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 постављеен 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 потрених 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

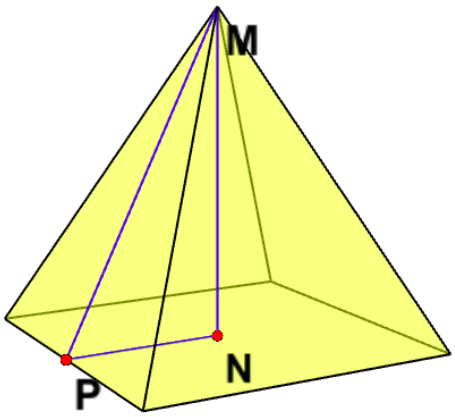
Улаз	*Излаз*
2	520
1.5	
72	

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, порвршина 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 променљивој.

```
#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`	`-236.15` `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 1830	5.63

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 бачна 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>
#include <cmath>

#define _USE_MATH_DEFINES

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.

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

Задатак: 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 11	1 11

Пример 2

Улаз	*Излаз*
15 16	2 1

Пример 3

Улаз	*Излаз*
15 103	7 13

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).

```
#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

Аутор: Филип Марић

Такмичење: Ревијално такмичење 2019/2020., V и VI разред, 1. задатак

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.

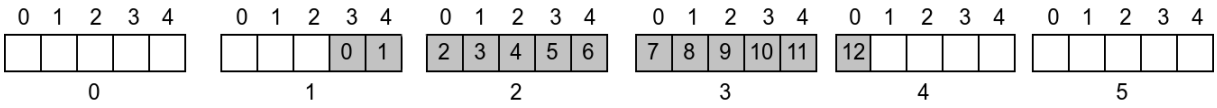
```
#include <iostream>

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

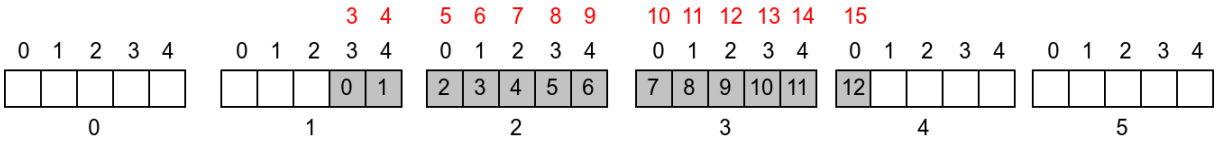
Улаз	*Излаз*
5	3
1	2
3	
9	

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 $s_n = (s_0 + n)$, 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.

```
#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).

```
#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	0:0:2
59	
59	
0	
0	
1	

Пример 2

Улаз	*Излаз*
0	23:59:58
0	
1	
23	
59	
59	

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 $2,54$ cm. Написати програм који врши прерачунавање између ових и метричнких 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 како бројевне типове (`int`, `double`, ...) и 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 >= 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 постоји троугао са датим страницама, иначе не постоји такав троугао. Бројеви a, b и c су по претпоставци задатка позитивни, тако да тај услов није неопходно посебно проверавати.

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

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		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`5 3` `1 7`	`da`	`5 3` `1 8`	`ne`	`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` ак 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 супрорном 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 ни једна 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 промењива. 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>
```

```cpp
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 дугпругаши трче трке дуже од тога. Написати програм којим се на основу дужине трке у којој атлетичар учествује одређује категорију 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		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
`400`	`kratko`	`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 температуре 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 средњепругач;
- 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 врло добар успех тај услов не проверавамо већ само да ли је просек већи или једнак 3,5. На исти начин настављамо по потреби провере за добар и довољан успех, и на крају ако ниједан услов није испуњен ученик има недовољан успех.

```
#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 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;
```

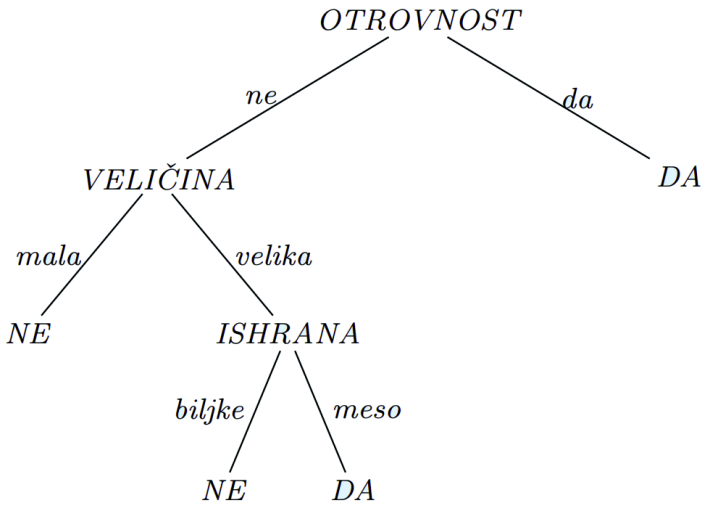
```
    else
      cout << 'j' << endl;
  } else if (mesec < 12)
    cout << 'j' << endl;
  else if (mesec == 12) {
    if (dan < 21)
      cout << 'j' << endl;
    else
      cout << 'z' << endl;
  }
  return 0;
}
```

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`.

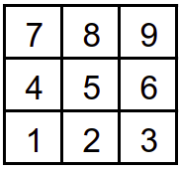
```cpp
#include <iostream>
#include <string>

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 је дужину странице означимо са $a = 100$, тада су x координате пиксела који припадају врсти 0 између 1 и a, координате пиксела који припадају врсти 1 су између $a + 1$ и $2a$, а врсти 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 $0x + c = 0$. 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. Постоје различити начини да се то уради.

Торке и библиотечко лексикографско поређење

Могуће је искористити библиотечку подршку за рад са n-торкама и њихово поређење које се по правилу врши лексикографски. Ако датуме представимо тројкама бројева у којима је на првом месту година, на другом месец, а на трећем дан, тада лексикографски поредак ових тројки одговара уобичајеном поретку датума. У језику C++ торке се представљају типом `tuple` и лексикографски се могу поредити коришћењем релацијских оператора `<`, `>`, `<=`, `>=`,

```
#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 је 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 45 22 47	1

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 подподверзјиа 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;
  return 0;
}
```

Задатак се може решити и експлицитним гранањем.

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

```cpp
#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.

```cpp
#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

Улаз	*Излаз*
-3	5
5	

Пример 2

Улаз	*Излаз*
2	3
5	

Пример 3

Улаз	*Излаз*
-3	0
-1	

Решење

Један начин да одредимо дужину дела интервала $[a, b]$ који лежи на позитивном делу праве је да применимо технику проналажења пресека два интервала (приказану у задатку Интервали), при чему је први интервал $[a, b]$, а други је $[0, +\infty)$. Пресек одређујемо тако што одредимо већи од два почетка (то је број $\max(a, 0)$ и леви од два краја (пошто је један крај $+\infty$ то је број b), и затим ако је крај десно од почетка та два броја одузмемо, док је у супротном резултат нула.

```
#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 промењиву 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 промењиве. Затим поредимо вредност те промењиве са цифром стотина и ако је цифра стотина мања, ажурирамо вредност те промењиве. 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

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 $a, a+dx, a+2\cdot dx, \ldots, a+(n-1)\cdot dx$, 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	203.0.113.15
01/10/2024	1
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	

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 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;
}
```