-даблова иницијализујемо пре спољне петље, а бројач двоцифрених учинака пре унутрашње.

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

int main()
{
  int brojUtakmica;
  cin >> brojUtakmica;
  // prebrojavamo broj triple double utakmica
  int brojTriplDabl = 0;
  for (int utakmica = 0; utakmica < brojUtakmica; utakmica++) {
    // prebrojavamo broj dvocifrenih rezultata u 5 kategorija
    int brojDvocifrenih = 0;
    for (int kategorija = 0; kategorija < 5; kategorija++) {
      int rezultat;
      cin >> rezultat;
      if (rezultat >= 10)
        brojDvocifrenih++;
    }

    // tripl dabl je postignut ako postoje bar 3 dvocifrena rezultata
    if (brojDvocifrenih >= 3)
      brojTriplDabl++;
  }
  cout << brojTriplDabl << endl;
  return 0;
}
```

3.5 Сабирање, множење, средине

Задатак: Закључна оцена

Написати програм који на основу досадашњих оцена ученика из неког предмета одређује његов просек оцена, оцену која би била закључена на основу тренутног просека и број петица које треба да добије да би имао закључену оцену 5.

Опис улаза

У првом реду стандардног улаза се налази тренутни број оцена n ($1 \leq n \leq 8$). У наредном реду се налазе оцене од 1 до 5, раздвојене са по једним размаком.

Опис излаза

На стандардни излаз исписати сваки тражени податак, редом, у посебном реду. Тренутни просек оцена исписати заокружен на две децимале.

Пример

Улаз	*Излаз*
2	3.50
2 5	4
	4

Решење

Просек оцена можемо израчунати тако што израчунамо збир свих оцена и поделимо га са бројем оцена n. Збир израчунавамо тако што га на почетку иницијализујемо на нулу, а затим у сваком кораку петље увећавамо за нову учитану оцену. Ако су и збир и број оцена целобројни, приликом израчунавања просека, пре дељења је потребно извршити експлицитну конверзију у реални тип.

Закључну оцену можемо израчунати гранањем, али можемо и тако што на тренутни просек додамо $0{,}55$ и заокружимо резултат наниже. Тиме обезбеђујемо да ће се за просек чији је разломљени део $0{,}5$ закључивати оцена која је за један већа од целог дела просека, а за остале просеке на цео део просека.

Означимо са n_5 број петица које треба добити да би закључна оцена била једнака 5. Нови збир оцена ће бити за $n_5 \cdot 5$ већи од старог збира, а нови број оцена ће за n_5 бити већи од старог броја n. Да би била закључена петица, нови просек треба да достигне вредност $4{,}5$. Одатле следи да број петица n_5 можемо израчунати на следећи начин.

$$\begin{aligned}
\frac{zbir + n_5 \cdot 5}{n + n_5} &\geq 4{,}5 \\
zbir + n_5 \cdot 5 &\geq 4{,}5 \cdot (n + n_5) \\
zbir + n_5 \cdot 5 &\geq 4{,}5 \cdot n + 4{,}5 \cdot n_5 \\
0{,}5 \cdot n_5 &\geq 4{,}5 \cdot n - zbir \\
n_5 &\geq \frac{4,5n - zbir}{0,5} \\
n_5 &= \left\lceil \frac{4,5n - zbir}{0,5} \right\rceil
\end{aligned}$$

```
#include <iostream>
#include <cmath>
#include <iomanip>

using namespace std;

int main() {
  int n;
  cin >> n;
  int zbir = 0;
  for (int i = 0; i < n; i++) {
    int ocena;
    cin >> ocena;
    zbir += ocena;
  }
  double prosek = (double)zbir/(double)n;
  cout << fixed << showpoint << setprecision(2) << prosek << endl;
  int zakljucna = (int)floor(prosek + 0.5);
  cout << zakljucna << endl;
  int brojPetica = max(0, (int)ceil((4.5*n - zbir) / 0.5));
  cout << brojPetica << endl;
  return 0;
}
```

Задатак: Средња брзина

Током свог кретања возило је први део пута прешло крећући се равномерно брзином v_1 током времена t_1, затим је други део пута прешло крећући се равномерно брзином v_2 током времена t_2 итд. Написати програм који на основу ових података током n деоница пута израчунава средњу брзину (она је једнака односу укупног пређеног пута и укупног времена).

Опис улаза

Са стандардног улаза се уноси број деоница n ($1 \leq n \leq 10$). Из наредних n редова уносе се брзине дате у $\frac{\text{km}}{\text{h}}$ и времена дата у h (у питању су реални бројеви).

Опис излаза

На стандардни излаз исписати средњу брзину у $\frac{\text{km}}{\text{h}}$.

Пример

Улаз	*Излаз*
3 110 0.5 100 0.8 80 1.2	92.40

Решење

Средња брзина је једнака количнику укупног пређеног пута и укупног времена. Укупно време се може израчунати сабирајући сва учитана времена (применом итеративног алгоритма сабирања, који иницијализује збир на нулу и увећава га за свако учитано време). Укупан пут се може израчунајући пређене путеве $s_i = v_i \cdot t_i$ на свакој појединачној деоници и сабирајући применог истог итеративног алгоритма сабирања.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
  int n;
  cin >> n;
  double sUk = 0.0;
  double tUk = 0.0;
  for (int i = 0; i < n; i++) {
    double vi, ti;
    cin >> vi >> ti;
    double si = vi * ti;
    tUk += ti;
    sUk += si;
  }
  double vSr = sUk / tUk;
  cout << fixed << showpoint << setprecision(2) << vSr << endl;
  return 0;
}
```

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.

Опис улаза

Са стандардног улаза уноси се n ($2 \leq n \leq 10$) позитивних реалних бројева: v_1, v_2, …, v_n (za svako v_i важи $30 \leq v_i \leq 120$), након чега следи крај улаза. Подаци се учитавају до краја улаза.

Опис излаза

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	*Излаз*
60	50.00
40	48.00

Решење

Просечна брзина представља однос укупног пређеног пута и укупног времена трајања пута.

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 $(v_1 + v_2 + ... + v_n) \cdot t$ километара. Укупно време је $n \cdot t$ сати. Зато је просечна брзина једнака

$$\frac{(v_1 + v_2 + ... + v_n) \cdot t}{n \cdot t} = \frac{v_1 + v_2 + ... + v_n}{n}.$$

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 $\frac{s}{v_1} + \frac{s}{v_2} + ... + \frac{s}{v_n}$. Дакле, просечна брзина једнака је

$$\frac{s \cdot n}{\frac{s}{v_1} + \frac{s}{v_2} + ... + \frac{s}{v_n}} = \frac{n}{\frac{1}{v_1} + \frac{1}{v_2} + ... + \frac{1}{v_n}}.$$

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.

Аритметичку средину израчунавамо тако што итеративним алгоритмом саберемо све брзине (иницијализујући збир на нулу и увећавајући га за једну по једну учитану брзину) и поделимо добијени збир нулом. У овом случају радимо са реалним бројевима, па не морамо да водимо рачуна о конверзији типова приликом дељења.

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.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
  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;
  return 0;
}
```

Задатак: Пораст цене акција

Акције једне компаније су се мењале током 12 месеци. Позната је почетна цена и проценат промене на крају сваког месеца, изражен као реалан број. На пример, број $0{,}02$ означава да се цена акција повећала за 2%, а $-0{,}01$ значи да се смањила за 1%. Написати програм који израчунава цену акције на крају године, као и укупан годишњи проценат промене.

Опис улаза

Први ред стандардног улаза садржи почетну цену акција (реалан број између 1 и 1000). Други ред садржи 12 реалних бројева који представљају месечне проценте промене.

Опис излаза

У први ред стандардног излаза исписати нову цену акција након годину дана, а у наредни годишњи проценат промене.

Пример 1

Улаз	*Излаз*
`100`	`126.82`
`0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02`	`0.27`

Објашњење

Током сваког месеца цена је расла 2%, тако да је након првог месеца цена била 102, након другог $104{,}04$ итд. Укупни пораст на крају године је око 27%.

Пример 2

Улаз

```
50
0.02 -0.01 0.02 -0.01 0.02 -0.01 0.02 -0.01 0.02 -0.01 0.02 -0.01
```

Излаз

```
53.01
0.06
```

Решење

Ако је p проценат промене, а цена акције C, тада је нова цена једнака $C(1+p)$. Зато је цена након годину дана једнака

$$C(1+p_1)\cdot\ldots\cdot(1+p_{12}).$$

Проценат промене једнак је

$$(1+p_1)\cdot\ldots\cdot(1+p_{12})-1.$$

Примењујемо алгоритам множења серије бројева. Приликом израчунавања цене можемо цену иницијализовати на почетну вредност, а затим је након сваког учитаног p_i множити са $1+p_i$. Проценат промене можемо иницијализовати на вредност 1 (то је неутрална вредност за операцију множења).

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
  double cena;
  cin >> cena;
  double P = 1.0;
  for (int i = 1; i <= 12; i++) {
    double p;
    cin >> p;
    cena *= (1 + p);
    P *= (1 + p);
  }
  cout << fixed << showpoint << setprecision(2)
       << cena << endl
       << P - 1 << endl;
  return 0;
}
```

3.6 Одређивање минимума и максимума

Задатак: Највиши кошаркаш

Кошаркаши једног тима носе редом дресове са бројевима од 1 до n. Ако је познатса висина сваког од њих, написати програм који дрес носи највиши од њих и колика му је висина (у случају да више кошаркаша има исту највећу висину, исписати најмањи њихов број дреса).

Опис улаза

Са стандардног улаза се уноси број кошаркаша n ($1 \leq n \leq 20$), а затим њихове висине у центиметрима (цели бројеви између 150 и 230).

Опис излаза

На стандардни излаз исписати тражени број дреса и висину.

Пример

Улаз	*Излаз*
5	3
189	201
194	
201	
199	
201	

Решење

Примењујемо алгоритам одређивања максимума и позиције максимума. Пошто су висине кошаркаша позитивни бројеви, иницијално вредност максимума можемо поставити на нулу (већ након учитавања прве висине кошаркаша, та вредност ће се променити). Учитавамо висину једног по једног кошаркаша и ако је она строго већа од текуће вредности максимума, тада мењамо и вредност максимума и вредност дреса (приметимо да до промене неће доћи ако је висина једнака текућем максимуму, па ће у случају играча исте висине предност имати онај чији је број дреса мањи).

```
#include <iostream>

using namespace std;

int main() {
  int n;
```

```
  cin >> n;
  int maxVisina = 0;
  int maxDres;
  for (int dres = 1; dres <= n; dres++) {
    int visina;
    cin >> visina;
    if (visina > maxVisina) {
      maxVisina = visina;
      maxDres = dres;
    }
  }
  cout << maxDres << endl;
  cout << maxVisina << endl;
  return 0;
}
```

Задатак: Најмањи круг

За дати низ тачака у равни одредити полупречник најмањег круга, са центром у координатном почетку, који садржи све дате тачке.

Опис улаза

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.

Опис излаза

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 2.6 3.4 1.2 6.7 3.33 5.55 2.57 3.44	6.80661

Решење

Решење задатка је полупречник круга (са центром у координатном почетку) који је једнак растојању најудаљеније учитане тачке од координатног почетка. Растојање најудаљеније тачке од координатног почетка се одређује стандардним алгоритмом за одређивање најмањег елемента серије бројева. За прву учитану тачку треба израчунати растојање од координатног почетка и то растојање прогласити максималним. Алтернативно, пошто су сва растојања ненегативна, иницијализацију можемо извршити и на нулу (она је најмања могућа вредност растојања). Учитавање координата тачака се наставља и одређује се њихово растојањем од координатног почетка, да би се кад год је то растојање веће од текућег максимума, максимум увећао на ту вредност. Растојање између две тачке рачуна се Питагорином теоремом.

```
// rastojanje od tacke (x1, y1) do tacke (x2, y2)
double rastojanje(double x1, double y1, double x2, double y2) {
  double dx = x1 - x2, dy = y1 - y2;
  return sqrt(dx*dx + dy*dy);
}

int main() {
  int n;                        // broj tacaka
  cin >> n;
  double cx = 0.0, cy = 0.0;    // koordinate centra kruga
  double max_rastojanje = 0.0;  // najvece rastojanje od centra kruga
  for (int i = 0; i < n; i++) {
```

```
    double x, y;    // ucitavaju se koordinate naredne tacke
    cin >> x >> y;
    // korekcija maksimuma ako rastojanje tacke (x, y) od centra vece
    // od tekuceg maksimuma
    double rast = rastojanje(x, y, cx, cy);
    if (rast > max_rastojanje)
      max_rastojanje = rast;
  }
  // poluprecnik kruga (najvece rastojanje od neke tacke
  // do centra kruga)
  double poluprecnik = max_rastojanje;
  // ispisuje se rezultat
  cout << fixed << showpoint << setprecision(5) << poluprecnik << endl;

  return 0;
}
```

Могуће је направити још једну додатну оптимизацију. Пошто је корена функција монотона (важи $\sqrt{x} > \sqrt{y}$ ако и само ако важи $x > y$), уместо поређења растојања, можемо поредити њихове квадрате, чиме се програм мало убрзава јер се избегава скупа операција израчунавања квадратног корена.

```
// kvadrat rastojanja od tacke (x1, y1) do tacke (x2, y2)
double kvadrat_rastojanja(double x1, double y1, double x2, double y2) {
  double dx = x1 - x2, dy = y1 - y2;
  return dx*dx + dy*dy;
}

int main() {
  int n;                      // broj tacaka
  cin >> n;
  double cx = 0.0, cy = 0.0;  // koordinate centra kruga
  double x, y;                // koordinate prve ucitane tacke
  cin >> x >> y;
  // najveci kvadrat rastojanja do centra zasada ucitanih tacaka
  double max_kvadrat_rastojanja = kvadrat_rastojanja(x, y, cx, cy);
  for (int i = 1; i < n; i++) {
    cin >> x >> y;  // ucitavaju se koordinate naredne tacke
    // korekcija maksimuma ako je kvadrat rastojanja tacke (x, y) od
    // centra vece od tekuceg maksimuma
    max_kvadrat_rastojanja = max(max_kvadrat_rastojanja,
                                 kvadrat_rastojanja(x, y, cx, cy));
  }
  // poluprecnik kruga (najvece rastojanje
  // od neke tacke do centra kruga)
  double poluprecnik = sqrt(max_kvadrat_rastojanja);
  // ispisuje se rezultat
  cout << fixed << showpoint << setprecision(5) << poluprecnik << endl;

  return 0;
}
```

Задатак: Најмлађи студент

Написати програм који у списку студената за које се знају имена, презимена и датуми рођења проналази оног ко је најмлађи.

Опис улаза

Са стандардног улаза се учитава број студената n ($1 \leq n \leq 100$), а затим из наредних n редова подаци о студентима (име, презиме и датум рођења у формату `dd/mm/gggg`). Сви студенти су рођени различитих дана.

Опис излаза

На стандардни излаз исписати име и презиме најмлађег студента.

Пример

Улаз	*Излаз*
3 Jovana Arsic 28/04/2002 Zoran Lukic 25/06/2001 Alma Jezdic 04/11/2002	Alma Jezdic

Решење

Примењујемо уобичајени алгоритам одређивања максималне вредности, међутим, овај пут користимо нетривијалан критеријум поређења. Пошто су у питању датуми, млађа особа је она чији је датум рођења каснији. Зато дефинишемо и користимо функцију која пореди два датума (проверавамо да ли је текући студент млађи од до тог тренутка најмлађег). Пошто су сви студенти по условима задатка рођени увелико после 1970, датум рођења најмлађег студента можемо иницијализовати на 1. 1. 1970.

```
#include <iostream>

using namespace std;

struct Datum {
  int dan, mesec, godina;
};

// provera da li datum dva ide posle datuma 1
// efikasnost prenosa datuma se moze popraviti
bool datumPosle(Datum datum1, Datum datum2) {
  if (datum1.godina < datum2.godina)
    return true;
  if (datum1.godina > datum2.godina)
    return false;
  if (datum1.mesec < datum2.mesec)
    return true;
  if (datum1.mesec > datum2.mesec)
    return false;
  return datum1.dan < datum2.dan;
}

int main() {
  int n;
  cin >> n;

  // nijedan student nije rodjen pre 1. 1. 1970.
  Datum datumNajmladji = {1, 1, 1970};
  string imePrezimeNajmladji;
  for (int i = 0; i < n; i++) {
    // ucitavamo podatke o novom studentu
    string ime, prezime;
    Datum datum;
    char crta;
    cin >> ime >> prezime
        >> datum.dan >> crta >> datum.mesec >> crta >> datum.godina;
    // proveravamo da li je on mladji od do sada najmladjeg
    if (datumPosle(datumNajmladji, datum)) {
      datumNajmladji = datum;
      imePrezimeNajmladji = ime + " " + prezime;
    }
  }
```

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

Задатак: Најближи рејтинг

Онлајн систем за играње допушта пријављеним корисницима да се такмиче једни против других. Познат је рејтинг сваког такмичара. Када играч A жели да започне партију, аутоматски се бира његов противник. Противник се бира са списка регистрованих играча, тако да има што сличнији рејтинг играчу A (играч не може играти сам против себе). Ако се рејтинг више играча подједнако разликује од рејтинга играча A, предност имају они играчи који имају мањи рејтинг од играча A. Ако има више таквих, бира се било који од њих.

Опис улаза

У првој линији стандардног улаза налазе се корисничко име и рејтинг играча A (рејтинг сваког играча је реалан број x ($0 \leq x \leq 20$). У другој линији налази се природан број n ($3 \leq n \leq 100$) који представља број регистрованих играча, док следећих n линија садрже податке о играчима (њихова корисничка имена и рејтинге)

Опис излаза

У првој линији стандардног излаза приказати корисничко име и рејтинг играча који је одабран да игра против играча A.

Пример

Улаз	*Излаз*

`nautilus 13.0`
`4`
`jovanaa 12.4`
`a324 8.4`
`mirjana.petrovic 13.6`
`nautilus 13.0`
`terminator07 7.5`
	`jovanaa 12.4`

Решење

Сличност два броја мери се њиховом апсолутном разликом. Два кандидата се пореде тако што се прво упореди разлика њихових рејтинга од рејтинга играча A, а ако су оне једнаке, онда се рејтинзи пореде по величини (предност има мањи). Дакле, играч са рејтингом a представља бољи избор од играча са рејтингом b ако и само ако важи $|a - x| < |b - x|$ или ако је $|a - x| = |b - x|$ и $a < b$.

Дакле, потребно је пронаћи минималну 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. Једина разлика је то што ћемо приликом поређења вредности тренутно учитаног броја са тренутном вредношћу минимума, уместо обичне примене релације `<` применити описани поредак рејтинга

Посебну пажњу треба обратити на услов да играч не сме да игра сам против себе. Зато када током одређивања минимума учитамо поново податке о играчу A, те податке треба прескочити. Унутар петље то можемо урадити наредбом `continue`. Међутим, може да се деси да је играч A први на списку, па ово прескакање морамо урадити и приликом иницијализације, пре петље.

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

int main() {
  // ucitavmao podatke o igracu A
  string igracA;
  cin >> igracA;
  double rejtingA;
  cin >> rejtingA;
```

```
  // broj registrovanih igraca
  int n;
  cin >> n;

  // redni broj igraca kojeg obradjujemo
  int i = 0;
  // ucitavamo podatke o prvom igracu
  string igrac;
  cin >> igrac;
  double rejting;
  cin >> rejting;
  i++;

  // ako je to igrac A, preskacemo ga
  if (igrac == igracA) {
    cin >> igrac;
    cin >> rejting;
    i++;
  }

  // prvog igraca proglasavamo za najblizeg igracu A
  string najbliziIgrac = igrac;
  double najbliziRejting = rejting;
  double najmanjaRazlika = abs(rejting - rejtingA);

  // obradjujemo ostale igrace
  for (; i < n; i++) {
    // ucitavamo podatke o narednom igracu
    cin >> igrac;
    cin >> rejting;
    // ako je u pitanju igrac A, preskacemo ga
    if (igrac == igracA)
      continue;

    //  razlika rejtinga trenutnog igraca i igraca A
    double razlika = abs(rejting - rejtingA);
    // ako je rejting blizi igracu A od trenutno najblizeg ili je
    // ili je razlika rejtinga jednaka, ali rejting manji
    if (razlika < najmanjaRazlika ||
        (razlika == najmanjaRazlika && rejting < najbliziRejting)) {
      // azuriramo podatke o protivniku
      najbliziRejting = rejting;
      najmanjaRazlika = razlika;
      najbliziIgrac = igrac;
    }
  }

  // prijavljujemo rezultat
  cout << najbliziIgrac << " " << najbliziRejting << endl;
  cout << najmanjaRazlika << endl;
  return 0;
}
```

Задатак: Број максималних

Испит је полагало n студената. За сваког студента је познат број поена које је освојио на испиту. Написати програм којим се одређује број студената који су постигли највећи освојен број поена на том испиту.

Опис улаза

Прва линија стандарног улаза садржи природан број n ($10 \leq n \leq 250$) који представља број студената. У наредних n линија налази се по један цео број из интервала $[0, 100]$, ти бројеви представљају поене које су студенти освојили.

Опис излаза

У првој линији стандарног излаза приказати укупан број студената који су постигли најбољи резултат на испиту.

Пример

Улаз	*Излаз*
5 57 90 68 90 73	2

Решење

У овом задатку комбинује се одређивање максимумума серије елемената и број појављивања неког елемента у серији.

Кључни увид који нам омогућава да задатак решимо само једним пролазом кроз елементе серије и да их не морамо памтити истовремено у меморији (у низу) 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.

```
#include <iostream>

using namespace std;

int main() {
  // broj studenata
  int n;
  cin >> n;
  // poeni tekuceg studenta
  int poeni;
  // ucitavamo poene prvog studenta
  cin >> poeni;
  // najveci broj poena od poena studenata 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 studente
  for (int i = 1; i < n; i++) {
    // ucitavamo poene narednog studenta
```

```
    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 studentima
      brojPojavljivanjaMaksimuma++;
  }
  // prijavljujemo rezultat
  cout << brojPojavljivanjaMaksimuma << endl;
  return 0;
}
```

Задатак: Други студент и други резултат

Појавили су се резултати испита и списак је сортиран по броју поена, опадајуће. Написати програм који одређује:

- број поена које је освојио студент који је први на списку (ранг листи),
- број поена које је освојио студент који је други на ранг листи,
- и други по реду освојени број поена (други резултат).

Опис улаза

Са стандардног улаза се учитава број студената n ($2 \leq n \leq 200$). Након тога се учитава број поена сваког студента.

Опис излаза

На стандардни излаз исписати тражене податке, сваки у посебном реду. Ако су сви студенти освојили исти број поена, онда други резултат не постоји и треба исписати -1.

Пример 1

Улаз	*Излаз*
8	90
90	90
80	80
90	
70	
75	
40	
90	
80	

Пример 2

Улаз	*Излаз*
3	95
95	95
95	-1
95	

Решење

Током извршавања алгоритма чуваћемо три променљиве:

- `prvi` – резултат првог такмичара на ранг листи
- `drugiNaRangListi` – резултат другог такмичара на ранг листи
- `drugiRezultat` – други по реду освојен број поена

Све променљиве ћемо пре петље иницијализовати на вредност $-\infty$ (пошто су поени између 0 и 100, уместо $-\infty$ можемо користити вредност -1).

Све време током извршавања алгоритма важиће однос `prvi` $\leq$ `drugiNaRangListi` $\leq$ `drugiRezultat`, при чему је вредност `drugiNaRangListi` увек или једнака вредности `prvi` (када постоји бар два студента са истим, мак-

смалним бројем поена) или вредности `drugiRezultat` (када постоји само један студент са максимални бројем поена).

Обрађујемо једног по једног студента. Када учитамо број поена наредног студента, анализирамо следеће могућности.

- Ако је број поена тог студента строго већи од броја поена до тада најбољег студента, тада тај до тада најбољи студент постаје други на ранг листи и његов резултат је други резултат, а текући студент постоје најбољи.

- У супротном, ако је број поена тог студента једнак броју до тада поена најбољег студента, онда њих двојица имају једнак, максимални број поена и то је уједно вредност поена прве двојице на на ранг листи. Други резултат по реду мора бити мањи од тог броја поена, па се он не мења.

- У супротном знамо да текући студент има мањи строго број поена од најбољег студента. Ако текући студент има већи број поена од другог студента на ранг листи, онда је он други студент на ранг листи и то је уједно други резултат (јер је строго мањи од резултата првог).

- У супротном је могуће и да други студент на ранг листи има исти број поена као и први. Тада проверавамо да ли је резултат текућег студента већи од другог резултата и ако јесте, ажурирамо (повећавамо) други резултат.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;

  int prvi = -1;
  int drugiRezultat = -1;
  int drugiNaRangListi = -1;

  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    if (x > prvi) {
      drugiNaRangListi = drugiRezultat = prvi;
      prvi = x;
    } else if (prvi == x) {
      drugiNaRangListi = prvi;
    } else if (x > drugiNaRangListi) {
      drugiNaRangListi = drugiRezultat = x;
    } else if (x > drugiRezultat) {
      drugiRezultat = x;
    }
  }

  cout << prvi << endl
       << drugiNaRangListi << endl
       << drugiRezultat << endl;

  return 0;
}
```

3.7 Линеарна претрага

Задатак: Одлични студенти

Одличним студентима ћемо сматрати оне чији је просек оцена бар 9,00. Написати програм који проверава да ли на датом списку постоји одличан студент.

Опис улаза

Са стандардног улаза се учитава број студената n ($1 \leq n \leq 20$), а затим подаци о n студената: име, презиме и просечна оцена.

Опис излаза

На стандардни излаз исписати `da` ако на списку постоји неки одличан студент тј. `ne` ако не постоји.

Пример

Улаз	*Излаз*

`3`
`Bojana Stevanovic 8.32`
`Djordje Petrovic 9.42`
`Sara Jelenkovic 6.52`
	`da`

Решење

Помоћу логичке променљиве пратићемо да ли смо у досадашњем прегледу списка наишли на неког одличног студента. Пошту у почетку нисмо видели ни једног, променљиву ћемо иницијализовати на вредност нетачно `false`. У петљи ћемо учитавати податке о једном по једном студенту. Ако текући студент има просечну оцену већу или једнаку од 9, тада постоји одличан студент, и вредност променљиве можемо поставити на `true`. Пошто ће резултат ће остати такав без обзира какви су наредни студенти, петљу можемо одмах прекинути (помоћу наредбе `break`).

```
#include <iostream>

using namespace std;

int main() {
  bool imaOdlicnih = false;
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    string ime, prezime;
    cin >> ime >> prezime;
    double prosek;
    cin >> prosek;
    if (prosek >= 9.0) {
      imaOdlicnih = true;
      break;
    }
  }
  if (imaOdlicnih)
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

Задатак: Речи без самогласника

Написати програм који одређује речи које се састоје само од сугласника, без употребе самогласника.

Опис улаза

Са стандардног улаза се уноси текст који се састоји само од речи српског језика записаних употребом само малих слова енглеске абецеде. Речи треба обрађивати све до краја стандардног улаза.

Опис излаза

На стандардни излаз исписати све речи из текста које су записане искључиво коришћењем сугласника (слова различитих од `a`, `e`, `i`, `o` и `u`).

Пример

Улаз	*Излаз*
`mrk trn se popeo na vrh`	`mrk`
`krv tece kroz vene`	`trn`
	`vrh`
	`krv`

Решење

У главном програму учитавамо реч по реч са стандардног улаза, све док се не стигне до краја стандардног улаза. За сваку реч алгоритмом линеарне претраге проверавамо да ли су сва њена слова сугласници. То је тачно у тренутку када нисмо видели још једно слово, па променљиву којом пратимо да ли су сви сугласници постављамо на тачно. Обрађујемо једно по једно слово и ако је самогласник, тада постављамо вредност те променљиве на нетачно (и одмах можемо прекинути петљу). На крају петље, ако је вредност променљиве остала тачно, реч се састоји само од сугласника, па је исписујемо.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  string rec;
  while (cin >> rec) {
    bool bezSamoglasnika = true;
    for (char c : rec) {
      if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {
        bezSamoglasnika = false;
        break;
      }
    }
    if (bezSamoglasnika)
      cout << rec << endl;
  }
  return 0;
}
```

Задатак: Све бинарне јединице

Написати програм који проверава да ли се бинарни запис броја састоји само од јединица.

Опис улаза

Са стандардног улаза се учитава природан број n ($1 \leq n \leq 10^9$).

Опис излаза

На стандардни излаз исписати `da` ако се бинарни запис броја састоји само од јединица, односно `ne` у супротном.

Пример 1

Улаз	*Излаз*
`9`	`ne`

Пример 2

Улаз	*Излаз*
`15`	`da`

Пример 3

Улаз	*Излаз*
`0`	`ne`

Решење

Задатак можемо решити комбинацијом итеративног алгоритма за одређивање бинарних цифара броја и алгоритма линеарне претраге којим проверавамо да ли сви елементи неке серије задовољавају дато својство (једнаки су 1).

У петљи `do-while` обрађујемо једну по једну цифру у бинарном запису броја. Прву цифру здесна у бинарном запису одређујемо као целобројни остатак при дељењу са 2, а бришемо је заменом броја његовим целобројним количником са 2. Ако је цифра различита од 1, тада променљиву којом пратимо да ли су све цифре једнаке 1, постављамо на вредност нетачно (и одмах можемо прекинути петљу).

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  bool sveJedinice = true;
  do {
    int cifra = n % 2;  n /= 2;
    if (cifra != 1) {
      sveJedinice = false;
      break;
    }
  } while (n > 0);

  if (sveJedinice)
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

Илустрације ради, приказујемо и варијанту у којој је провера изолована у посебну функцију. Захваљујући могућности да функција врати вредност и прекине извршавање током тела петље, није нам потребна помоћна логичка променљива.

У петљи унутар функције одређујемо једну по једну цифру. Ако је цифра различита од 1, тада се у запису не јављају искључиво јединице и функција може одмах да врати вредност нетачно (наредбом `return` се прекидају и петља и функција). У супротном се поступак наставља. Ако се петља заврши, то значи да није нађена ниједна цифра различита од јединице, па функција тада може да врати вредност тачно (ову вредност не можемо вратити у склопу тела петље, већ искључиво након петље, када су све цифре испитане).

```
#include <iostream>

using namespace std;

bool sveBinarneJedinice(int n) {
  do {
    int cifra = n % 2; n /= 2;
    if (cifra != 1)
      return false;
  } while (n > 0);
  return true;
}

int main() {
  int n;
  cin >> n;
  if (sveBinarneJedinice(n))
    cout << "da" << endl;
  else
```

```
    cout << "ne" << endl;
  i
  return 0;
}
```

Ефикасније решење задатка се може засновати на примени битовских операција. Ако број има све јединице у бинарном запису, његовим увећавањем за 1 се добија број који има једну цифру више, прва цифра му је 1, а остале су нула. Дакле, $n + 1$ се у свакој цифри разликује од броја n, па ће применом појединачних конјункција битова на свакој позицији (тзв. битовском конјункцијом) добити 0. Ако број нема све јединице, то се неће десити (на позицији прве нуле здесна у броју n у броју $n+1$ ће се налазити јединица). Дакле, услов да број има све јединице је да је `(n & (n+1)) == 0`. Треба посебну пажњу обратити на број 0, који задовољава овај услов, али није сачињен од свих јединица.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  if (n != 0 && (n & (n + 1)) == 0)
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

Задатак: Број јаких лозинки

Лозинка је јака ако има бар 8 карактера, бар једно мало, бар једно велико слово, бар једну цифру и бар један специјални карактер (карактер који није ни слово ни цифра). Напиши програм који одређује колико унетих лозинки је јако.

Опис улаза

Са стандардног улаза се учитава број лозинки n ($1 \leq n \leq 100$), а затим n лозинки (свака у посебном реду).

Опис излаза

На стандардни излаз исписати број јаких лозинки.

Пример

Улаз	*Излаз*
5	2
Zdravo123!	
3@abC	
petlja17	
GEJMERI_2024	
music_Shop-85	

Објашњење

Јаке су само прва и последња лозинка. Друга је прекратка, У трећој недостају велико слово и специјални карактер, а у трећој недостаје мало слово.

Решење

Можемо дефинисати посебну функцију којом се линеарном претрагом проверава да ли је лозинка јака. Пролазимо кроз карактере дате ниске и за сваки одређујемо да ли је мало слово, велико слово, цифра или ништа од тога. Користимо четири логичке промењиве којима региструјемо да ли се појавио неки крактер одговарајуће врсте. Њих на почетку постављамо на вредност нетачно, а ако се појави одговарајући карактер, мењамо им вредност на тачно. Лозинка је јака ако и само ако након проласка кроз све карактере све промењиве имају вредност тачно и ако је лозинка довољно дугачка (проверу дужине је могуће урадити и на почетку функције).

У главном програму учитавамо n лозинки, за сваку проверавамо да ли је јака и ако јесте увећавамо вредност бројача (који је на почетку иницијализован на нулу). Након обраде свих лозинки исписујемо вредност бројача.

```
#include <iostream>
#include <cctype>

using namespace std;

bool jakaLozinka(const string& lozinka) {
  if (lozinka.size() < 8)
    return false;
  bool malo = false, veliko = false, cifra = false, specijalni = false;
  for (char c : lozinka)
    if (islower(c)) malo = true;
    else if (isupper(c)) veliko = true;
    else if (isdigit(c)) cifra = true;
    else specijalni = true;
  return malo && veliko && cifra && specijalni;
}

int main() {
  int n;
  cin >> n;
  int brojJakih = 0;
  for (int i = 0; i < n; i++) {
    string lozinka;
    cin >> lozinka;
    if (jakaLozinka(lozinka))
      brojJakih++;
  }
  cout << brojJakih << endl;
  return 0;
}
```

Задатак: Први и последњи приступ

Дат је дневник приступа неком сајту који се састоји од IP адреса са којих је приступано сајту и времена приступа. Дневник је организован хронолошки, по временима присутпа. Написати програм који за дату IP адресу одређује прво и последње време приступа.

Опис улаза

Са стандардног улаза се у првој линији уноси IP адреса која нас занима (ниска карактера која садржи запис четири броја између 0 и 255, раздвојена тачком). У другој линији се налази укупан број n ($1 \leq n \leq 50000$) података о приступима, а затим се у наредних n линија налазе подаци о појединачним приступима: IP адреса и време (цео број између 0 и 10^9).

Опис излаза

На стандардни излаз исписати време првог и последњег приступа са дате IP адресе (свако у посебном реду) или текст `nema` ако се са те адресе није приступало сајту. Времена првог и последњег приступа могу бити једнака (ако је постојао само један приступ).

Пример

Улаз	*Излаз*
192.168.0.1 5 10.0.0.1 3882265 192.168.0.1 3882278 10.0.0.1 3882310 192.168.0.1 3882342 192.168.0.1 3882355	3882278 3882355

Решење

На почетку и прво и последње време приступал иницијализујемо на неку специјалну вредност (на пример, -1), која не може бити право време приступа и која означава да до тог тренутна није пронађен ниједан приступ сајту са разматране адресе. Учитавамо један по један елемент и ако се IP адреса поклапа, тада прву вредност ажурирамо само ако је тренутно једнака тој специјалној вредности, а последњу вредност ажурирамо свакако. Наравно, уместо ове специјалне вредности можемо и увести логичку променљиву која показује да ли је прва вредност постављена.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  string IP; cin >> IP;
  int n; cin >> n;
  int prvo = -1;
  int poslednje = -1;
  for (int i = 0; i < n; i++) {
    string IPPristupa; int vremePristupa;
    cin >> IPPristupa >> vremePristupa;
    if (IPPristupa == IP) {
      // ako do sada nije zabelezen pristup
      if (prvo == -1)
        prvo = vremePristupa;
      poslednje = vremePristupa;
    }
  }
  if (prvo != -1) {
    cout << prvo << endl;
    cout << poslednje << endl;
  } else {
    cout << "nema" << endl;
  }

  return 0;
}
```

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).

Опис улаза

Са стандардног улаза се уноси број бројева n ($1 \leq n \leq 20$), а након тога и сами бројеви, сваки у посебном реду.

Опис излаза

На стандардни излаз исписати `da` ако су бројеви уређени како је наведено тј. `ne` у супротном.

Пример

Улаз	*Излаз*
5 -3 -1 0 4 2	da

Решење

Током провере наш програм може бити у једном од три стања: читању негативног дела низа, читању нула и читању позитивног дела иза.

- Ако се у стању читања негативног дела низа појави нула прелази се у стање читање нула, а ако се појави неки позитивни број прелази се у стање читања позитивних бројева.
- Ако се у стању читања нула појави неки негативан број, улаз није исправан, а ако се појави неки позитиван број прелази се у стање читања позитивних бројева.
- Ако се у стању читања позитивних бројева појави било шта осим позитивног броја, улаз није исправан.

Стање се може представити једном целобројном променљивом и током читања сваког броја гранањем можемо одредити како се мења стање. Логичком променљивом можемо представити да ли је дошло до грешке и чим први пут дође до грешке, можемо прекинути петљу и пријавити да улаз није исправан.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  int NEGATIVNI = 0, NULE = 1, POZITIVNI = 2;
  int stanje = NEGATIVNI;
  bool OK = true;
  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    if (stanje == NEGATIVNI) {
      if (x == 0)
        stanje = NULE;
      else if (x > 0)
        stanje = POZITIVNI;
    } else if (stanje == NULE) {
      if (x < 0) {
        OK = false;
        break;
      } else if (x > 0) {
        stanje = POZITIVNI;
      }
    } else if (stanje == POZITIVNI) {
      if (x <= 0) {
        OK = false;
        break;
      }
    }
  }

  cout << (OK ? "da" : "ne") << endl;
  return 0;
}
```

Још једно могуће решење је да се читају редом елементи све док иду негативни, затим да се читају елементи све док иду нуле и на крају да се читају елементи све док иду позитивни (проверавајући да се у међувремену, случајно, није стигло до краја улаза). Улаз ће бити исправан ако и само ако су након ове три петље прочитани сви елементи са улаза. Током имплементације потребно је водити рачуна да се грешком не прочита више од n бројева.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int n;
  cin >> n;

  int i = 0;
  int x; cin >> x;
  while (i < n && x < 0) {
    i++;
    if (i < n) cin >> x;
  }
  while (i < n && x == 0) {
    i++;
    if (i < n) cin >> x;
  }
  while (i < n && x > 0) {
    i++;
    if (i < n) cin >> x;
  }

  if (i == n)
    cout << "da" << endl;
  else
    cout << "ne" << endl;
}
```

3.8 Угнежђене петље

Задатак: Таблица множења

Напиши програм који исписује таблицу множења.

Опис улаза

Са стандардног улаза се уносе два цела броја m и n ($1 \leq m, n \leq 9$), сваки у посебном реду.

Опис излаза

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

Решење

Све комбинације чинилаца се могу добити помоћу две угнежђене петље. У спољној петљи мења се први

чинилац од 1 до m, а у унутрашњој други чинилац од 1 до n. У телу унутрашње петље знају се вредности оба чиниоца i и j и тада се може израчунати и њихов производ $i \cdot j$ и форматирати испис. Прелазак у нови ред се врши на крају тела спољашње петље.

```
#include <iostream>

using namespace std;

int main() {
  int m, n;
  cin >> m >> n;
  for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++)
      cout << i << " * " << j << " = " << i*j << "\t";
    cout << endl;
  }
  return 0;
}
```

Задатак: Бројеви у датој основи

Бројеви у основи b се могу записати помоћу цифара $0, 1, \ldots, b-1$. Ако је основа већа од 10, тада се уместо цифара користе слова енглеске абецеде (цифра 10 записује се са a, цифра 11 са b итд.). Напиши програм који исписује све троцифрене бројеве у датој основи.

Опис улаза

Са стандардног улаза се учитава основа b ($2 \leq b \leq 16$).

Опис излаза

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.

- 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.

```
#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;
}
```

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.

Опис улаза

Са стандардног улаза се уноси број n ($2 \leq n \leq 20$) - највећи број јабука које сваки од другара може да има.

Опис излаза

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 су 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. Услов $d_1 \neq d_2$ има смисла проверити пре него што се уопште уђе у трећу петљу.

```
#include <iostream>

using namespace std;

int main() {
  // 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;
  return 0;
}
```

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.

Опис улаза

Са стандардног улаза се уноси број n ($4 \leq n \leq 20$).

Опис излаза

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).

```
#include <iostream>

using namespace std;

int main() {
  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;
  return 0;
}
```

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.

Опис улаза

У првој линији стандардног улаза налази се укупан број поена на кошаркашкој утакмици (природан број између 40 и 150).

Опис излаза

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	*Излаз*
6	0 0 6
	0 1 4
	0 2 2
	0 3 0
	1 0 3
	1 1 1
	2 0 0

Решење

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, 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).

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

int main() {
  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;
  return 0;
}
```

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 вредност за j.

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

int main() {
  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;
    }
  return 0;
}
```

Напоменимо и да није неопходно било израчунавати експлицитно горње границе, и итерација може бити контролисана оригиналним условима.

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

int main() {
  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;
    }
  return 0;
}
```

Задатак: Цикличне пермутације

Цикличним померањем за једно место улево низа бројева $x_1, x_2, x_3, ..., x_n$ добијамо $x_2, x_3, ..., x_n, x_1$, ако вршимо циклично померање за два места улево добијамо $x_3, ..., x_n, x_1, x_2$. 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, …, n редом за 0, 1, 2, …, $n-1$ места улево.

Опис улаза

Прва линија стандардног улаза садржи природан број $n \leq 30$.

Опис излаза

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 $i, i+1, ..., n, 1, 2, ..., i-1$.

Задатак можемо решити петљом у којој коришћењем бројачке промењиве 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 $i-1$.

```
#include <iostream>

using namespace std;

int main() {
  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;
  }
  return 0;
}
```

Приметимо да бројеве $i, i+1, ..., n, 1, 2, ..., i-1$ можемо приказати у једној петљи, увећавањем бројачке промењиве редом за 0, 1, …, $n-1$ и коришћењем остатка при дељењу са n (пошто петља креће од 1, а не од 0, од збира i и j потребно је одузети 1, пронаћи остатак, а затим додати 1.

```
#include <iostream>

using namespace std;

int main() {
  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;
  }
  return 0;
}
```

Такође, у унутрашњој петљи можемо водити посебну промењиву која чува вредност која се исписује. Ту промењиву 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.

```
#include <iostream>

using namespace std;

int main() {
  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;
  }
  return 0;
}
```

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 $1, 2, 3, \ldots, n, 2, 3, 4, \ldots, n, 3, 4, \ldots, n, \ldots, n-1, n, n$.

Опис улаза

Са стандардног улаза се учитава природан број n у границама од 1 до 30.

Опис излаза

Бројеви се исписују на стандардни излаз, раздвојени размацима, сваки суфикс у посебном реду.

Пример

Улаз	*Излаз*
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, 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.

```
#include <iostream>

using namespace std;

int main() {
  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;
  }
  return 0;
}
```

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.

Опис улаза

Са стандардног улаза се учитава цео број n ($1 \leq n \leq 20$).

Опис излаза

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 Сви суфикси низа бројева од 1 до n 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).

```
#include <iostream>
```

```
using namespace std;

int main() {
  int n;
  cin >> n;
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= i; j++)
      cout << j << " ";
    cout << endl;
  }
  return 0;
}
```

Задатак: Све подречи дужине n

Дата је реч и природан број n. Напиши програм који исписује све њене подречи (сегменте узастопних карактера) дужине n.

Опис улаза

Први ред стандардног улаза садржи реч састављену од малих слова енглеског алфабета, а други садржи број n.

Опис излаза

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 $i, i+1, \ldots i+n-1$. 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 $i+j$.

```
#include <iostream>

using namespace std;

int main() {
  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;
  }
  return 0;
}
```

Уместо унутрашње петље можемо искористити и готову функцију која издваја део ниске од позиције i и која има n карактера. У језику C++ можемо употребити методу `s.substr(i, n)`. У језику Пајтон 3 можемо употребити израз `s[i:i+n]`.

```
#include <iostream>

using namespace std;

int main() {
  string s;
  cin >> s;
  int n;
  cin >> n;
  for (int i = 0; i + n - 1 < s.size(); i++)
    cout << s.substr(i, n) << endl;
  return 0;
}
```

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 очитати изразом `s[i]`.

Опис улаза

Са стандардног улаза се учитава једна реч састављена само од малих слова енглеске абецеде.

Опис излаза

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;
  }
```

Када је фиксиран сегмент $[i, j]$ подреч можемо издвојити и коришћењем библиотечке 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: Све подречи по опадајућој дужини

За унету реч исписати све подречи у редоследу опадајуће дужине и растућих левих граница за речи исте дужине.

Опис улаза

Са стандардног улаза се уноси једна реч састављена само од малих слова енглеске абецеде.

Опис излаза

На стандардни излаз исписати тражене подречи, сваку у посебном реду.

Пример

Улаз	*Излаз*
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.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  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;
    }
  return 0;
}
```

У језику C++ можемо употребити методу `substr` у чијем позиву као аргументе наводимо почетну позицију i и дужину подречи која је једнака d.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  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++)
      // ispisujemo rec
      cout << s.substr(i, d) << endl;
  return 0;
}
```

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.

Опис улаза

Са стандардног улаза се учитава реч састављена од малих слова, не дужа од 100 карактера.

Опис излаза

На стандардни излаз исписати тражене цикличне пермутације те речи.

Пример

Улаз	*Излаз*
zdravo	zdravo
	dravoz
	ravozd
	avozdr
	vozdra
	ozdrav

Решење

У првом кораку исписујемо карактере ниске са позиција $0, 1, \ldots n-1$, у другом са позиција $1, 2, \ldots, n-1, 0$, итд. све до последњег корака у коме исписујемо карактере са позиција $n-1, 0, 1, \ldots n-2$. Ређање оваквих индекса је веома слично ономе приказаном у задатку Цикличне пермутације и може се остварити угнежђеним петљама, на било који од начина приказаних у том задатку.

Једна могућност је да у сваком кораку спољне петље имамо две посебне унутрашње петље.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
   string s;
   cin >> s;
   int n = s.size();
   for (int i = 0; i < n; i++) {
     for (int j = i; j < n; j++)
       cout << s[j];
     for (int j = 0; j < i; j++)
       cout << s[j];
     cout << endl;
   }
   return 0;
}
```

Једна могућност је да користимо модуларну аритметику и да у сваком кораку унутрашње петље приступамо карактеру на позицији $(i + j) \bmod n$.

```
#include <iostream>
#include <string>
```

```
using namespace std;

int main() {
   string s;
   cin >> s;
   int n = s.size();
   for (int i = 0; i < n; i++) {
     for (int j = 0; j  < n; j++)
       cout << s[(i + j) % n];
     cout << endl;
   }
   return 0;
}
```

Једна могућност је да користимо посебну променљиву за поцизију карактера koja у сваком кораку спољне петље креће од позиције i и чија се вредност ресетује на нулу када достигне n.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
   string s;
   cin >> s;
   int n = s.size();
   for (int i = 0; i < n; i++) {
     int p = i;
     for (int j = 0; j  < n; j++) {
       cout << s[p];
       p++;
       if (p == n)
         p = 0;
     }
     cout << endl;
   }
   return 0;
}
```

Прикажимо још једно специфично решење. Кôд доста можемо поједноставити ако ниску проширимо тако што је два пута поновимо. Тада можемо исписати све подречи дужине n тако проширене ниске (исто као у задатку Све подречи дужине n). На пример, ако је ниска `abc` тада се након проширивања добија `abcabc` и онда се исисују све подречи дужине 3 и добија се `abc`, `bca` и `cab`.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
   string s;
   cin >> s;
   int n = s.size();
   s = s + s;
   for (int i = 0; i < n; i++) {
     for (int j = 0; j < n; j++)
       cout << s[i + j];
     cout << endl;
   }
```

```
   return 0;
}
```

Задатак: Квадрат од звездица

Напиши програм који исцртава квадрат од карактера * (као што је приказано у примерима).

Опис улаза

Са стандардног улаза се учитава природан број n ($1 \leq n \leq 20$), који представља димензију квадрата (број звездица у свакој врсти и колони).

Опис излаза

На стандардни излаз исписати тражени цртеж.

Пример

Улаз	*Излаз*
4	****
	****
	****
	****

Решење

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 исписује један ред. То се врши тако што се у телу унутрашње петље која се извршава n пута исписује једна звездица, а након те унутрашње петље се исписује прелазак у нови ред.

```
#include <iostream>

using namespace std;

int main() {
  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
  }
  return 0;
}
```

Задатак: Ромб од звездица

Напиши програм који исписује ромб направљен од звездица (како је приказано у примеру).

Опис улаза

Са стандардног улаза се уноси природан број n ($3 \leq n \leq 20$), који представља димензију ромба (дужину његове странице тј. број редова које треба исписати).

Опис излаза

На стандардни излаз исцртати ромб.

Пример

Улаз	*Излаз*
5	*****
	*****
	*****
	*****
	*****

Решење

Ромб се састоји од n редова, а у сваком реду се прво исписује одређени број размака, а затим и n звездица. Претпоставимо да ћемо то остварити спољашњом петљом у којој i узима вредности од 0 до $n-1$. Обележимо за сваки ред вредност бројача i, број звездица и број размака.

```
          i   br  bz
    *****     0   4   5
   *****      1   3   5
  *****       2   2   5
 *****        3   1   5
*****         4   0   5
```

Можемо приметити да је у сваком реду збир броја размака и бројача i константан и једнак вредности $n-1$, па је стога број размака увек једнак $n-1-i$. Дакле у телу спољашње петље имаћемо унутрашњу петљу која исписује $n-1-i$ размака, а затим и другу унутрашњу петљу која исписује n звездица. На крају тела спољашње петље (након обе унутрашње) прелазимо у нови ред.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n-1-i; j++)
      cout << " ";
    for (int j = 0; j < n; j++)
      cout << "*";
    cout << endl;
  }
  return 0;
}
```

Унутрашње петље можемо издвојити у функцију која дати карактер исписује дати број пута.

```
#include <iostream>

using namespace std;

void ispisiKPuta(char c, int k) {
  for (int i = 0; i < k; i++)
    cout << c;
}

int main() {
  int n;
  cin >> n;
  for (int i = n-1; i >= 0; i--) {
    ispisiKPuta(' ', i);
    ispisiKPuta('*', n);
    cout << endl;
  }
  return 0;
}
```

Спољашњу петљу можемо реализовати тако да бројач i креће од $n-1$ и спушта се до 0 и тада би број размака био једнак i.

```
#include <iostream>
```

```
using namespace std;

int main() {
  int n;
  cin >> n;
  for (int i = n-1; i >= 0; i--) {
    for (int j = 0; j < i; j++)
      cout << " ";
    for (int j = 0; j < n; j++)
      cout << "*";
    cout << endl;
  }
  return 0;
}
```

Једно могуће решење је и да број размака пратимо помоћу посебне променљиве коју иницијализујемо на $n - 1$ и умањујемо је за 1 на крају тела унутрашње петље.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  for (int i = 0, r = n-1; i < n; i++, r--) {
    for (int j = 0; j < r; j++)
      cout << " ";
    for (int j = 0; j < n; j++)
      cout << "*";
    cout << endl;
  }
  return 0;
}
```

Задатак: Троугао од звездица

Напиши програм који исцртава троугао какав је приказан у примеру.

Опис улаза

Са стандардног улаза се учитава један природан број n ($1 \leq n \leq 30$), који представља висину троугла (број редова које цртеж садржи).

Опис излаза

На стандардни излаз се исписује тражени троугао, исписивањем карактера `*`, размака (карактера бланко) и преласка у нови ред. После сваке последње звездице у свакој врсти пређи у наредни ред (не исписивати размаке после звездица).

Пример

Улаз	*Излаз*

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 четвртом (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 елелемената -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 исписују размаци и једне у којој се исписују звездице).

```
#include <iostream>

using namespace std;

int main() {
  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;
  }
  return 0;
}
```

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.

```
#include <iostream>

using namespace std;

void ispisiNPuta(char c, int n) {
  for (int i = 0; i < n; i++)
    cout << c;
}

int main() {
  int n;
  cin >> n;
  int brojZvezdica = 1, brojRazmaka = n-1;
  for (int i = 0; i < n; i++) {
    ispisiNPuta(' ', brojRazmaka);
    ispisiNPuta('*', brojZvezdica);
    cout << endl;
    brojRazmaka--; brojZvezdica += 2;
  }
  return 0;
}
```

Број звездица и размака није неопходно експлицитно израчунавати, већ је могуће одржавати две посебне променљиве у којима се памте ови бројеви.

```
#include <iostream>

using namespace std;

int main() {
  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;
  }
  return 0;
}
```

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.

Опис улаза

Са стандардног улаза се учитава реч дужине између 3 и 20 карактера.

Опис излаза

На стандардни излаз исписати тражени троугао.

Пример

Улаз	*Излаз*

`bravo`	`  b`
	`  r r`
	`  a   a`
	`  v     v`
	`  ovarbravo`

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 $n-1$.

```
#include <iostream>
#include <string>

using namespace std;

void ispisiNPuta(char c, int n) {
  for (int i = 0; i < n; i++)
    cout << c;
}

int main() {
  string s;
  cin >> s;
  int n = s.size();
```

```
  // prvi red sadrzi samo jedno slovo
  ispisiNPuta(' ', n-1);
  cout << s[0] << endl;

  // srednji redovi sadrze po tacno dva slova
  for (int i = 1; i < n - 1; i++) {
    ispisiNPuta(' ', n-1-i);
    cout << s[i];
    ispisiNPuta(' ', 2*i-1);
    cout << s[i];
    cout << endl;
  }

  // poslednji red sadrzi 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;

  return 0;
}
```

3.9 Однос узастопних елемената, подсерије

Задатак: Сортирани датуми

Написати програм који проверава да ли је листа датума и времена сортирана у обратном хронолошком периоду (најновији подаци треба да буду први у листи).

Опис улаза

Са стандардног улаза се уноси број n ($1 \leq n \leq 100$). У наредних n редова се налазе временске одреднице у облику `dd/mm/yyyy hh/mm/ss` (датум, па време). Неки временски тренуци могу бити поновљени и више пута.

Опис излаза

На стандардни излаз исписати `da` ако су датуми и времена у обратном хронолошком редоследу тј. `ne` ако нису (ако има поновљених временских тренутака, они треба да буду један уз други).

Пример

Улаз	*Излаз*
4 09/11/2024 07:55:12 13/08/2024 13:24:08 13/08/2024 13:24:08 01/01/2023 09:55:42	da

Решење

Потребно је проверити да ли је серија временских тренутака сортирана тј. да ли су свака два узастопна тренутка таква да је други после првог. Најлакши начин да се тренуци упореде је да се сместе у торку и то редом, година, месец, дан, сат, минут секунд (торке се подразумевано пореде лексикографски). У сваком кораку алгоритма треба да знамо претходни и текући датум и да их упоредимо. Примењујемо алгоритам линеарне претраге и тражимо да ли постоји неки пар датума који су у погрешном редоследу (то можемо регистровати помоћу засебне логичке променљиве).

```
#include <iostream>
#include <tuple>

using namespace std;
```

```
typedef tuple<int, int, int, int, int, int> trenutak;

trenutak ucitajTrenutak() {
  int dan, mesec, godina, sat, minut, sekund;
  cin >> dan; cin.ignore();
  cin >> mesec; cin.ignore();
  cin >> godina;

  cin >> sat; cin.ignore();
  cin >> minut; cin.ignore();
  cin >> sekund; cin.ignore();
  return make_tuple(godina, mesec, dan, sat, minut, sekund);
}

int main() {
  int n;
  trenutak prethodni;
  prethodni = ucitajTrenutak();
  bool OK = true;
  for (int i = 1; i < n; i++) {
    trenutak tekuci = ucitajTrenutak();
    if (!(prethodni >= tekuci))
      OK = false;
    prethodni = tekuci;
  }

  if (OK)
    cout << "da" << endl;
  else
    cout << "ne" << endl;

  return 0;
}
```

Задатак: Распоред пакета на полицама

На полице у магацину се распоређују редом предмети. Полице се попуњавају редом, и то тако што на сваку полицу стављамо по K (унапред задат природан број) предмета. Написати програм који за сваку полицу исписује укупну масу предмета стављених на њу а затим и редни број полице на којој је укупна маса распоређених предмета највећа.

Опис улаза

У првој линији стандардног улаза налази се број предмета које стављамо на једну полицу K ($1 \leq K \leq 1000$). У наредној линији налазе се масе предмета m ($1 \leq m \leq 100$) које треба поставити на полицу која је на реду (предмета има највише 50000). Улаз се завршава масом 0 (на последњој позицији може бити мањи број предмета од K).

Опис излаза

На стандардном излазу се прво, за сваку полицу, у једној линији, налази укупна маса предмета стављених на њу (ти бројеви су раздвојени размацима), а затим у последњој линији и редни број полице са највећом укупном масом (полице се броје од 1).

Пример

Улаз	*Излаз*
`3`	`23 20 31`
`5 10 8 12 3 5 22 9 0`	`3`

Решење

Угнежђене петље

Један начин да задатак решимо је да употребимо угнежђене петље, тако што у спољној петљи обрађујемо полицу по полицу, док у унутрашњој попуњавамо сваку појединачну полицу.

Унутрашња петља има задатак да прочита податке о предметима на текућој полици и одреди њихов број, укупну масу и податак о томе да ли се током читања предмета дошло до краја улаза или није. Пре унутрашње петље (на почетку тела спољашње петље) број предмета и њихова укупна маса се иницијализују на нулу. Унутрашња петља се извршава све док се не учита K предмета или док се не учита ознака за крај (предмет масе 0). Након сваке учитане масе предмета, проверава се да ли она представља ознаку краја. Ако не, тада се број учитаних предмета увећава за 1, док се укупна маса увећава за масу тог предмета. У супротном се логичка променљива која означава да ли се дошло до краја улаза, а која је на почетку програма иницијализована на вредност `false`, поставља на вредност `true`. Та променљива се употребљава и у услову спољашње и у услову унутрашње петље, тако да ће њено постављање на `true` проузроковати да се обе петље прекину пре преласка на њихову наредну итерацију.

Спољашња петља има задатак да одреди вредност и позицију максимума у серији израчунатих укупних тежина предмета на полицама. Пре петље уводе се променљиве у којима се памти највећа до сада виђена маса на полици и редни број полице на којој се та маса налази и иницијализују се на вредност нула. Након изласка из унутрашње петље (без обзира на то да ли је прочитано K предмета или је прочитана ознака за крај), проверава се да ли је на тој полици било предмета (да ли је њихов број израчунат унутрашњом петљом већи од нуле) и ако јесте, да ли је маса предмета на тој полици већа од до тада највеће виђене масе. Ако јесте, ажурира се вредност максимума и редни број полице на којој се тај максимум налази (он се поставља на редни број текуће полице који се увећава за 1 у сваком кораку спољашње петље). При том, ако на текућој полици има предмета, исписује се њихова укупна маса.

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

int main() {
  // najveci broj predmeta na jednoj polici
  int K;
  cin >> K;

  // podaci o najtezoj do sada obradjenoj polici - njen redni broj i masa
  int brojNajtezePolice = 1, najvecaMasa = 0;

  // redni broj tekuce police
  int tekuciBrojPolice = 1;

  // logicki indikator koji odredjuje da li smo ucitali sve podatke o
  // predmetima
  bool kraj = false;
  while (!kraj) {
    // podaci o tekucoj polici masa i broj predmeta na njoj
    int tekucaMasa = 0;
    int tekuciBrojPredmeta = 0;

    // ucitavamo sve predmete na tekucoj polici
    while (tekuciBrojPredmeta < K && !kraj) {
      int masaPredmeta;
      cin >> masaPredmeta;
      // dosli smo do oznake kraja
      if (masaPredmeta == 0)
        kraj = true;
      else {
        tekuciBrojPredmeta++;
        tekucaMasa += masaPredmeta;
      }
    }
```

```
    // ako je masa predmeta na tekucoj polici veca od dosadasnjeg maksimuma,
    // azuriramo maksimum i njegovu poziciju
    if (tekucaMasa > najvecaMasa) {
      najvecaMasa = tekucaMasa;
      brojNajtezePolice = tekuciBrojPolice;
    }

    // ako ima predmeta na tekucoj polici, ispisujemo njenu masu
    if (tekuciBrojPredmeta > 0)
      cout << tekucaMasa << " ";

    // pripremamo se za obradu naredne police
    tekuciBrojPolice++;
  }
  cout << endl;

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

Једна петља

Други начин за решавање овог задатка избегава употребу угнежђених петљи. У једној петљи се учитава један по један предмет, проверавајући да ли је учитана ознака за крај. Ако јесте, логички индикатор краја, који је уједно и услов петље, се поставља на вредност `true`, чиме се проузрокује да се након завршетка извршавања тела петље она прекине. Ако није учитана ознака краја, већ маса неког предмета, број предмета на текућој полици се увећава за 1, а маса предмета на текућој полици се увећава за управо учитану масу. Након учитавања проверава се да ли је завршена обрада текуће полице тј. да ли је број предмета на њој једнак K или је учитана ознака за крај (што можемо одредити помоћу вредности логичког индикатора краја). Ако јесте, тада, ако на полици има предмета (тј. ако је број предмета на текућој полици већи од нуле), тада се исписује маса предмета на њој и она се упоређује са тежином до тада најтеже полице и ако је управо учитана полица тежа, ажурира се максимум (уједно ажурирајући и редни број полице). Након тога, ако се није дошло до краја, тада се врши припрема за обраду наредне полице тако што се за један увећа редни број полице, а текућа маса и број предмета на полици поставе на нулу.

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

int main() {
  // najveci broj predmeta na jednoj polici
  int K;
  cin >> K;

  // podaci o tekucoj polici - njen redni broj, broj predmeta na njoj
  // i njihova ukupna masa
  int tekuciBrojPolice = 1, tekuciBrojPredmeta = 0, tekucaMasa = 0;
  // podaci o najtezoj do sada obradjenoj polici - njen redni broj i masa
  int brojNajtezePolice = 1, najvecaMasa = 0;

  // indikator kojim se oznacava da li se doslo do kraja tj. da li je
  // ucitan predmet tezine 0
  bool kraj = false;
  while (!kraj) {
    // ucitavamo masu sledeceg predmeta
    int masaPredmeta;
    cin >> masaPredmeta;
    // proveravamo da li je 0, tj. da li se stiglo do kraja
```

```
    if (masaPredmeta == 0)
      kraj = true;
    else {
      // azuriramo podatke o tekucoj polici
      tekuciBrojPredmeta++;
      tekucaMasa += masaPredmeta;
    }

    // ako smo stigli do kraja tekuce police (ako smo ucitali K
    // predmeta ili predmet tezine 0)
    if (tekuciBrojPredmeta == K || kraj) {
      // ako na tekucoj polici ima predmeta
      if (tekuciBrojPredmeta > 0) {
        // ispisujemo masu police
        cout << tekucaMasa << " ";
        // ako je tekuca polica teza od najteze, azuriramo podatke o
        // najtezoj polici
        if (tekucaMasa > najvecaMasa) {
          najvecaMasa = tekucaMasa;
          brojNajtezePolice = tekuciBrojPolice;
        }
      }

      // ako ima potrebe za tim, pripremamo se za citanje naredne
      // police
      if (!kraj) {
        tekuciBrojPolice++;
        tekuciBrojPredmeta = 0;
        tekucaMasa = 0;
      }
    }
  }

  // ispisujemo redni broj najteze police
  cout << brojNajtezePolice << endl;
  return 0;
}
```

Задатак: Ледене недеље

Дате су максималне дневне температуре сваког дана у K узастопних недеља. Одредити колико има "ледених" недеља у којима је максимална температура сваког дана била негативна.

Опис улаза

У првој линији стандардног улаза налази се број недеља које анализирамо, K (природан број $3 \leq K \leq 100$). У свакој од наредних K линија налази се по седам максималних дневних температура током одговарајуће недеље (температуре су цели бројеви између -50 и 50).

Опис излаза

У једној линији стандардног излаза налази се број недеља у којима је максимална температура сваког дана била негативна (природан број, укључујући нулу).

Пример

Улаз	*Излаз*
2 2 3 2 -4 -3 0 1 -1 -2 -1 -1 -2 -3 -2	1

Решење

Задатак захтева да се серија улазних бројева подели на своје седмочлане подсерије и да се затим за сваку седмочлану подсерију провери да ли садржи само негативне бројеве (што се може урадити алгоритмом линеарне претраге).

Помоћна функција

Један од начина да имплементирамо решење овог задатка је да уведемо помоћну функцију чији ће задатак бити да провери да ли је недеља ледена, тј. да учита податке за једну недељу (свих седам бројева који представљају дневне температуре) и да линеарном претрагом провери да ли су све оне негативне.

Употребићемо помоћну логичку променљиву којом се региструје да ли су током извршавања функције сви дани "ледени" (тј. да ли су сви учитани бројеви негативни). На почетку ту променљиву инцијализујемо на `true`, а затим у петљи која се извршава тачно 7 пута (чија се бројачка променљива креће од 1 до 7), учитавамо наредну максималну дневну температуру, проверавамо да ли је негативна и ако није, логичкој променљивој додељујемо вредност `false`. На крају, након извршавања петље, враћамо вредност логичке променљиве.

Напоменимо да у овом случају није могуће извршити рани прекид петље (чим се наиђе на први дан који није леден), јер је неопходно да се за сваку недељу учитају подаци о свих њених 7 дана.

Још једно могуће решење било би да се изброји колико је дана у недељи било "ледено" и да се на крају провери да ли је тај број једнак 7 (што је еквивалентно услову да су сви дани "ледени").

У главном програму у петљи чије се тело извршава K пута, помоћу написане функције обрађујемо појединачне недеље и бројимо колико је недеља било "ледено" (опет је у питању бројање елемената серије који задовољавају неки услов).

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

bool LedenaNedelja() {
  bool ledena = true;
  for (int j = 0; j < 7; j++) {
    int dnevnaTemperatura;
    cin >> dnevnaTemperatura;
    if (dnevnaTemperatura >= 0)
      ledena = false;
  }
  return ledena;
}

int main() {
  int K;
  cin >> K;
  int brojLedenihNedelja=0;
  for (int i = 0; i < K; i++)
    if (LedenaNedelja())
      brojLedenihNedelja++;
  cout << brojLedenihNedelja << endl;
  return 0;
}
```

Угнежђене петље

Задатак је, наравно, могуће решити и без помоћне функције, али тада би програм садржао две угнежђене петље. Спољна петља обезбеђује учитавање K недеља (њена бројачка променљива може да се мења од 0, па све док је мања од K), док се у унутрашњој петљи (њена бројачка променљива може да се мења од 1 до 7), учитавају температуре за сваки дан.

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

int main() {
```

```
  int K;
  cin >> K;
  int brojLedenihNedelja = 0;
  for (int i = 0; i < K; i++) {
    bool ledenaNedelja = true;
    for (int j = 0; j < 7; j++) {
      int dnevnaTemperatura;
      cin >> dnevnaTemperatura;
      if (dnevnaTemperatura >= 0)
        ledenaNedelja = false;
    }
    if (ledenaNedelja)
      brojLedenihNedelja++;
  }
  cout << brojLedenihNedelja << endl;
  return 0;
}
```

Задатак: Утовар транспортног брода

У транспортни брод преносе се редом пакети задатих маса колицима дате носивости (она је већа или једнака маси сваког пакета). При томе пакете увек преносимо у целости. Када неки пакет не може стати у колица због тренутног прекорачења носивости колица, та колица превозимо до брода, и почињемо пуњење нових колица. Написати програм који приказује редом, за свака колица, број пакета и укупну масу пакета пренетих њима. На крају приказати редни број колица која садрже највећи број пакета.

Опис улаза

У првој линији стандардног улаза налази се носивост колица N (природан број $10 \leq N \leq 500$). У свакој наредној линија налази се маса пакета m_i (природан број $1 \leq m_i \leq N$) који треба пренети. Улаз се завршава линијом у којој се налази број 0 (њих има највише 50000).

Опис излаза

На стандардном излазу се прво, за свака колица, у по једној линији, налази број пакета на колицима и укупна маса пакета стављених на њих (раздвојени размаком), а затим у последњој линији и редни број колица са највећим бројем предмета.

Пример

Улаз	*Излаз*
30	3 23
5	4 22
10	1 9
8	2
12	
3	
5	
2	
9	
0	

Решење

Једна петља

У решењу одржавамо:

- редни број текуће серије (редни број колица),
- број њених елемената (број предмета на колицима)
- њихов збир (укупну масу предмета на колицима),
- податке о вредности и редном броју максимума (највећем до тада виђеном броју предмета на колицима и редном броју тих колица).

У петљи учитавамо податке о предметима и након учитавања сваког, проверавамо да ли је учитана ознака за крај (нула). Ако јесте постављамо индикатор краја којим ће петља бити прекинута након завршетка извршавања њеног тела.

Након тога, проверавамо да ли се завршило са попуњавањем текућих колица. То се дешава у два случаја: када је учитана ознака за крај или када је збир текуће масе предмета на колицима и управо учитаног предмета већа од носивости. Тада се списује број и маса пакета на њима и, ако је потребно, ажурира се максимум. Ако се није стигло до краја, припремају се нова колица тиме што се увећа њихов редни број, а број и маса пакета на текућим колицима поставе на нулу.

Ако се није стигло до краја, учитани предмет се поставља на колица тиме што се увећа број и маса предмета на колицима. Ово ће се десити и у случају када предмет стаје на текућа колица, а и у случају када предмет није стајао на текућа колица и када су зато започета нова колица (ово ће бити први предмет на тим новим).

Не би била грешка ни да су се нова колица започела и број и маса предмета на њима ажурирали и у случају када је учитана ознака краја - пошто се наредна итерација петље неће извршавати, ирелевантно је како ће се мењати вредности променљивих у текућој итерацији.

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

int main() {
  // nosivost paketa
  int Nosivost;
  cin >> Nosivost;

  // podaci o tekucem stanju na kolicima
  int brojKolica = 1, masaNaKolicima = 0, brojPaketaNaKolicima = 0;
  // podaci o kolicima sa najvise paketa
  int brojMaxKolica = 1, maxBrojPaketaNaKolicima = 0;

  // indikator da li se stiglo do kraja
  bool kraj = false;
  while (!kraj) {
    // ucitavamo paket
    int masaPaketa;
    cin >> masaPaketa;
    // proveravamo da li je u pitanju oznaka kraja
    if (masaPaketa == 0)
      kraj = true;

    // ako se tekuca kolica vise ne mogu popuniti (stiglo se do kraja ili
    // ucitani paket ne staje na njih)
    if (kraj || masaNaKolicima + masaPaketa > Nosivost) {
      // ispisujemo broj i masu paketa na njima
      cout << brojPaketaNaKolicima << " " << masaNaKolicima << endl;
      // ako na kolicima ima vise paketa od dosadasnjeg maksimuma,
      // azuriramo maksimum
      if (brojPaketaNaKolicima > maxBrojPaketaNaKolicima) {
        maxBrojPaketaNaKolicima = brojPaketaNaKolicima;
        brojMaxKolica = brojKolica;
      }
      // ako nismo dosli do kraja, zapocinjemo nova kolica
      if (!kraj) {
        brojKolica++;
        brojPaketaNaKolicima = 0;
        masaNaKolicima = 0;
      }
    }
    // ako nismo dosli do kraja, ucitani predmet postavljamo
```

```
    // na kolica (stara, ako staje na njih, tj. nova ako ne staje)
    if (!kraj) {
      masaNaKolicima += masaPaketa;
      brojPaketaNaKolicima++;
    }
  }
  // prijavljujemo redni broj kolica sa najvise paketa
  cout << brojMaxKolica << endl;
  return 0;
}
```

Глава 4

Структуре података

4.1 Основна употреба низова и вектора

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).

Опис улаза

У првој линији стандардног улаза уноси се број елемената низа n ($1 \leq n \leq 100$), а затим у следећих n линија елементи низа $-10000 \leq a_i \leq 10000$.

Опис излаза

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, као количник збира и броја тих бројева. 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 учитали 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 и броја елемената) и израчунати минимално одступање као минимум низа $|a_i - p|$ за $i = 0, 1, \ldots, n-1$.

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).

```
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <cmath>
#include <vector>
using namespace std;
```

```
int main() {
  // 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;
}
```

Уместо смештања елемената у вектор, можемо их сместити и у статички алоцирани низ. Иако је оваква решења могу бити мало бржа у фази алокације меморијског простора, она су по правилу нефлексибилнија. На пример, број елемената низа мора бити унапред познат, приликом писања програма и без обзира на то колико нам је стварно елемената потребно, увек се алоцира простор за максимални дозвољени број елемената. Стога ћемо у наставку чешће користити векторе.

```
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <cmath>
#include <vector>
using namespace std;

int main() {
  // broj elemenata niza
  int n;
  cin >> n;

  // ucitavamo sve elemente u staticki alocirani niz
  // iz teksta zadatka znamo da nece biti vise od 100 elemenata
  double a[100];
  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;
}
```

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.

Опис улаза

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.

Опис излаза

На стандардном излазу приказати на две децимале заокружено просечно одступање цена од минималне цене.

Пример

Улаз	*Излаз*
4 100 95 120 95	7.50

Решење

Овај задатак је веома сличан задатку Минимално одступање од просека. Један начин је да се након учитавања елемената (на пример, у класичан низ који има највише 200 елемената), пронађе минимална цена m, а након тога просечно одступање сваке цене од пронађене минималне (одступање је разлика цене артикла и минималне цене). Тј. да се израчуна

$$\frac{\sum_{i=0}^{n-1}(a_i - m)}{n}.$$

Минимум можемо пронаћи алгоритмом одређивања минимума, а просек као количник суме одступања коју израчунавамо алгоритмом сабирања (овај пут пресликане) серије бројева и броја производа у продавници.

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

int main() {
  // broj elemenata niza
  int n;
  cin >> n;

  // ucitavamo elemente i istovremeno odredjujemo minimum
  double a[200];
  double min = numeric_limits<double>::max();
  for (int i = 0; i < n; i++) {
    cin >> a[i];
    if(a[i] < min)
      min = a[i];
  }
```

```
  // odredjujemo prosecno odstupanje od minimuma kao kolicnik zbira
  // odstupanja i broja elemenata
  double s = 0.0;
  for (int i = 0; i < n; i++)
    s += a[i] - min;
  s /= n;

  // ispisujemo rezultat
  cout << fixed << showpoint << setprecision(2) << s << endl;
  return 0;
}
```

Задатак је могуће решити и без низа, само у једном пролазу. Наиме, важи да је

$$\sum_{i=0}^{n-1}(a_i - m) = \sum_{i=0}^{n-1} a_i - \sum_{i=0}^{n-1} m = \sum_{i=0}^{n-1} a_i - n \cdot m,$$

па је зато

$$\frac{\sum_{i=0}^{n-1}(a_i - m)}{n} = \frac{\sum_{i=0}^{n-1} a_i}{n} - m.$$

Зато је заправо могуће избећи низ и два пролаза и приликом учитавања израчунати вредност збира и минимума елемената и онда решење израчунати као разлику просечне и минималне цене.

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

int main() {
  // broj elemenata
  int n;
  cin >> n;

  // ucitavamo prvi element i na njega inicijalizujemo zbir i minimum
  double x;
  cin >> x;
  double zbir = x, min = x;
  // ucitavamo ostale elemente i azuriramo zbir i minimum
  for (int i = 1; i < n; i++) {
    cin >> x;
    zbir += x;
    if (x < min) min = x;
  }

  // izracunavamo prosecno odstupanje od minimuma
  double s = (double)zbir / (double)n - min;

  // ispisujemo rezultat
  cout << fixed << showpoint << setprecision(2) << s << endl;
}
```

Задатак: Стандардна девијација

У статистици *стандардна девијација* мери колико подаци варирају тј. одступају од просечне вредности. На пример, ако ученик има оцене 1, 2, 3, 4 и 5, његова просечна оцена ће бити 3, исто као и када има оцене 3,

3, 3, 3 и 3. У првој случају оцене много више варирају, па ће стандардна девијација бити већа, док у другом случају оцене не варирају ни мало, па ће стандардардна девијација бити 0.

Стандардна девијација низа $x_1, \ldots x_N$ се израчунава на основу формуле (μ је аритметичка средина, а σ стандардна девијација):

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{N}(x_i - \mu)^2}, \qquad \mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Опис улаза

Са стандардног улаза се уноси број елемената низа N ($1 \leq N \leq 100$), а затим и елементи низа (реални бројеви из интервала $[-1000, +1000]$).

Опис излаза

На стандардни излаз исписати стандардну девијацију, израчунату на 3 децимале.

Пример

Улаз	*Излаз*
`5` `1 2 3 4 5`	`1.414`

Решење

Ако стандардну девијацију рачунамо на основу дате формуле, потребно је да два пута прођемо кроз исте податке (једном приликом рачунања аритметичке средине, а други пут приликом рачунања стандардне девијације), па елементе смештамо у низ тј. вектор. Аритметичку средину рачунамо на уобичајени начин (рачунајући збир унетих података). Да бисмо израчунали стандардну девијацију, рачунамо збир квадрата одступања елемената од просека.

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>

using namespace std;

int main() {
  int n;
  cin >> n;
  vector<double> x(n);
  for (int i = 0; i < n; i++)
    cin >> x[i];

  double zbir = 0.0;
  for (int i = 0; i < n; i++)
    zbir += x[i];
  double prosek = zbir / n;

  double zbirKvadrataOdstupanja = 0.0;
  for (int i = 0; i < n; i++)
    zbirKvadrataOdstupanja += (x[i] - prosek) * (x[i] - prosek);
  double standardnaDevijacija = sqrt(zbirKvadrataOdstupanja / n);
  cout << fixed << showpoint << setprecision(3) << standardnaDevijacija << endl;
  return 0;
}
```

Лако се доказује да се стандардна девијација може израчунати и на основу наредне формуле:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2 - \left(\frac{1}{n}\sum_{i=1}^{N} x_i\right)^2}$$

Приметимо да се приликом примене ове формуле подаци не морају памтити у низу.

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>

using namespace std;

int main() {
  int n;
  cin >> n;
  double zbir = 0.0;
  double zbirKvadrata = 0.0;
  for (int i = 0; i < n; i++) {
    double x;
    zbir += x;
    zbirKvadrata += x*x;
  }
  double standardnaDevijacija = sqrt(zbirKvadrata / n - (zbir / n) * (zbir / n));
  cout << fixed << showpoint << setprecision(3) << standardnaDevijacija << endl;
  return 0;
}
```

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 $(a_1, \ldots, a_n)$ и $(b_1, \ldots, b_n)$ једнак је $a_1 \cdot b_1 + \ldots + a_n \cdot b_n$. Димензија вектора није унапред позната, али није већа од 50.

Опис улаза

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.

Опис излаза

Исписати вредност скаларног производа првог и другог вектора на стандардни излаз.

Пример

Улаз	*Излаз*
3 1 2 3 3 2 1	10

Објашњење

Резултат се добија као збир $1 \cdot 3 + 2 \cdot 2 + 3 \cdot 1$.

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.

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

int main() {
  // dimenzija vektora
  int n;
  cin >> n;

  //ucitavanje prvog vektora
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  //ucitavanje drugog vektora
  vector<int> b(n);
  for (int i = 0; i < n; i++)
    cin >> b[i];

  // izracunavanje skalarnog proizvoda
  int skalarniProizvod = 0;
  for (int i = 0; i < n; i++)
    skalarniProizvod += a[i]*b[i];

  // ispis resenja
  cout << skalarniProizvod << endl;

  return 0;
}
```

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, множити је са одговарајућом координатом првог вектора и додавати на текући збир.

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

int main() {
  const int MAX = 50;
  int n;
  cin >> n;

  // ucitavanje prvog vektora
  int a[MAX];
  for (int i = 0; i < n; i++)
    cin >> a[i];

  // vrednost skalarnog proizvoda
  int skalarniProizvod = 0;

  // ucitavanje koordinata drugog vektora u promenljivu b i
  // azuriranje skalarnog proizvoda
  for (int i = 0; i < n; i++) {
    int b;
    cin >> b;
    skalarniProizvod += a[i] * b;
```

```
  }

  cout << skalarniProizvod << endl;

  return 0;
}
```

Задатак: Линије у обратном редоследу

Напиши програм који исписује све линије које се учитају са стандардног улаза у обратном редоследу од редоследа учитавања.

Опис улаза

Са стандардног улаза се учитавају линије текста, све до краја улаза.

Опис излаза

На стандардни излаз исписати учитане линије у обратном редоследу.

Пример

Улаз	*Излаз*
`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 C++ 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 (getline(cin, linija))
    a.push_back(linija);
  for (int i = a.size() - 1; i >= 0; i--)
    cout << a[i] << endl;
  return 0;
}
```

Задатак: Учешљавање аутомобила

Познати су регистарски бројеви аутомобила у две колоне. По закону се приликом спајања у једну траку аутомобили пропуштају наизменично. Написати програм који одређује редослед аутомобила након спајања у једну траку.

Опис улаза

Са стандардног улаза се учитава број аутомобила у левој колони (природан број између 0 и 100), а затим регистарске таблице свих аутомобила у левој колони (свака је наведена у посебном реду). Затим се на исти начин учитавају подаци о аутомобилима из десне колоне.

Опис излаза

На стандардни излаз исписати регистарске таблице у обједињеној колони (прво пролази први аутомобил из леве колоне).

Пример

Улаз	*Излаз*
4	BG732LM
BG732LM	NS111PP
BG311AD	BG311AD
KG241ED	NI829RL
SO919MN	KG241ED
2	SO919MN
NS111PP	
NI829RL	

Решење

Све регистарске таблице ћемо учитати у два вектора, а затим ћемо штампати по један елемент из оба низа, све док не стигнемо до краја једног од два низа. Након тога ћемо исписати преостале елементе из другог низа (ако их има).

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int main() {
  int brojLevo;
  cin >> brojLevo;
  vector<string> levo(brojLevo);
  for (int i = 0; i < brojLevo; i++)
    cin >> levo[i];

  int brojDesno;
  cin >> brojDesno;
  vector<string> desno(brojDesno);
  for (int i = 0; i < brojDesno; i++)
    cin >> desno[i];

  int l = 0, d = 0;
  while (l < brojLevo && d < brojDesno) {
    cout << levo[l++] << endl;
    cout << desno[d++] << endl;
  }

  while (l < brojLevo)
    cout << levo[l++] << endl;

  while (d < brojDesno)
    cout << desno[d++] << endl;

  return 0;
}
```

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.

Опис улаза

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.

Опис излаза

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 1 2 3 4 5	2 4 1 3 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 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, 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`).

```
#include <iostream>

using namespace std;

// maksimalan broj elemenata dat tekstom 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;

  // ucitavamo broj elemenata
  int k;
  cin >> k;
  // ucitavamo k elemenata
  for (int i = 0; i < k; i++) {
    // ucitavamo 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
  }

  // ispisujemo niz ucitanih parnih brojeva
  for (int i = 0; i < p; i++)
```

```
    cout << parni[i] << " ";
  cout << endl;

  // ispisujemo niz ucitanih neparnih brojeva
  for (int i = 0; i < n; i++)
    cout << neparni[i] << " ";
  cout << endl;

  return 0;
}
```

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.

```
#include <iostream>

using namespace std;

// maksimalan broj elemenata dat tekstom zadatka
const int MAX = 100;

// ucitavamo elemente u niz a i vracamo njihov broj kroz promenljivu k
void ucitajNiz(int a[], int& k) {
  // ucitavamo 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() {
  // ucitavamo 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 C++ 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).

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  // niz parnih i niz neparnih brojeva i njihove trenutne duzine
  vector<int> parni, neparni;

  // ucitavamo broj elemenata
  int k;
  cin >> k;
  // ucitavamo k elemenata
  for (int i = 0; i < k; i++) {
    // ucitavamo naredni element
    int x;
    cin >> x;
    // ako je paran
    if (x % 2 == 0)
      // smestamo ga u niz parnih
      parni.push_back(x);
    // u suprotnom
    else
      neparni.push_back(x);
  }

  // ispisujemo niz ucitanih parnih brojeva
  for (int x : parni)
    cout << x << " ";
```

```
  cout << endl;

  // ispisujemo niz ucitanih neparnih brojeva
  for (int x : neparni)
    cout << x << " ";
  cout << endl;

  return 0;
}
```

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.

```
#include <iostream>
#include <vector>

using namespace std;

// ucitavamo 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);
  return parni;
}

// izdvajamo neparne elemente vektora u novi vektor
vector<int> izdvojNeparne(const vector<int>& a) {
  vector<int> neparni;
  for (int x : a)
    if (x % 2 != 0)
      neparni.push_back(x);
  return neparni;
}

// ispisujemo elemente vektora a na standardni izlaz (razdvojene razmakom)
void ispisiVektor(const vector<int>& a) {
  for (int x : a)
    cout << x << " ";
  cout << endl;
}
```

```
int main() {
  // ucitavamo brojeve u niz
  vector<int> a = ucitajVektor();

  // izdvajamo elemente u niz parnih i niz neparnih brojeva
  vector<int> parni = izdvojParne(a);
  vector<int> neparni = izdvojNeparne(a);

  // ispisujemo niz ucitanih parnih brojeva
  ispisiVektor(parni);
  // ispisujemo niz ucitanih neparnih brojeva
  ispisiVektor(neparni);

  return 0;
}
```

Задатак: Циклично померање за једно место

Написати програм који учитава низ целих бројева а 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 $p > q$.

Опис улаза

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.

Опис излаза

У свакој линији стандарног 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	

Решење

Учитаћемо број елемемената и све чланове низа. У петљи ћемо учитавати вредности p и q. Након анализе вредности p и q, ако је $p < q$ позиваћемо помоћну функцију за померање удесно, а ако је $p > q$, помоћну функцију за померање улево. Функције врше ротирање дела низа између позиција p и q, укључујући и елементе на тим позицијама и у оба случаја треба прво треба навести мању, а затим већу позицију.

Имплементација цикличног померања за једно место

Функција за циклично померање улево, прима два индекса: индекс левог краја l и индекс десног краја d. Сви елементи са позиција од $l + 1$ до d се померају за једно место улево, док се елемент са позиције l уписује на крај низа, на позицију d. При обради чланова низа полази се од индекса l. Вредност са те позиције морамо сачувати у помоћној променљивој, пре уласка у петљу (у супротном би била преписана након померања елемента са позиције $l + 1$). У петљи се врши померање вредности елемената низа са позиција $i + 1$ на i. Петља почиње од индекса l и завршава се када се помери елемент низа са позиције d. На позицију већег индекса d се тада уписује вредност помоћне променљиве.

Функција за циклично померање удесно, је веома слична, осим што је неопходно да се обрада врши од десног ка левом крају (наиме, ако бисмо обилазак вршили с лева на десно, приликом пребацивања елемента са позиције l на позицију $l+1$ вредност на позицији $l+1$ би била пребрисана, пре него што је померена и тиме трајно уништена). Вредност са позиције d морамо сачувати у помоћној променљивој, пре уласка у петљу. У петљи се врши померање вредности елемената низа са позиција $i-1$ на i. Петља почиње од индекса d и завршава се када се помери елемент низа са позиције l. На позицију левог, мањег индекса l се тада уписује вредност помоћне променљиве.

Нагласимо да функције примају само адресе почетка низа, тако да слободно могу да мењају елементе оригиналног низа. Да је уместо низа употребљен вектор, њега би било неопходно пренети у функцију по референци (у супротном би у функцију била пренетака копија вектора и оригиналнин вектор не би био мењан).

```
#include <iostream>

using namespace std;

// ciklicno pomera ulevo deo niza a izmedju pozicija l i d
// (ukljucujuci i njih)
void rotirajUlevo(int a[], int l, int d) {
    int pom = a[l];
    for (int i = l; i < d; i++)
        a[i] = a[i + 1];
    a[d] = pom;
}

// ciklicno pomera udesno deo niza a izmedju pozicija l i d
// (ukljucujuci i njih)
void rotirajUdesno(int a[], int l, int d) {
  int pom = a[d];
  for (int i = d; i > l; i--)
    a[i] = a[i - 1];
  a[l] = pom;
}

int main() {
  // ucitavamo broj elemenata niza
  int n;
  cin >> n;
  // ucitavamo niz
  int a[50000];
  for (int i = 0; i < n; i++)
    cin >> a[i];

  while (true) {
    // ucitavamo pozicije p i q
    int p, q;
    cin >> p >> q;

    if (p < q)
      rotirajUdesno(a, p, q);
    else if (p > q)
      rotirajUlevo(a, q, p);
    else
      // ako je p = q postupak se prekida
      break;
  }

  // ispisujemo rezultat
  for (int i = 0; i < n; i++)
```

```
    cout << a[i] << endl;

  return 0;
}
```

Задатак: Обртање низа

Написати програм који учитава 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.

Опис улаза

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.

Опис излаза

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`.

```
#include <iostream>
#include <vector>

using namespace std;

// unos elemenata niza
vector<int> unosNiza() {
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  return a;
}

// obrtanje dela niza izmedju pozicija p i q (ukljucujuci i njih)
void obrni(vector<int>& a, int p, int q) {
```

```
  for (int i = p, j = q; i < j; i++, j--)
    swap(a[i], a[j]);
}

// ispis elemenata niza a
void ispisNiza(const vector<int>& a) {
  for (int x : a)
    cout << x << endl;
}


int main()  {
  // ucitavamo niz
  vector<int> a = unosNiza();

  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
    obrni(a, p, q);
  }

  // ispisujemo rezultat
  ispisNiza(a);
  return 0;
}
```

Задатак: Ниска палиндром

Написати програм којим се проверава да ли је дата реч *палиндром*, што значи да се једнако чита слево на десно и здесна на лево.

Опис улаза

Прва и једина линија стандардног улаза садржи реч која се састоји само од малих слова енглеске абецеде.

Опис излаза

На стандардном излазу приказати реч `da` ако реч представља палиндром, тј. `ne` у супротном.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
`madam`	`da`	`ranac`	`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 промењиве 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 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 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.

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

bool palindrom(const string& s) {
  for (int i = 0, j = s.size() - 1; i < j; i++, j--)
    if (s[i] != s[j])
      return false;
  return true;
}

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

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 здесна на лево, при чему се разматрају само слова и не прави се разлика између великих и малих слова.

Опис улаза

Прва и једина линија стандардног улаза садржи реченицу (састављену од слова, празнина и интерпукцијских знакова).

Опис излаза

На стандардном излазу приказати реч `da` ако реченица представља палиндром иначе приказати реч `ne`.

Пример

Улаз	*Излаз*
`Ana voli Milovana!!!`	`da`

Решење

Проверу да ли је ниска палиндром већ смо описали у задатку Ниска палиндром. Разлика између тог и овог задатка је то што се у овом задатку занемарују сви карактери осим слова, и то што се не прави разлика између великих и малих слова.

Парове за поређење бирамо тако што једним бројачем i крећемо од почетка реченице, а другим бројачем j крећемо од краја реченице и петљу извршавамо све док је $i < j$ (или док се у телу петље не наиђе на различита слова). Ако су на позицијама i и j слова, онда их поредимо (али претходно их преводимо у мала) и ако су различита, закључујемо да реченица није палиндромска (функција може да врати вредност `false`). Ако су слова иста, онда прелазимо на следеће знакове реченица (бројач i увећавамо за 1, а бројач j умањујемо за 1). Ако на позицији i није слово прелазимо на следећи знак са леве стране (бројач i увећавамо за 1). Ако на позицији j није слово, прелазимо на следећи знак са десне стране (бројач j умањујемо за 1). У случају да су сви одговарајући парови слова једнаки провера се завршава када i достигне j, јер тада смо извршили анализу целе реченице, и закључак је да је дата реченица палиндром.

Проверу да ли је карактер слово у језику C++ можемо извршити библиотечком функцијом `isalpha`. Конверзију у мало слово у језику C++ можемо извршити библиотечком функцијом `tolower`.

```cpp
#include <iostream>
#include <string>
#include <cctype>
using namespace std;

bool palindrom(const string& s) {
  int i = 0, j = s.size() - 1;
  while (i < j) {
    if (!isalpha(s[i]))
      i++;
    else if (!isalpha(s[j]))
      j--;
    else {
      if (tolower(s[i]) != tolower(s[j]))
        return false;
      else
        i++, j--;
    }
  }
  return true;
}

int main() {
  string s;
  getline(cin, s);
  if (palindrom(s))
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

4.2 Сортирање

Задатак: Сортирање дневник

Познат је списак имена и презимена ученика једног одељења. Исписати их у редоследу у ком су поређани у електронском дневнику (уређени су лексикографски, прво на основу презимена, а затим на основу имена).

Опис улаза

Свака линија стандардног улаза садржи име и презиме ученика (записани су словима енглеске абецеде).

Опис излаза

На стандардни излаз исписати ученике у траженом редоследу. За сваког ученика исписати прво презиме, па онда име раздвојене запетом.

Пример

Улаз	*Излаз*
`Aleksa Jovanovic`	`Andrejic, Dimitrije`
`Dimitrije Andrejic`	`Boskovic, Ivana`
`Ivana Boskovic`	`Jovanovic, Aleksa`
`Dragana Petrovic`	`Petrovic, Dragana`

Решење

Задатак најлакше можемо решити тако што креирамо низ парова (презиме, име) и сортирамо га библиотечком функцијом. Наиме, парови се пореде тако што се прво пореди први елемент (презиме), па затим други (име). Пошто унапред није познато колико ће бити ученика, користимо вектор на чији крај додајемо једног по једног ученика (уместо методом `push_back`, пар се ефикасније додаје у вектор методом `emplace_back`).

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <utility>

using namespace std;

int main() {
  vector<pair<string, string>> ucenici;
  string ime, prezime;
  while (cin >> ime >> prezime)
    ucenici.emplace_back(prezime, ime);
  sort(begin(ucenici), end(ucenici));
  for (const auto& [prezime, ime] : ucenici)
    cout << prezime << ", " << ime << endl;
  return 0;
}
```

Задатак: Пласман

На студентском хакатону студентски тимови су освајали поене решавајући програмерске изазове. На крају такмичења познат је списак тимова и њихових освојених поена. Написати програм који сортира тимове по поенима, опадајући и исписује листу, али тако да се испред сваког тима одреди његов пласман на Хакатону. Ако два или више тимова имају исти број поена они деле исти пласман.

Опис улаза

Са стандардног улаза се учитава број тимова n ($1 \leq n100$). У наредних N редова наведено је име тима и освојен број поена (цео број између 0 и 100).

Опис излаза

На стандардни излаз исписати сортирани списак. У сваком реду исписати пласман, затим назив тима и на крају освојени број поена.

Пример

Улаз	*Излаз*
`4`	`1. matematicari123 45`
`hakeri 38`	`2. hakeri 38`
`matematicari123 45`	`2. zikini_petlici 38`
`zikini_petlici 38`	`4. pythonovci 21`
`pythonovci 21`	

Решење

Податке о сваком тиму можемо представити структуром (или уређеним паром). Низ структура можемо сортирати опадајући по броју поена, тако што ћемо употребити библиотечку функцију `sort` којој ћемо као трећи аргумент проследити функцију која пореди две структуре тако што пореди број поена сачуван у њима.

Након сортирања ћемо исписати низ, при чему пажљиво треба да бројимо пласман. Први тим у низу сигурно има редни број 1. За наредне тимове постоје две могућности:

- ако им је број поена једнак претходном тиму, тада имају и исти пласман.
- у супротном је пласман једнак редном броју у низу увећаном за 1 (јер се тимови броје од 1, а елементи низа од 0).

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Tim {
```

```
  string naziv;
  int poeni;
};

int main() {
  int n;
  cin >> n;
  vector<Tim> timi(n);
  for (int i = 0; i < n; i++)
    cin >> timi[i].naziv >> timi[i].poeni;

  sort(begin(timi), end(timi),
       [](const Tim& t1, const Tim& t2) {
         return t1.poeni > t2.poeni;
       });

  int plasman = 1;
  cout << plasman << "." << " "
       << timi[0].naziv << " " << timi[0].poeni << endl;
  for (int i = 1; i < n; i++) {
    if (timi[i].poeni < timi[i-1].poeni)
      plasman = i + 1;
    cout << plasman << "." << " "
         << timi[i].naziv << " " << timi[i].poeni << endl;
  }

  return 0;
}
```

Задатак: Сортирање налога на серверу Алас

На студентском серверу Алас налози су облика `mi24135` (прва два карактера служе да означе смер, наредна два годину уписа, а преостала 3 број индекса студента). Напиши програм који сортира студенте по годинама уписа, у оквиру сваке године по смеровима (редослед смерова је `I`, па `R`, па `M`, па `V`, па `N`), а у оквиру сваког смера по броју индекса.

Опис улаза

Са стандардног улаза се учитава број студената n, а затим списак од n студентских налога.

Опис излаза

На стандардни излаз исписати сортиран списак студената.

Пример

Улаз	*Излаз*
`8`	`7/22 I`
`mi24385`	`178/22 R`
`mr24145`	`27/23 M`
`mm23123`	`123/23 M`
`mi24123`	`123/24 I`
`mr22178`	`385/24 I`
`mn24003`	`145/24 R`
`mi22007`	`3/24 N`
`mm23027`	

Решење

Податке ос ваком студенту можемо организовати у структуру, а податке о свим студентима можемо сместити у низ или вектор структура. Податак о смеру можемо чувати у облику ниске (типа `string`), а годину уписа и индекс у облику података типа `int`. Ове податке ћемо издвојити из налога студената. У ту сврху нам је

корисна метода `substr` класе `string` која омогућава да се издвоји подниска (која почиње на датој позицији и дате је дужине) и функција `stoi` која служи да се одреди цео број записан у облику ниске.

Неопходно је дефинисати функцију којом се дефинише критеријум поређења структура. Могуће је дефинисати методу под специјалним називом `operator<` унутар структуре, која омоућава да се две структуре пореде оепратором `<`. Након дефинисања овог оператора, функција `sort` не мора да има додатне аргументе (осим два итератора којима се задаје део низа тј. вектора који се сортира).

```
#include <iostream>
#include <string>
#include <vector>
#include <cctype>
#include <algorithm>

using namespace std;

struct student {
  string smer;
  int godina;
  int indeks;

  bool operator<(const student& drugi) const {
    string smerovi = "IRMVN";
    if (godina < drugi.godina)
      return true;
    if (godina > drugi.godina)
      return false;
    if (smerovi.find(smer) < smerovi.find(drugi.smer))
      return true;
    if (smerovi.find(smer) > smerovi.find(drugi.smer))
      return false;
    return indeks < drugi.indeks;
  }
};

student protumaciNalog(const string& nalog) {
  student s;
  s.smer = toupper(nalog[1]);
  s.godina = stoi(nalog.substr(2, 2));
  s.indeks = stoi(nalog.substr(4));
  return s;
}

int main() {
  int n;
  cin >> n;
  vector<student> studenti(n);
  for (int i = 0; i < n; i++) {
    string nalog;
    cin >> nalog;
    studenti[i] = protumaciNalog(nalog);
  }
  sort(begin(studenti), end(studenti));
  for (const student& s: studenti)
    cout << s.indeks << "/" << s.godina << " " << s.smer << endl;
  return 0;
}
```

Уместо оператора можемо дефинисати функцију којом се пореде два студента и онда навести име те функције као трећи аргумент функције `sort`.

4.3 Матрице и вишедимензиони низови

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.

Опис улаза

Са стандардног улаза се учитавају 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).

Опис излаза

Матрица димензије $m \times n$ која одређује број бомби у околини сваког поља полазне матрице.

Пример

Улаз	*Излаз*
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 учитавања 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 угнежђене 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 угнежђене 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.

```
#include <iostream>

using namespace std;

// 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`.

```
#include <iostream>

using namespace std;

// 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 uz okvir debljine 1 koji sadrzi sve vrednosti false
  bool bomba[MAX_M + 2][MAX_N + 2] = {{false}};
  // ucitavamo matricu bombi
  for (int v = 1; v <= m; v++)
    for (int k = 1; 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 = 1; v <= m; v++)
    for (int k = 1; k <= n; k++) {
      broj_bombi[v - 1][k - 1] = 0;
      for (int dv = -1; dv <= 1; dv++)
        for (int dk = -1; dk <= 1; dk++) {
          if (dv == 0 && dk == 0)
            continue;
          broj_bombi[v - 1][k - 1] += bomba[v + dv][k + dk];
```

```
        }
    }

  // 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;
}
```

Уместо статички алоциране матрице, можемо употребити и вектор вектора. На пример,

```
vector<vector<bool>> bombe(m, vector<bool>(n, false));
```

декларише промењљиву `bombe` чији је тип `vector<vector<bool>>` и иницијализује је тако да се алоцира меморија за вектор који садржи `m`, вектора димензије `n` чији су сви елементи постављени на вредност `false` (подсетимо се, приликом конструкције вектора, први параметар је број елемената вектора, а други иницијална вредност елемената вектора).

Предност овакве репрезентације матрица је то што се количина утрошене меморије аутоматски прилагођава улазним подацима и није потребно унапред да знамо ограничење димензије матрице. Мана је то што фаза алокације меморије може бити знатно спорија него када се користе статички алоциране матрице.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  // dimenzije matrice
  int m, n;
  cin >> m >> n;

  // matrica bombi
  vector<vector<bool>> bomba(m, vector<bool>(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)
  vector<vector<int>> broj_bombi(m, vector<int>(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;
}
```

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.

Опис улаза

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 p ($1 \leq p, q \leq 10$).

Опис излаза

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.

Пример

Улаз	*Излаз*
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	

Решење

Најбоља структура података за чување збирних података о свим утакмицама је матрица која овде има улогу асоцијативног низа (даје могућност да се одреди вредност на основу кључа, при чему је овде кључ пар тимова, а вредност број датих кошева).

Како редни бројеви тимова почињу од 1, а индекси низова (и матрица) од 0, на позицији $(p-1, q-1)$ у матрици чуваћемо укупан број кошева које је тим p током сезоне дао тиму q (за разне вредности p и q). Ове збирове ћемо израчунати стандардним алгоритмом сабирања серије бројева. На почетку упишемо да је сваки тим дао 0 кошева другом тиму (матрицу иницијализујемо нулама). Потом учитавамо резултате сваке кошаркашке утакмице на турниру. Сабирамо кошеве сваког од тима водећи рачуна да учитани број p означава тим који је постигао кошеве, а учитани број q означава тим који је примио кошеве и да се релевантни подаци у матрици налазе на позицијама $(p-1, q-1)$ и $(q-1, p-1)$.

```
#include <iostream>

using namespace std;

const int MAX_TIMOVA = 10;

int main() {
  int kosevi[MAX_TIMOVA][MAX_TIMOVA] = {{0}};

  // broj utakmica
  int n;
  cin >> n;
```

```
  for (int i = 0; i < n; i++) {
    // utakmica koju su igrali timovi p i q zavrsena je rezultatom r : s
    int p, q, r, s;
    cin >> p >> q >> r >> s;
    kosevi[p - 1][q - 1] += r;  // dodajemo po r koseva
    kosevi[q - 1][p - 1] += s;  // dodajemo po s koseva
  }

  // zanima nas ukupan ishod za m parova
  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    // ucitavamo par timova p i q
    int p, q;
    cin >> p >> q;
    // ispisujemo ukupan rezultat njihovih susreta tokom sezone
    cout << kosevi[p - 1][q - 1] << ":" << kosevi[q - 1][p - 1] << endl;
  }

  return 0;
}
```

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).

Опис улаза

Са стандардног улаза учитава се $0-1$ матрица димензије 8×8 чијих 8 јединица описује положај 8 дама.

Опис излаза

На стандардном излазу исписати текст `NE` ако се даме не нападају или `DA` ако се неке две даме нападају.

Пример 1

Улаз	*Излаз*
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	NE

Пример 2

Улаз	*Излаз*
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	DA

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 - 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. 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.

```
#include <iostream>
#include <algorithm>

using namespace std;

void ucitaj(int tabla[8][8]) {
  for (int v = 0; v < 8; v++)
    for (int k = 0; k < 8; k++)
      cin >> tabla[v][k];
}

bool proveraKolona(int tabla[8][8]) {
  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(int tabla[8][8]) {
  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(int tabla[8][8]) {
  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(int tabla[8][8]) {
  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() {
  int tabla[8][8];
  ucitaj(tabla);
  if (proveraVrsta(tabla) ||
      proveraKolona(tabla) ||
      proveraGlavnihDijagonala(tabla) ||
      proveraSporednihDijagonala(tabla))
    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. 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).

```
#include <iostream>
#include <cmath>

using namespace std;

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 proveraParovaDama() {
  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;
}
```

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).

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

int main(){
  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;

  return 0;
}
```

Задатак: Множење матрица

Јовани је рођендан и одлучила је да направи торту да почасти своје другарице и другаре. Она зна рецепте за t различитих торти. У сваку торту иде s различитих састојака (за сваку торту 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 торту да прави и у којој продавници да купује састојке да би јој остало што више пара за екскурзију.

Опис улаза

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 торти (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).

Опис излаза

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 2 1 3 2 3 2 2 1 2 1 2 3 250 170 160 120 135 142 145 155	1 1 1189

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. 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 - 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.

```
#include <iostream>

using namespace std;

// najveca dimenzija matrice dopustena uslovima zadatka
const int MAX = 10;

// 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).

Задатак: Релација зависности

На факултету постоје зависности између испита који се полажу. 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.

Опис улаза

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.

Опис излаза

На стандардни излаз исписати `DA` ако матрица испуњава све задате услове тј. `NE` ако нарушава било који од њих.

Пример

Улаз	*Излаз*
6 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	DA

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 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, j) 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).

```
#include <iostream>

using namespace std;

const int MAX = 100;

// 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() {
  // ucitavamo 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: 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

Дефиниције

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 леемент 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 $\{papir, kamen, makaze\}$ у познатој игри (папир је бољи од камена а камен од маказа, али папир није бољи од маказа).

Опис улаза

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.

Опис излаза

У свакој од 4 линије излаз треба исписати реч `da` или `ne`. Исписане речи су редом одговори на питања да ли је релација која је задата матрицом рефлексивна, симетрична, антисиметрична и транзитивна.

Пример

Улаз	*Излаз*
`3`	`da`
`1 1 0`	`ne`
`1 1 0`	`ne`
`1 1 1`	`da`

Решење

Нека је релација ρ задата квадратном бинарном матрицом a (матрица је бинарна ако су сви њени елементи нуле или јединице). Тада, на основу дефиниција особина релације и начина представљања релације матрицом важи:

Релација је рефлексивна ако је $a_{i,i} = 1$ за сваки индекс i.

Релација је симетрична ако је $a_{i,j} = a_{j,i}$ за сваки пар индекса i, j.

Релација је антисиметрична ако је за сваки пар различитих индекса i, j бар један од бројева $a_{i,j}, a_{j,i}$ једнак нули.

Релација је транзитивна ако не постоји тројка индекса i, j, k таква да су $a_{i,j}, a_{j,i}$ једнаки јединици, а $a_{i,k}$ нули.

Овако формулисане особине релације директно се преводе у програм, коришћењем алгоритма линеарне претраге.

```
#include <iostream>

using namespace std;

int main() {
    const int N_MAX = 10;
    int a[N_MAX][N_MAX] = {{0}};

    int n;
    cin >> n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
```

```
            cin >> a[i][j];

    bool refleksivna = true;
    for (int i = 0; i < n; i++)
        if (a[i][i] == 0)
            refleksivna = false;

    bool simetricna = true;
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (a[i][j] != a[j][i])
                simetricna = false;

    bool antisimetricna = true;
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (a[i][j] == 1 && a[j][i] == 1)
                antisimetricna = false;

    bool tranzitivna = 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] == 1 && a[j][k] == 1 && a[i][k] == 0)
                    tranzitivna = false;

    cout << (refleksivna ? "da" : "ne") << endl;
    cout << (simetricna ? "da" : "ne") << endl;
    cout << (antisimetricna ? "da" : "ne") << endl;
    cout << (tranzitivna ?  "da" : "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.

Опис улаза

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.

Опис излаза

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.

Решење

Најједноставнији начин да се реши задатак је да се формира нова матрица B исте димензије. Матрицу B можемо иницијално да попунимо елементима матрице A, додајући јединице по главној дијагонали.

Након тога, потребно је за сваки пар индекса i, j проверити да ли се из града i може стићи у град j користећи једно преседање. Другим речима, треба проверити да ли постоји индекс k такав да су $a_{i,k}, a_{k,j}$ једнаки јединици. Ако такво k постоји, онда треба уписати јединицу и на место $b_{i,j}$.

```
#include <iostream>

using namespace std;

int main() {
  const int N_MAX = 10;
  int a[N_MAX][N_MAX] = {{0}};

  int n;
  cin >> n;
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      cin >> a[i][j];

  int b[N_MAX][N_MAX] = {{0}};
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      b[i][j] = a[i][j];
      for (int k = 0; k < n; k++) {
        if (a[i][k] == 1 && a[k][j] == 1) {
          b[i][j] = 1;
          break;
        }
      }
    }
    b[i][i] = 1;
  }

  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      cout << b[i][j] << " ";
    cout << 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.

Опис улаза

Са стандардног улаза учитава се димензија квадратне матрице n, а затим и елементи матрице.

Опис излаза

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 укључучујући 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).

```
#include <iostream>

using namespace std;

int main() {
  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;

  return 0;
}
```

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 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.

```
#include <iostream>

using namespace std;

int main() {
  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;

  return 0;
}
```

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.

Опис улаза

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 размацима). 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).

Опис излаза

На стандардни излаз исписати све оцене, а затим и просек оцена заокружен на две децимале у редоследу који је задат на улазу.

Пример

Улаз	*Излаз*
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 учитавања броја ученика, а онда за сваког ученика учитавамо 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.

```
#include <iostream>
#include <iomanip>
#include <vector>

using namespace std;

double prosek(const vector<int>& ocene) {
  int zbir = 0;
  for (const int ocena : ocene)
    zbir += ocena;
  return (double)zbir / (double)ocene.size();
```

```
}

int main() {
  int brojUcenika;
  cin >> brojUcenika >> ws;
  vector<vector<int>> ocene(brojUcenika);
  for (int i = 0; i < brojUcenika; i++) {
    string linija;
    getline(cin, linija);
    stringstream ss(linija);
    int o;
    while (ss >> o)
      ocene[i].push_back(o);
  }

  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;
}
```

4.4 Скупови и мапе

Задатак: Дупликати

Претпоставимо да су интернет адресе представљене природним бројевима (IP адресе се, на пример, чувају у облику неозначених 32-битних бројева). Претраживач чува списак свих адреса које је корисник посетио током неког претходног периода. Корисник је многе адресе посећивао и више пута. Напиши програм који одређује број различитих адреса које је корисник посетио.

Опис улаза

Са стандардног улаза се уноси број n ($1 \leq n \leq 10^5$), а затим и n природних бројева (мањих од 2^{32}), сваки у посебном реду.

Опис излаза

На стандардни излаз исписати број различитих адреса које је корисник посетио.

Пример

Улаз	*Излаз*
8 123456789 234567890 345678901 234567890 456789012 234567890 456789012 234567890	4

Решење

Сортирање

Један од најчешћих начина уклањања дупликата из низа је заснован на сортирању, јер се након сортирања дупликати нађу један до другог. Сортирање можемо најбоље урадити позивом библиотечке функције. Након

сортирања пролазимо редом кроз низ и бројимо први елемент, а затим и све елементе који су различити од њима претходног (то су прва појављивања елемената у сортираном низу).

Често је непожељно да се током неке анализе података подаци трансформишу, јер то може да спречи неке наредне анализе података. На пример, ако сортирамо низ, даље анализе у којима је важан редослед података неће бити могуће, јер се губи информација о полазном редоследу. Због тога је пре сортирања понекад неопходно направити копију низа.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// broj razlicitih elemenata niza a
int brojRazlicitih(const vector<unsigned>& a) {
  // pravimo kopiju niza, da bi originalni niz ostao nepromenjen
  auto as = a;
  // sortiramo niz
  sort(as.begin(), as.end());
  // brojimo prvi element i sve elemente koje su razliciti od
  // svojih prethodnika
  int broj = 1;
  for (int i = 1; i < as.size(); i++)
    if (as[i] != as[i-1])
      broj++;
  return broj;
}

int main() {
  // ucitavamo niz
  int n;
  cin >> n;
  vector<unsigned> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  cout << brojRazlicitih(a) << endl;

  return 0;
}
```

Скупови

Библиотеке савремених програмских језика обично нуде и колекције података за репрезентовање скупова. Једна могућност је да се користи колекција заснована на балансираном бинарном дрвету.

У језику C++ таква је колекција `set`. Додавање елемента у скуп се врши методом `insert` (при том се аутоматски води рачуна да се скуп не мења додавањем елемента који већ постоји), док се број елемената одређује методом `size()`.

```
#include <iostream>
#include <vector>
#include <set>

using namespace std;

int main() {
  // ucitavamo elemente u skup
  set<unsigned> a;
  int n;
  cin >> n;
```

```
  for (int i = 0; i < n; i++) {
    unsigned x;
    cin >> x;
    a.insert(x);
  }
  // ispisujemo broj elemenata skupa
  cout << a.size() << endl;

  return 0;
}
```

Задатак: Број различитих дужина дужи

Дато је N парова тачака које представљају крајеве дужи у простору. Исписати колико различитих дужина дужи се појављује у задатом скупу дужи.

Опис улаза

У првом реду улаза налази се природан број N ($N \leq 50000$) који представља број дужи. У следећих N редова следи опис тих N дужи са 6 целих бројева ($-10^9 \leq X_1, Y_1, Z_1, X_2, Y_2, Z_2 \leq 10^9$) одвојених празним местима, који редом представљају крајеве сваке дужи.

Опис излаза

У једини ред излаза потребно је исписати колико различитих дужина се појављује у задатом скупу дужи.

Пример

Улаз	*Излаз*
7 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 1 1 1	3

Решење

Скуп дужина дужи

Задатак можемо решити и помоћу библиотечких структура података.

У језику C++ можемо употребити једну од две имплементације скупа (`set` или `unordered_set`).

Квадрате дужина дужи убацујемо у скуп и на крају очитавамо број елемената тог скупа. Јако је важно нагласити да проналажење самих дужина кореновањем, у облику реалних бројева, не би дало добро решење. Наиме, услед грешака у запису тј. заокруживању реалних бројева, дешава се да се једнаке вредности могу протумачити као различите, а различите вредности као једнаке. Стога реалне бројеве никада не би требало чувати као елементе скупа нити као кључеве у мапи.

```
#include <iostream>
#include <unordered_set>

using namespace std;

int main() {
  // skup svih duzina duzi
  unordered_set<long long> duzine;

  // ucitavamo koordinate temena duzi i ubacujemo njihove duzine u skup
  int n;
  cin >> n;
```

```
  for (int i = 0; i < n; i++) {
    long long x1, y1, z1, x2, y2, z2;
    cin >> x1 >> y1 >> z1 >> x2 >> y2 >> z2;
    duzine.insert((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) + (z1-z2)*(z1-z2));
  }

  // ispisujemo broj elemenata skupa
  cout << duzine.size() << endl;

  return 0;
}
```

Задатак: Провера Судоку загонетке

Судоку загонетка се решава тако што се у матрицу димензије 9 пута 9 уписују бројеви од 1 до 9 и то тако да су:

- елементи сваке врсте различити;
- елементи сваке колоне различити;
- када се матрица подели на 9 дисјунктних квадрата димензије 3 пута 3, елементи сваког тако добијеног квадрата су различити.

Написати програм који проверава да ли је дата Судоку загонетка исправно попуњена.

Опис улаза

Са стандрадног улаза се учитава матрица димензије 9 пута 9 која садржи бројеве од 1 до 9.

Опис излаза

На стандардни излаз исписати `da` ако матрица представља исправан Судоку тј. `ne` ако не представља.

Пример

Улаз	*Излаз*
9 1 3 4 2 7 5 8 6 6 8 7 9 1 5 3 2 4 2 5 4 6 8 3 1 7 9 4 7 9 1 3 2 6 5 8 1 6 2 5 9 8 7 4 3 5 3 8 7 6 4 2 9 1 3 4 5 8 7 1 9 6 2 7 2 6 3 4 9 8 1 5 8 9 1 2 5 6 4 3 7	da

Решење

Основно питање у решењу је како проверити да ли су сви елементи у датој врсти, колони или квадрату различити. Један начин да то одредимо је да прилиском обиласка елемената, елементе које смо видели чувамо у скупу. Приликом додавања новог елемента можемо проверити да ли се он већ јавља у скупу и ако се јавља, знамо да је у питању дупликат и да бројеви нису различити. Проверу припадности је могуће урадити и у склопу додавања, јер метода `insert` класе `set` враћа уређени бар чији је други елемент логичка вредност и то `true` ако је елемент убачен, а `false` ако није (јер већ постоји).

```
#include <iostream>
#include <set>

using namespace std;

bool vrstaOK(int tabla[9][9], int vrsta) {
  set<int> brojevi;
  for (int kolona = 0; kolona < 9; kolona++) {
    int broj = tabla[vrsta][kolona];
    if (!brojevi.insert(broj).second)
      return false;
```

```
  }
  return true;
}

bool kolonaOK(int tabla[9][9], int kolona) {
  set<int> brojevi;
  for (int vrsta = 0; vrsta < 9; vrsta++) {
    int broj = tabla[vrsta][kolona];
    if (!brojevi.insert(broj).second)
      return false;
  }
  return true;
}

bool kvadratOK(int tabla[9][9], int startVrsta, int startKolona) {
  set<int> brojevi;
  for (int v = 0; v < 3; v++) {
    for (int k = 0; k < 3; k++) {
      int broj = tabla[startVrsta + v][startKolona + k];
      if (!brojevi.insert(broj).second)
        return false;
    }
  }
  return true;
}

bool sudokuOK(int tabla[9][9]) {
  for (int vrsta = 0; vrsta < 9; vrsta++)
    if (!vrstaOK(tabla, vrsta))
      return false;

  for (int kolona = 0; kolona < 9; kolona++)
    if (!kolonaOK(tabla, kolona))
      return false;

  for (int startVrsta = 0; startVrsta < 9; startVrsta += 3)
    for (int startKolona = 0; startKolona < 9; startKolona += 3)
      if (!kvadratOK(tabla, startVrsta, startKolona))
        return false;

  return true;
}

int main() {
  int tabla[9][9];
  for (int vrsta = 0; vrsta < 9; vrsta++)
    for (int kolona = 0; kolona < 9; kolona++)
      cin >> tabla[vrsta][kolona];
  if (sudokuOK(tabla))
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

Задатак: Различите цифре

Написати програм којим се испитује да ли су све цифре у декадном запису датог природног броја различите?

Опис улаза

Са стандардног улаза уноси се природан број од 0 до $2 \cdot 10^9$.

Опис излаза

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 здесна налево, 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 – 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.

```
#include <iostream>

using namespace std;

bool sadrziCifru(int n, int c) {
  do {
    if (n % 10 == c)
      return true;
    n /= 10;
  } while (n > 0);
  return false;
}

bool razliciteCifre(int n) {
  while (n >= 10) {
    if (sadrziCifru(n / 10, n % 10))
      return false;
    n /= 10;
  }
  return true;
}

int main() {
  int n;
  cin >> n;
  cout << (razliciteCifre(n) ? "DA" : "NE") << endl;
  return 0;
}
```

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};`.

```
#include <iostream>

using namespace std;

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;
}

int main() {
  int n;
  cin >> n;
  cout << (razliciteCifre(n) ? "DA" : "NE") << endl;
  return 0;
}
```

Представљање скупа виђених цифара библиотечким скуповима битова

Програмски језици нуде и специфичне структуре података које омогућавају да се представи низ елемената типа `bool` на мало компактнији начин од обичног низа.

У језику C++ за то је могуће употребити колекцију `vector<bool>`, или, још боље, колекцију `bitset<10>`. Вектор димензије 10 се иницијализује тако да су му вредности `false` помоћу `vector<bool> skupCifara(10, false);`, док је колекција `bitset` таква да су јој подразумевано сви елементи постављени на вредност 0 тј. `false` и довољно је само увести декларацију `biset<10> skupCifara;`. И са колекцијом `vector` и `bitset` надаље се поступа као са обичним низом (елементима се приступа оператором индексног приступа `[]`).

```
#include <iostream>
#include <bitset>

using namespace std;

bool razliciteCifre(int n) {
  bitset<10> skupCifara;
  do {
    int cifra = n % 10;
    if (skupCifara[cifra])
      return false;
    skupCifara[cifra] = true;
    n /= 10;
  } while (n > 0);
  return true;
}

int main() {
  int n;
  cin >> n;
  cout << (razliciteCifre(n) ? "DA" : "NE") << endl;
  return 0;
}
```

Задатак: Фреквенција знака

Написати програм који чита једну реч текста коју чине само велика слова енглесне абецеде 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.

Опис улаза

У једној линији стандардног улаза налази се једна реч текста са не више од 20 слова.

Опис излаза

У првој линији стандарног излаза приказати слово које се најчешће појављује, а у другој линији стандардног улаза исписати и колико пута се појављује.

Пример 1

Улаз	*Излаз*
`POPOKATEPETL`	`P` `3`

Пример 2

Улаз	*Излаз*
`BACACB`	`B` `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 `A`, на позицији један слова `B` 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 `A` од кода тог карактера (на пример, кôд карактера `D` је 68, док је код карактера `A` 65, одузимањем се добија 3, што значи да се бројач карактера `D` налази на месту број 3 у низу).

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.

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.

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

int main() {
  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;
    }

  return 0;
}
```

Пресликавање карактера у њихов број појављивања могуће је остварити и библиотечким структурама података које представљају тзв. асоцијативне низове (мапе, речнике).

У језику C++ бисмо могли употребити `map<char, int>` или `unordered_map<char, int>`.

У мапи `m` Изразом `m[c]` се приступа броју појављивања карактера `c`. У језику C++ се изразом `m[c]++` може увећати број појављивања било да карактер `c` постоји у мапи или не (ако не постоји, овим се број појављивања поставља на 1).

Пошто се приликом коришћењем мапа тј. речника бројеви појављивања придружују само оним карактерима који се заиста јављају у речи (а не унапред свим карактерима), мало се штеди меморија. Ипак, ове структуре података су примереније за ситуације у којима није тако једноставно на основу кључа добити нумеричку вредност позиције и када је скуп допустивих кључева шири.

```
#include <iostream>
#include <string>
#include <algorithm>
#include <map>
using namespace std;

int main() {
  // 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 najveci broj pojavljivanja slova
  int maxPojavljivanja = 0;
  for (const auto& p : brojPojavljivanja)
    if (p.second > maxPojavljivanja)
      maxPojavljivanja = p.second;

  // ispisujemo slovo koje se prvo pojavljuje u reci sa tim brojem
  // pojavljivanja
  for (char c : rec)
    if (brojPojavljivanja[c] == maxPojavljivanja) {
      cout << c << endl
           << maxPojavljivanja << endl;
      break;
    }

  return 0;
}
```

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 исписује 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.

Опис улаза

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).

Опис излаза

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`

Решење

У првој фази пребројавамо појављивања свих малих и великих слова. За то можемо користити два низа са по 26 елемената (један за мала, један за велика слова), слично као у задатку Фреквенција знака. Када се цео текст учита и преброје се сва слова, одређујемо максимални број појављивања неког малог и максимални број појављивања неког великог слова (најједноставније, коришћењем библиотечке функције). На крају, пролазимо кроз сва слова азбуке редом и за свако проверавамо да ли је оно једно од слова које се јавило максимални број пута (тј. да ли је његов број појављивања једнак максималном). Ако јесте, исписујемо га.

```
#include <iostream>
#include <algorithm>
#include <cctype>

using namespace std;

int main() {
  // broj pojavljivanja malih slova a do z
  int mala_slova[26] = {0};
  // broj pojavljivanja velikih slova A do Z
  int velika_slova[26] = {0};

  // citamo liniju po liniju teksta
  string linija;
  while (getline(cin, linija)) {
    // analiziramo jedan po jedan karakter ucitane linije
    for (char c : linija) {
      // ako je malo slovo, brojimo ga
      if (islower(c))
        mala_slova[c - 'a']++;

      // ako je veliko slovo, brojimo ga
      if (isupper(c))
        velika_slova[c - 'A']++;
    }
  }

  // odredjujemo maksimalni broj pojavljivanja nekog malog slova
  int max_mala = 0;
  for (int x : mala_slova)
    max_mala = max(x, max_mala);
  // ako ima malih slova
  if (max_mala > 0) {
    // analiziramo sva mala slova redom i ispisujemo ona koja se
```

```
    // najcesce pojavljuju
    for (char c = 'a'; c <= 'z'; c++)
      if (mala_slova[c - 'a'] == max_mala)
        cout << c;
    cout << endl;
  }

  // odredjujemo maksimalni broj pojavljivanja nekog velikog slova
  int max_velika = 0;
  for (int x : velika_slova)
    max_velika = max(max_velika, x);

  // ako ima velikih slova
  if (max_velika > 0) {
    // analiziramo sva velika slova redom i ispisujemo ona koja se
    // najcesce pojavljuju
    for (char c = 'A'; c <= 'Z'; c++)
      if (velika_slova[c - 'A'] == max_velika)
        cout << c;
    cout << endl;
  }

  return 0;
}
```

Пошто се и за мала и за велика слова спроводи практично исти поступак, понављање кода се може избећи издвајањем помоћних функција.

```
#include <iostream>
#include <algorithm>
#include <cctype>

using namespace std;

// niz brojaca karaktera iz intervala karaktera [poc, kraj] se azurira
// na osnovu date liste
void prebrojKaraktere(char poc, char kraj, string s, int broj[]) {
  for (char c : s)
    if (poc <= c && c <= kraj)
      broj[c - poc]++;
}

// ispisujemo najcesce karaktere iz  intervala karaktera [poc, kraj]
// na osnovu datih podataka o njihovom broju pojavljivanja u tekstu
void ispisiNajcesceKaraktere(char poc, char kraj, int broj[]) {
  // odredjujemo maksimalni broj pojavljivanja nekog slova
  int Max = 0;
  for (char c = poc; c <= kraj; c++)
    Max = max(Max, broj[c]);

  // ako ima slova
  if (Max > 0) {
    // analiziramo sva slova redom i ispisujemo ona koja se
    // najcesce pojavljuju
    for (char c = poc; c <= kraj; c++)
      if (broj[c - poc] == max)
        cout << c;
    cout << endl;
  }
```

```
}



int main() {
  // broj pojavljivanja malih slova a do z
  int mala_slova[26] = {0};
  // broj pojavljivanja velikih slova A do Z
  int velika_slova[26] = {0};

  // citamo liniju po liniju teksta
  string linija;
  while (getline(cin, linija)) {
    // brojimo mala i velika slova
    prebrojKaraktere('a', 'z', linija, mala_slova);
    prebrojKaraktere('A', 'Z', linija, velika_slova);
  }

  // ispisujemo najcesca mala i velika slova
  ispisiNajcesceKaraktere('a', 'z', mala_slova);
  ispisiNajcesceKaraktere('A', 'Z', velika_slova);

  return 0;
}
```

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 Милици да одреди производ који је најчешће куповала током претходне године.

Опис улаза

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.

Опис излаза

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`
`kruska`
`jabuka`
`sljiva`
`hleb`
`mleko`
`jabuka`
`hleb`
	`hleb 3`

Решење

Бројање речи

Централно место у задатку је избројати појављивања сваке речи која се јавила на улазу. Овај задатак је донекле сличан задатку Фреквенција знака, једино што се уместо појединачних карактера броје речи. 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 података.

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. За сваку реч увећавамо њен број појављивања у мапи тј. у речнику.

Изразом `brojPojavljivanja[rec]` се приступа броју појављивања речи. У језику C++ се изразом `brojPojavljivanja[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).

```
#include <iostream>
#include <string>
#include <map>

using namespace std;

int main() {
  string s;
  map<string, int> brojPojavljivanja;
  while (cin >> s)
    brojPojavljivanja[s]++;

  int max = 0; string maxRec;
  for (auto it : brojPojavljivanja)
    if (it.second > max ||
        (it.second == max && it.first < maxRec)) {
      maxRec = it.first;
      max = it.second;
    }

  cout << maxRec << " " << max << endl;

  return 0;
}
```

Сортирање

Још један могући приступ решавању овог проблема је да се речи сортирају тако да сва појављивања исте речи буду узастопна. Пошто се не зна унапред број речи, најбоље их је учитати у неки динамички низ. У језику C++ то може бити `vector`.

Сортирање је најбоље урадити лексикографски, применом библиотечке функције. Након сортирања, тражимо дужину најдуже узастопне серије једнаких елемената.

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  string s;
  vector<string> reci;
```

```
  while (cin >> s)
    reci.push_back(s);

  sort(begin(reci), end(reci));

  int tekucaDuzina = 1;
  int maxDuzina = 1; string maxRec = reci[0];
  for (int i = 1; i < reci.size(); i++) {
    if (reci[i] == reci[i-1])
      tekucaDuzina++;
    else
      tekucaDuzina = 1;
    if (tekucaDuzina > maxDuzina) {
      maxDuzina = tekucaDuzina;
      maxRec = reci[i];
    }
  }

  cout << maxRec << " " << maxDuzina << endl;
  return 0;
}
```

Задатак: Речник

Напиши програм који имплементира српско-енглески речник. Прво се учитава списак речи и њихових превода, а затим коринсник од система тражи да му се преведу неке речи (било са српског на енглески, било са енглеског на српски).

Опис улаза

Са стандардног улаза се уноси број $n (1 \leq n \leq 50\,000)$ парова речи на српском и енглеском језику. Све речи на српском су међусобно различите и све речи на енглеском су међусобно различите. Након тога се уноси $m (1 \leq m \leq 50\,000)$ упита. Сваки упит садржи жељени језик (или `sr` или `en`) и затим реч на супротном језику коју треба превести.

Опис излаза

На стандардни излаз исписати све преводе. Ако за неку реч није унет превод, исписати `?`.

Пример

Улаз	*Излаз*
3	dog
cat macka	mis
dog pas	?
mouse mis	mouse
4	
en pas	
sr mouse	
sr lion	
en mis	

Решење

Као што име задатка каже, задатак се најлакше решава ако се употреби структура података речник (тј. мапа, асоцијативни низ). У једном речнику пресликавамо речи на српском у речи на енглеском, а у другом речи на енглеском у речи на српском језику.

```
#include <iostream>
#include <map>

using namespace std;

int main() {
```

```
  map<string, string> sr_en;
  map<string, string> en_sr;
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    string en, sr;
    cin >> en >> sr;
    sr_en[sr] = en;
    en_sr[en] = sr;
  }

  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    string jezik, rec;
    cin >> jezik >> rec;
    if (jezik == "en") {
      if (sr_en.count(rec))
        cout << sr_en[rec] << endl;
      else
        cout << "?" << endl;;
    } else if (jezik == "sr") {
      if (en_sr.count(rec))
        cout << en_sr[rec] << endl;
      else
        cout << "?" << endl;
    }
  }

  return 0;
}
```

Задатак: Изоморфне ниске

Две ниске су изоморфне ако се друга може добити неким 1-1 пресликавањем слова прве - сваком слову абецеде 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 `cilim` 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.

Опис улаза

Са стандардног улаза се учитавају две речи састављене од малих слова енглеске абецеде, свака у посебном реду.

Опис излаза

На стандардни излаз исписати `da` ако су речи изоморфне, тј. `ne` у супротном.

Пример 1

Улаз	*Излаз*
`filip` `leden`	`da`

Пример 2

Улаз	*Излаз*
`filip` `melem`	`ne`

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 абецеде (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 `chr` лако одређујемо као разлику кода тог карактера и карактера `a`.

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`.

```
#include <iostream>
#include <string>

using namespace std;

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;
}

int main() {
  string s, t;
  cin >> s >> t;
  cout << (izomorfne(s, t) ? "da" : "ne") << endl;
  return 0;
}
```

Директан начин да се представи пресликавање је да се употребе библиотечке колекције, чиме се добија мало читљивији текст програма.

У језику C++ се може употребити `map` из заглавља `<map>` или `unordered_map` из заглавља `<unordered_map>`. И скуп може бити представљен библиотечким колекцијама. У језику C++ се може употребити `set` из заглавља `<set>` или `unordered_set` из заглавља `<unordered_set>`.

```
#include <iostream>
#include <string>
#include <map>
#include <set>
```

```cpp
using namespace std;

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;
}

int main() {
  string s, t;
  cin >> s >> t;
  cout << (izomorfne(s, t) ? "da" : "ne") << endl;
  return 0;
}
```

Задатак: Рачуни

Потребно је симулирати банковни систем за k различитих корисника. Сваки корисник има рачун са почетним стањем 0. Потребно је подржати две врсте операција:

- `upit x` одређује колико постоји корисника чији рачун садржи тачно x динара
- `ime x` додаје x динара на рачун особе са именом `ime` (x може бити и негативно)

Написати програм који пордржава извршавање n оваквих операција.

Опис улаза

Са стандардног улаза се уносе бројеви n и k. Након тога се у n редова уноси по једна операција.

Опис излаза

За сваки упит (операцију првог типа) исписати одговор, сваки у засебном реду.

Пример

Улаз	*Излаз*

`6 4`
`marko 2`
`milan 5`
`dragana 4`
`upit 0`
`milan -1`
`upit 4`
	`1`
	`2`

Решење

Задатак се једноставно и лако може решити употребом две мапе тј. речника. Једна се користи да преслика име корисника у износ на његовом рачуну, а друга да преслика износ на рачуну у број рачуна који садрже тај износ.

```
#include <iostream>
#include <map>

using namespace std;

int main() {
  int n, m, x;
  cin >> n >> m;

  map<string, int> racun;
  map<int, int> brPojavljivanja;
  brPojavljivanja[0] = m;

  for (int i = 0; i < n; i++) {
    string s;
    cin >> s >> x;

    if (s == "upit")
      cout << brPojavljivanja[x] << endl;
    else {
      brPojavljivanja[racun[s]]--;
      racun[s] += x;
      brPojavljivanja[racun[s]]++;
    }
  }

  return 0;
}
```

4.5 Стек, ред, ред са приоритетом

Задатак: Упареност заграда

У изразу учествују следеће врсте заграда (,), {, }, [и]. Написати програм који проверава да ли су у унетом изразу заграде исправно упарене.

Опис улаза

Са стандардног улаза се уноси израз (цео у једном реду, без размака).

Опис излаза

На стандардни излаз исписати `da` ако су заграде исправно упарене тј. `ne` ако нису.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
`[3*(2+4)]*5`	`da`	`{3*(2+4)}*[5+7)+(4*5)`	`ne`

Решење

Учитану ниску обрађујемо карактер по карактер. Заграда која се затвара мора да одговара последњој загради која је отворена. Зато у сваком тренутку потребно је да знамо ниске које су отворене, а још нису затворене. Структура података у којој се заграде памте је таква да се нове отворене заграде додају на врх и скидају са врха – дакле, у питању је стек. Сваку отворену заграду додајемо на стек, а за сваку затворену заграду проверавамо да ли одговара оној коју скидамо са врха стека (ако је стек празан или ако је на врху нека друга врста отворене заграде од оне која је тренутно затворена, пријављујемо грешку). На крају све заграде морају бити упарене и стек мора бити празан.

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

bool uparena(char oz, char zz) {
  return (oz == '(' && zz == ')') ||
         (oz == '[' && zz == ']') ||
         (oz == '{' && zz == '}');
}

bool otvorena(char c) {
   return c == '(' || c == '[' || c == '{';
}

bool zatvorena(char c) {
   return c == ')' || c == ']' || c == '}';
}

bool proveriZagrade(const string& izraz) {
  stack<char> zagrade;
  for (char c : izraz) {
    if (otvorena(c))
      zagrade.push(c);
    else if (zatvorena(c)) {
      // zatvorena zagrada nije uparena
      if (zagrade.empty() || !uparena(zagrade.top(), c))
        return false;
      zagrade.pop();
    }
  }
  // sve zatvorene zagrade su uparene, pa je izraz u redu ako i samo
  // ako nema otvorenih zagrada koje nisu zatvorene
  return zagrade.empty();
}

int main() {
   string izraz;
   cin >> izraz;
   cout << (proveriZagrade(izraz) ? "da" : "ne") << endl;
   return 0;
}
```

Задатак: Историја веб-прегледача

Прегледач веба памти историју посећених сајтова и корисник има могућност да се враћа унатраг на сајтове које је раније посетио. Написати програм који симулира историју прегледача тако што се учитавају адресе посећених сајтова (свака у посебном реду), а када се учита ред у коме пише `back` прегледач се враћа на последњу посећену страницу. Ако се наредбом `back` вратимо на почетну страницу, исписати `-`. Ако се на почетној страници изда наредба `back`, остаје се на почетној страници. Програм треба да испише све сајтове које је корисник посетио.

Опис улаза

Са стандардног улаза се учитавају веб-адресе, свака у посебној линији, њих највише 1000.

Опис излаза

На стандардни излаз исписати редом сајтове који се посећују.

Пример

Улаз	*Излаз*
`http://www.google.com`	`http://www.google.com`
`http://www.rts.rs`	`http://www.rts.rs`
`back`	`http://www.google.com`
`http://www.petlja.org`	`http://www.petlja.org`
`http://www.matf.bg.ac.rs`	`http://www.matf.bg.ac.rs`
`back`	`http://www.petlja.org`
`back`	`http://www.google.com`
`back`	`-`
`back`	`-`

Решење

Историја прегледача се понаша као стек јер се елементи само могу скидати и постављати на један крај историје (као последње посећени) и са тог краја се могу и скидати.

```
stack<string> istorija;
string linija;
while (getline(cin, linija)) {
  if (linija == "back") {
    if (!istorija.empty())
      istorija.pop();
    if (!istorija.empty())
      cout << istorija.top() << endl;
    else
      cout << "-" << endl;
  } else {
    cout << linija << endl;
    istorija.push(linija);
  }
}
```

Задатак: Вредност постфиксног израза

Префиксна нотација се понекад назива и пољска нотација, а постфиксна нотација се понекад назива и обратна пољска нотација (енгл. reverse polish notation, RPN) у част пољског логичара Јана Лукашијевича, који ју је изумео. Она подразумева да се бинарни оператори уместо између операнада записују након њих. На пример, уместо `3 + 5`, писаћемо `3 5 +`. Написати програм који одређује вредност постфиксно записаног израза.

Опис улаза

Са стандардног улаза се учитава постфиксно записан израз који садржи једноцифрене бројеве и операторе `+` и `*` (без размака).

Опис излаза

На стандардни излаз исписати вредност учитаног израза.

Пример 1

Улаз	*Излаз*
`12+3*`	`9`

Објашњење

Постфиксно је записан израз `(1+2)*3`.

Пример 2

Улаз

```
11+2*345+*+
```

Излаз

```
31
```

Објашњење

Постфиксно је записан израз `(1+1)*2+3*(4+5)`.

Решење

Велика предност постфиксно записаних израза је то што им се вредност веома једноставно израчунава уз помоћ стека. Када наиђемо на број постављамо га на врх стека. Када наиђемо на оператор, скидамо две вредности са врха стека, примењујемо на њих одговарајућу операцију и резултат постављамо на врх стека. Вредност целог израза се на крају налази на врху стека.

На пример, за израз `34+5*` на стек постављамо `3`, затим `4`, након тога те две вредности уклањамо и постављамо `7`, затим постављамо `5` и на крају уклањамо `7` и `5` и мењамо их са `35`.

```
bool jeOperator(char c) {
  return c == '+' || c == '*';
}

int primeniOperator(char op, int op1, int op2) {
  int v = 0;
  switch(op) {
     case '+': v = op1 + op2; break;
     case '*': v = op1 * op2; break;
  }
  return v;
}

int vrednost(const string& izraz) {
   stack<int> st;
   for (char c : izraz) {
      if (isdigit(c))
        st.push(c - '0');
      else if (jeOperator(c)) {
         int op2 = st.top(); st.pop();
         int op1 = st.top(); st.pop();
         st.push(primeniOperator(c, op1, op2));
      }
   }
   return st.top();
}
```

Задатак: Сажимање серија узастопних једнаких елемената

Низ се трансформише тако што се првих k или више узастопних појављивања неког елемента бришу. Написати програм који одређује садржај низа након исцрпне примене ове трансформације (трансформација се примењује док год је то могуће).

Опис улаза

Са стандардног улаза се из првог реда учитава број k ($1 \leq k \leq 10^4$), из другог број n ($1 \leq n \leq 10^6$), а из трећег елементи низа раздвојени размаком.

Опис излаза

На стандардни излаз исписати елементе резултујућег низа, раздвојене размаком.

Пример

Улаз	*Излаз*

`3`
`20`
`1 1 2 2 2 2 1 3 4 4 5 5 5 4 4 3 2 1 1 1`
	`3 3 2`

Објашњење

Након уклањања четири двојке, три јединице постају узастопне и оне се уклањају. Након укањања три петице, четири четворке постају узастопне и оне се уклањају. На крају се уклањају три узастопне јединице.

Решење

Обрађиваћемо елементе низа слева надесно тражећи серије узастопних једнаких елемената. Чим се заврши довољно дугачка серија брисаћемо је са краја до тада обрађеног дела низа. То нам сугерише да је корисно да употребимо стек. Приликом додавања новог елемента проверавамо да ли је њиме завршена претходна серија једнаких и ако јесте и ако је та претходна серија довољно дугачка, уклањамо је са стека. Након тога додајемо текући елемент на стек.

Прикажимо садржај стека алгоритма на примеру избацивања серија дужине бар три из низа `4 1 1 2 2 2 1 3 5 5 5`.

- Обрађујемо елемент 4. Постављамо га на стек и добијамо `4 1`.
- Обрађујемо елемент 1. Постављамо га на стек и добијамо `4 1`.
- Обрађујемо елемент 1. Постављамо га на стек и добијамо `4 1 1`.
- Обрађујемо елемент 2. Серија јединица је завршена, али још није довољно дугачка да би се избацила. Постављамо двојку на стек и добијамо `1 1 2`.
- Обрађујемо елемент 2. Постављамо га на стек и добијамо `4 1 1 2 2`.
- Обрађујемо елемент 2. Постављамо га на стек и добијамо `4 1 1 2 2 2`.
- Обрађујемо елемент 2. Постављамо га на стек и добијамо `4 1 1 2 2 2 2`.
- Обрађујемо елемент 1. Серија двојки је завршена и довољно је дугачка да би се избацила. Постављамо јединицу на стек и добијамо `4 1 1 1`.
- Обрађујемо елемент 3. Серија јединица је завршена и довољно је дугачка да би се избацила. Постављамо тројку на стек и добијамо `4 3`.
- Обрађујемо елемент 5. Постављамо га на стек и добијамо `4 3 5`.
- Обрађујемо елемент 5. Постављамо га на стек и добијамо `4 3 5 5`.
- Обрађујемо елемент 5. Постављамо га на стек и добијамо `4 3 5 5 5`.
- Дошли смо до краја низа. Серија петица је завршена и довољно је дугачка да би се избацила.

Резултат је, дакле, `4 3`.

Да бисмо исписали завршни садржај низа, потребно је да испишемо елементе стека од дна ка врху. Зато уместо стека можемо употребити ред са два краја или двоструко повезану листу. У језику C++ то може бити `deque` или `list`.

Да бисмо уштедели меморију и време потребно да се провери да ли је завршена серија довољно дугачка, уместо понављања истих елемената на стек можемо стављати уређене парове који садрже елемент и његов број појављивања. Тада се додавање елемента врши тако што се провери да ли се на врху стека налази баш тај елемент, и ако се налази увећа се његов број појављивања, а у супротном се елемент дода на врх стека уз број појављивања постављен на један.

```
#include <iostream>
#include <deque>

using namespace std;

int main() {
  int k;
  cin >> k;
  // potreban nam je stek, ali posto cemo na kraju elemente steka
  // ispisivati od dna do vrha, koristimo red sa dva kraja
  deque<pair<int, int>> s;
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;

    // ako je serija zavrsena i dovoljno je dugacka, skidamo je sa steka
    if (!s.empty() && x != s.back().first && s.back().second >= k)
```

```
      s.pop_back();

    // dodajemo novi element na stek
    if (!s.empty() && s.back().first == x)
      s.back().second++;
    else
      s.emplace_back(x, 1);

  }

  // ako je poslednja serija dovoljno dugacka, skidamo je sa steka
  if (!s.empty() && s.back().second >= k)
    s.pop_back();


  // ispisujemo sadrzaj steka
  for (auto it : s)
    for (int i = 0; i < it.second; i++)
      cout << it.first << " ";
  cout << endl;

  return 0;
}
```

Задатак: Штампање

Штампач прима захтеве за штампање докумената. Сваки захтев садржи јединствени идентификатор документа (шестоцифрени цео број). Документи се штампају редом којим су послати на штампу. Напистаи програм који симулира рад штампача.

Опис улаза

Са стандардног улаза се учитавају линије. Линија може представљати или захтев за штампу документа (тада садржи шестоцифрени цео број) или команду за узимање одштампаног документа (линија тада садржи само цртицу -).

Опис излаза

На сваку команду за узимање одштампаног документа одговорити тако што ће се исписати редни број тог документа или `err` ако су сви одштампани документи већ раније узети.

Пример

Улаз	*Излаз*
123456	123456
654321	654321
-	111111
111111	err
-	222222
-	
-	
222222	
-	

Решење

Пошто се документи штампају истим редом којим се захтева њихова штампа, захтеве ћемо чувати у структури података ред. Кда стигне нови захтев, додаћемо га на крај реда. Када се тражи узимање одштампаног документа, узећемо онај који се налази на почетку реда, а ако је ред празан, пријавићемо грешку.

```
#include <iostream>
#include <queue>

using namespace std;
```

```
int main() {
  queue<int> zahtevi;
  string linija;
  while (getline(cin, linija)) {
    if (linija[0] == '-') {
      if (zahtevi.empty())
        cout << "err" << endl;
      else {
        cout << zahtevi.front() << endl;
        zahtevi.pop();
      }
    } else {
      zahtevi.push(stoi(linija));
    }
  }
  return 0;
}
```

Задатак: Последњих k линија

Написати програм који исписује k последњих линија учитаних са стандардног улаза.

Опис улаза

Са стандардног улаза се учитава број k ($1 \leq k \leq 100$), а затим једна по једна линија текста (њих највише 10^6).

Опис излаза

На стандардни излаз исписати последњих k линија (претпоставити да је увек учитано барем k линија).

Пример

Улаз	*Излаз*
`2`	`poslednjih k`
`ispisati`	`linija`
`poslednjih k`	
`linija`	

Решење

Једно решење је да се линије учитају у низ и да се након учитавања испише последњих k елемената низа. Пошто не знамо унапред колико ће линија бити, користимо низ који може да се проширује додавањем нових елемената. У језику C++ то може бити `vector`.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int k;
  cin >> k;
  vector<string> linije;
  string linija;
  while (getline(cin, linija))
    linije.push_back(linija);
  for (int i = linije.size() - k; i < linije.size(); i++)
    cout << linije[i] << endl;
  return 0;
}
```

Није потребно чувати све елементе истовремено у меморији, већ само последњих k. Уместо у низу, линије ћемо чувати у структури података која нам омогућава да учитавања наредне линије избацимо линију која учитана k корака пре тога. Пошто се линије избацују у истом редоследу у ком се додају, користићемо ред.

```
#include <iostream>
#include <string>
#include <queue>

using namespace std;

int main() {
  int k;
  cin >> k;
  queue<string> red;

  string linija;
  while (getline(cin, linija)) {
    red.push(linija);
    if (red.size() > k)
      red.pop();
  }

  while (!red.empty()) {
    cout << red.front() << endl;
    red.pop();
  }

  return 0;
}
```

Задатак: Ограничена историја прегледача

Прегледач веба памти последњих n сајтова и корисник има могућност да се враћа унатраг на сајтове које је раније посетио. Написати програм који симулира историју прегледача тако што се учитавају адресе посећених сајтова (свака у посебном реду), а када се учита ред у коме пише `back` прегледач се враћа на претходну посећену страницу. Ако се наредба `back` зада док је прегледач на почетној страници, исписати `-`, а прегледач остаје се на почетној. страници. Програм треба да за сваку команду испише сајт на коме је корисник после те команде.

Опис улаза

Са стандардног улаза се учитава број n ($1 \leq n \leq 1000$), а затим веб-адресе, свака у посебној линији.

Опис излаза

На стандардни излаз исписати редом сајтове који се посећују.

Пример

Улаз	*Излаз*
`3`	`http://www.google.com`
`http://www.google.com`	`http://www.rts.rs`
`http://www.rts.rs`	`http://www.google.com`
`back`	`http://www.petlja.org`
`http://www.petlja.org`	`http://www.matf.bg.ac.rs`
`http://www.matf.bg.ac.rs`	`http://www.rg.edu.rs`
`http://www.rg.edu.rs`	`http://www.matf.bg.ac.rs`
`back`	`http://www.petlja.org`
`back`	`-`
`back`	

Решење

Задатак решавамо коришћењем реда са два краја. Посећене сајтове додајемо на један крај реда и при наредби `back` их скидамо са тог краја. Када се попуни n сајтова, пре додавања новог избацујемо онај наранније посећен (њега избацујемо са почетка реда). У језику C++ за ред са два краја можемо употребити `deque`.

```
#include <iostream>
#include <string>
#include <deque>

using namespace std;

int main() {
  int n;
  cin >> n >> ws;
  string linija;
  deque<string> istorija;
  while (getline(cin, linija)) {
    if (linija == "back") {
      if (!istorija.empty())
        istorija.pop_back();
      if (!istorija.empty()) {
        cout << istorija.back() << endl;
      } else {
        cout << "-" << endl;
      }
    } else {
      cout << linija << endl;
      if (istorija.size() == n)
        istorija.pop_front();
      istorija.push_back(linija);
    }
  }
  return 0;
}
```

Задатак: Збир k најбољих

Ученик је радио n задатака и за сваки задатак је добио одређени број поена. Одредити збир поена на k задатака које је најбоље урадио.

Опис улаза

У првој линији стандардног улаза унети природан број n ($1 \leq n \leq 10^6$) - број задатака које је ученик радио, у другој природан број k ($1 \leq k \leq n$) - број задатака које је најбоље урадио, а затим у следећих n линија број поена које је добио на задацима.

Опис излаза

Укупан број поена које је освојио на k најбоље оцењених задатака.

Пример

Улаз	*Излаз*
10 3 15 80 25 60 10 20 50 45 40 30	190

Решење

Сортирање целог низа

Задатак се може решити и тако што ће се низ сортирати библиотечком функцијом за сортирање. Ако се низ сортира нерастуће, онда је након сортирања потребно сабрати првих k елемената низа, а ако се сортира неопадајуће, онда је након сортирања потребно сабрати последњих k елемената низа (сортирање неопадајуће је обично подразумевани начин сортирања и лакше га је реализовати).

Може се приметити да овакав алгоритам непотребно губи време прецизно одређујући редослед тј. сортирајући елементе који су мањи од првих k као и одређујући прецизан редослед првих k елемената (да би се сабрало првих k елемената они не морају бити поређани, већ је само потребно на почетак низа (или на крај) довести првих k елемената, а иза њих (или испред) поставити остале елементе тј. раздвојити те две групе елемената, при чему је редослед елемената унутар сваке од група ирелевантан.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <functional>  // zbog greater<int>() u pozivu funkcije sort
using namespace std;

int main() {
  ios_base::sync_with_stdio(false);

  // ucitavamo ulazne podatke
  int n, k;
  cin >> n >> k;

  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  // sortiramo niz nerastuce
  sort(begin(a), end(a), greater<int>());

  // sabiramo prvih k elemenata niza
  int s = 0;
  for (int i = 0; i < k; i++)
    s += a[i];

  // ispisujemo rezultat
  cout << s << endl;
}
```

Ред са приоритетом Највећих k до сада виђених елемената низа можемо чувати у структури података која нам омогућава да пронађемо најмањи елемент у њој и да га евентуално заменимо оним који је тренутно учитан (ако је тренутно учитани елемент већи од њега). Идеална структура за то је хип тј. ред са приоритетом.

Ред са приоритетом у језику C++ можемо добити помоћу `priority_queue`. Елементе у ред можемо убацити методом `push`. Елемент који је најмањи можемо очитати методом `top` и избацити методом `pop`.

На почетку ред попуњавамо са k првих учитаних елемената, а затим сваки наредни учитани елемент поредимо са најмањим у реду и ако је већи од њега, најмањи избацујемо, а учитани елемент убацујемо.

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

int main() {
  ios_base::sync_with_stdio(false);

  int n, k;
  cin >> n >> k;

  // red sa prioritetom koji cuva k najvecih elemenata koristi se
  // min-hip, koji omogucava brzo uklanjanje najmanjeg elementa
  priority_queue<int, vector<int>, greater<int>> pq;

  // ucitavamo prvih k elemenata i ubacujemo ih u red
  for (int i = 0; i < k; i++) {
    int x;
    cin >> x;
    pq.push(x);
  }

  // ucitavamo preostale elemente
  for (int i = k; i < n; i++) {
    int x;
    cin >> x;
    // ako je ucitani element veci od najmanjeg trenutno u redu
    // izbacujemo taj najmanji i menjamo ga ucitanim
    if (x > pq.top()) {
      pq.pop();
      pq.push(x);
    }
  }

  // izbacujemo elemente iz reda racunajuci njihov zbir i ispisujemo ga
  int s = 0;
  while (!pq.empty()) {
    s += pq.top();
    pq.pop();
  }
  cout << s << endl;

  return 0;
}
```

Задатак: Распоредјивање послова

Постоји листа задатака које процесор треба да изврши. За сваки задатак познато је најраније време почетка, трајање и приоритет. Процесор увек започиње извршавање оног задатка који има највећи приоротет међу свим задацима који могу бити започети (оним задацима чије је најраније време почетка достигнуто). Ако два задатка имају исти приоритет, извршава се онај који је први додат у листу. Задатак који је започет извршава

се до краја, чак и ако се у међувремену појави задатак с већим приоритетом.

Опис улаза

Са стандардног улаза се учитава број задатака n ($1 \leq n \leq 10^5$). Затим следи n линија, где је свака линија садржи три цела броја:

- најраније време почетка задатка (цео број $0 \leq s \leq 10^9$),
- трајање задатка (цео број $1 \leq d \leq 10^9$),
- приоритет (цео број $1 \leq p \leq 10^9$).

Опис излаза

На стандардни излаз исписати задатке у редоследу у ком ће се извршавати. За сваки задатак исписати редни број и време у ком почиње његово извршавање.

Пример

Улаз	*Излаз*
6	0 0
0 5 10	2 5
2 4 15	1 7
3 2 20	4 11
10 3 5	3 14
8 3 10	5 20
20 8 30	

Објашњење

- У тренутку 0 може почети извршавање само посла 0. Он траје 5 јединица времена.
- У тренутку 5 се могу извршавати послови 1 и 2. Посао 2 има већи приоритет, па се он извршава током 2 јединице времена.
- У тренутку 7 се може извршавати још само посао 1. Он траје 4 јединице времена.
- У тренутку 11 се могу извршавати послови 3 и 4. Посао 4 има већи приоритет. Његово извршавање траје 3 јединице.
- У тренутку 14 се може извршавати посао 3, који траје 3 јединице времена.
- У тренутку 17 нема послова који се могу извршавати, па се чека време 20, када се извршава последњи посао број 5.

Решење

Одржаваћемо текуће време и скуп послова које је могуће у том тренутку започети (то су послови чије је најраније време почетка достигнуто, али нису до сада извршени). Од свих њих потребно је изабрати онај који има највећи приоритет и избацити га из тог скупа. Да бисмо ефикасно могли одредити које послове је могуће додати у скуп, сортираћемо низ свих послова по времену почетка. Скуп, дакле, треба да подржи додавање нових послова, одређивање оног са највећим приоритетом и његово избацивање из скупа. Ове три операције јасно указују на то да послове треба чувати у реду са приоритетом. Сваки посао ћемо представити структуром и дефинисаћемо оператор `<` којим се пореде два посла (прво по приоритету, а ако је приоритет исти, онда по редоследу додавања).

Тренутно време креће од нуле. У сваком тренутку све послове чије је време почетка достигнуто додајемо у ред са приоритетом. Након тога, ако ред са приоритетом није празан, узимамо посао из реда са приоритетом, и обрађујемо га. Након тога увећавамо тренутно време за трајање тог посла. Ако је ред са приоритетом празан, а има још послова који нису обрађени, померамо тренутно време почетка на време почетка првог таквог посла.

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>

using namespace std;

struct Zadatak {
  long long pocetak, trajanje, prioritet, redniBroj;
```

```
  // poredjenje dva zadatka (koristi se implicitno u redu sa prioritetom)
  bool operator<(const Zadatak& drugi) const {
    // ako je jednak prioritet, prednost imaju ranije dodati poslovi
    if (prioritet == drugi.prioritet)
      return redniBroj > drugi.redniBroj;
    // prvo se izvrsavaju poslovi veceg prioriteta
    return prioritet < drugi.prioritet;
  }
};

int main() {
  int n;
  cin >> n;

  // ucitavamo sve zadatke
  vector<Zadatak> zadaci(n);
  for (int i = 0; i < n; ++i) {
    cin >> zadaci[i].pocetak >> zadaci[i].trajanje >> zadaci[i].prioritet;
    zadaci[i].redniBroj = i; // cuvamo i redni broj dodavanja zadatka
  }

  // sortiramo zadatke po vremenu pocetka
  sort(zadaci.begin(), zadaci.end(), [](const Zadatak& a, const Zadatak& b) {
                                       return a.pocetak < b.pocetak;
                                     });

  // red sa prioritetom u kojem cuvamo zadatke koji traju u tekucem
  // trenutku
  priority_queue<Zadatak> pq;
  long long trenutnoVreme = 0;
  int i = 0;
  while (i < n || !pq.empty()) {
    // dodajemo sve zadatke koji su zapoceli pre trenutnog vremena u red
    while (i < n && zadaci[i].pocetak <= trenutnoVreme) {
      pq.push(zadaci[i]);
      i++;
    }

    // ako red nije prazan
    if (!pq.empty()) {
      // izvrsavamo zadatak sa najvecim prioritetom
      Zadatak trenutniZadatak = pq.top();
      pq.pop();
      cout << trenutniZadatak.redniBroj << " " << trenutnoVreme << endl;
      // pomeramo trenutno vreme na trenutak kada je taj zadatak zavrsen
      trenutnoVreme = trenutnoVreme + trenutniZadatak.trajanje;
    } else if (i < n) {
      // ako nema slobodnih zadataka pomeramo trenutno vreme na
      // pocetak narednog zadatka
      trenutnoVreme = zadaci[i].pocetak;
    }
  }
  return 0;
}
```