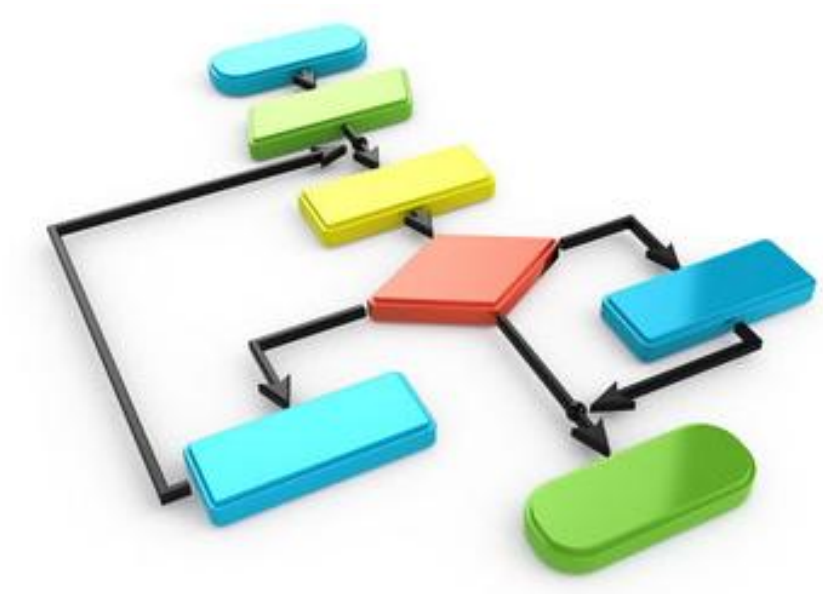
Liceul Teoretic Grigore Moisil



Algoritmi elementari în C++

Îndrumar de laborator

Autor: Spătaru Mihaela

Cuprins

Cuprins 2
1. Algoritmul 3
2. Structura liniară 4
3. Structura alternativă 6
3. 1 Aplicaţii cu structurile liniară şi alternativă 6
4. Fişiere 12
5. Structuri repetitive 15
5.1 Instrucţiunea while 15
5.2 Algoritmi elementari 16
5.3 Instrucţiunea for 24
6. Secvenţe sau şiruri de numere 30
7. Lucrul cu caractere 36
8. Instrucţiunea switch 37
Probleme date la olimpiadele de informatică 39
Bibliografie 47
Anexa 1 48
Anexa 2 49

1. Algoritmul

O **instrucţiune** este o comanda de baza prin care îi transmitem calculatorului să facă o singura acţiune sau operaţie.

Combinând mai multe instrucţiuni vom obţine un **algoritm**.

Aşadar, prin **algoritm** vom înţelege o succesiune finită de paşi, realizaţi într-o ordine bine definită, pentru ca, pornind de la anumite date cunoscute, numite **date de intrare**, să obţinem rezultatele dorite, numite **date de ieşire**.

La începutul anilor ’70, prelucrările cu ajutorul calculatorului au devenit tot mai dezvoltate, iar programele din ce în ce mai mari şi mai complicate. Chiar şi autorii acestor programe au început sa aibă probleme în a le înţelege, depana şi modifica ulterior. Pentru a ieşi din acea “*criza*” s-a impus ca programele să fie scrise sistematic, respectându-se anumite reguli, pentru a se obţine nişte programe mai clare. Astfel a apărut *programarea structurata*, care reprezintă un mod de concepere a programelor potrivit unor reguli bine stabilite, utilizându-se un set redus de *tipuri de structuri de control*.

La baza programării structurate sta **teorema lui Bohm §i Jacopini,** conform căreia orice algoritm poate fi compus din numai trei structuri de control:

- Structura secvenţiala (liniară);
- Structura alternativă;
- Structura repetitiva cu trei variante:
  - Structura repetitiva cu test iniţial;
  - Structura repetitiva cu test final;
  - Structura repetitiva cu număr cunoscut de paşi (sau structura repetitivă cu contor).

Structura generală a unui program C++

Structura unui program C++ este următoarea:

```
//declararea headerelor
#include <iostream>
using namespace std;
//declararea variabilelor globale
....
//programul principal
int main( )
{
//declararea variabilelor locale
// instrucţiunile programului
.....
return 0;
} //aici se încheie programul
```

2. Structura liniară

Structura liniară reprezintă un grup de operaţii sau instrucţiuni care se executa în ordinea scrierii lor.

Instrucţiunea declarativă

Printr-o instrucţiune declarativă se pot declara identificatori (variabile) de un anumit tip. Identificatorii pot fi variabile, dar vom vedea mai târziu că pot fi şi funcţii.

Sintaxa este:

```
tip_de_date  lista_variabile ;
```

unde

tip_de_date poate fi orice tip de dată C++ corect (int, double, unsigned, etc.), iar lista_variabile este alcătuită din cel puţin un identificator. Dacă sunt mai mulţi, se vor separa prin caracterul virgulă ,.

Exemple:

```
int  x,  y , z;
double a;
char  s;
```

Instrucţiuni de citire şi scriere

Printr-o instrucţiune de citire se pot introduce de la tastatură sau din fişier valorile variabilelor iar prin instrucţiunea de criere, se afişează pe ecran sau în fişier valorile variabilelor

Sintaxa este:

```
cin  >>  variabilă; // citire din consolă (de la tastatură)
cout << variabilă; //scriere în consolă (pe ecran)
```

Exemple:

```
cin>>x; //instrucţiunea de citire
cout << x; //instrucţiunea de afişare
cin>>a>>b>>c;
cout<<a<<"  "<<b<<" "<<c;
```

Notă: În C++, operaţiile de intrare/ieşire folosind fluxuri sunt şi ele operaţii. În exemplul de mai sus, cout, cin, x, a, b, c sunt operanzii, iar << este operatorul inserator şi >> este operatorul extractor.

Instrucţiunea *de atribuire*

Instrucţiunea *de atribuire* este cel mai frecvent folosit tip de instrucţiune dintr-un program C++. O expresie devine instrucţiune dacă este urmată de ;.

Sintaxa:

```
variabilă=expresie;
```

Exemple:

```
x = 2;// variabilei x i se atribuie valoarea 2
x =x+1;//variabila x se incrementează cu 1
x = 24/a + 100 / (c * c * c + b / (d * d + e * e)); /* se evaluează mai întâi expresia din partea dreaptă
                                                      şi valoarea rezultată este atribuită variabilei x */
```

Atenţie! *Operaţia de atribuire are sensul de la dreapta la stânga!*

Instrucţiunea compusă

Instrucţiunea compusă sau *blocul* este o grupare de declaraţii şi instrucţiuni închise între acolade {}. Ele au fost introduse cu scopul de a folosi mai multe instrucţiuni acolo unde sintaxa cere o singură instrucţiune. Instrucţiunea compusă sau blocul sunt echivalente sintactic cu o singură instrucţiune.

Blocul determină şi un domeniu de vizibilitate pentru identificatori. Mai precis, variabilele declarate într-un bloc vor fi golite de valori la terminarea acestuia.

Exemple:

```
#include <iostream>
using namespace std;
int main(){
int x = 4;
 {
int  x = 10;
cout << x << endl; // se va afisa 10
}
cout << x << endl; // se va afisa 4
}
```

Exemplu divizibilitate

Se citesc trei numere naturale, a, b şi k. Să se afişeze numărul de elemente divizibile cu k în intervalul [a, b] (inclusiv a şi b).

```
#include <iostream>
using namespace std;
int main()
{
        int a,b,k. n;//declararea variabilelor
        cin>>a>>b;//citirea variabilelor
                n=b/k -(a-1)/k;
        cout<<n;//afişarea numărului
}
```

Interschimbarea valorilor a două variabile

Uneori, în algoritmi, este necesar ca două variabile, să spunem **a** şi **b** să îşi schimbe între ele valorile, astfel încât valoarea care era la început în **a** să se afle în final în **b** şi valoarea care era la început în **b** să se afle în final în **a.** Această operaţie se cheamă interschimbare.

Pentru a înţelege mai bine soluţia vom face o analogie cu o altă problemă: să presupunem că avem două pahare **a** şi **b**, paharul **a** plin cu apă şi paharul **b** plin cu lapte. Să presupunem că vrem să schimbăm conţinutul paharelor astfel încât paharul **a** să conţină în final lapte şi paharul **b** să conţină în final apă. Este clar că nu putem rezolva problema fără a ne folosi de un al treilea pahar, gol, să îi spunem *aux*, de la *auxiliar*. Pentru a face interschimbul putem vărsa paharul **a** în paharul **aux**, apoi paharul **b** în paharul **a** şi în final paharul **aux** în paharul **b**.

Similar, pentru a interschimba variabilele **a** şi **b** vom folosi o a treia variabilă **aux**:

```
aux = a;
a = b;
b = aux;
```

3. Structura alternativă

În anumite situații, este necesară executarea unor instrucțiuni în cadrul unui program numai în anumite condiții. Numită și **structură de decizie**, structura alternativă permite rezolvarea unor asemenea situații.

Instrucțiunea if este cea mai utilizată structură alternativă.

Sintaxa ei are două forme:

Varianta 1

```
if ( condiție ) {
Instrucțiuni_1; }
else
  { Instrucțiuni_2; }
```

Varianta 2

```
if ( condiție ) {
  Instrucțiuni; }
```

Mod de execuție:

Instrucțiunea if se execută în felul următor:

- se evaluează condiția
- dacă valoarea ei este adevărată
  - se execută Instrucțiuni_1;
- dacă valoare expresiei este falsă
  - dacă există clauza else
    - se execută Instrucțiuni_2;

3. 1 Aplicații cu structurile liniară și alternativă

Următoarea secvență decide dacă un număr întreg citit este par sau nu:

```
int x;
cin >> x;
if(x % 2 == 0){
        cout << x << " este par";}
else
        {cout << x << " este impar";}
```

Următoarea secvență testează egalitatea cu 0 a unei expresii în forma scurtă, fără a folosi operatorii relaționali:

```
if ( n ){
        cout << n << " este nenul";}
else
        {cout << n << " este nul"; }
```

Instrucţiunea compusă

Instrucţiunea **if** permite câte o singură instrucţiune pe fiecare ramură. Ce facem dacă avem mai multe instrucţiuni pe o ramură? Folosim instrucţiunea compusă, folosind acolade.

Exerciţiu: ecuaţia de gradul 1. Fie ecuaţia $a \cdot x = b$

Să se calculeze x. **Atenţie**! Ecuaţia poate avea multiple soluţii, o soluţie sau niciuna!

```
#include <iostream>
using namespace std;
int main()
{       int a, b, x;//declararea variabilelor
        cin>>a>>b;//citirea variabilelor
        if ( a == 0 )
                if ( b == 0 )                   // a==0 and b==0
        cout<< "x poate lua orice valoare" ;
  else                                          // a==0 and b!=0
        cout<< "x nu are valori" ;
  else                                          //a!=0 and b!=0
  {     x = b / a;
        cout<< x ;
   }

return 0;

}
```

An bisect

Să se spună dacă un an este bisect .

```
#include <iostream>
using namespace std;
int main()
{         int a;
          cin>>a ;
 if ( a % 4 == 0 )
  if ( a % 100 == 0 )
   if ( a % 400 == 0 )
        cout<<"da";// a este divizibil cu 400
   else
        cout<<"nu"; // a este divizibil cu 100
  else
         cout<<"da"; // este divizibil cu 4
   else
         cout<<"nu";// an obişnuit

 return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{         int a;
          cin>>a ;
       if ( a % 4 >0 )
          cout<<"an obişnuit";
   else
       if ( a % 100 > 0 )
          cout<<" an bisect ";
   else
       if ( a % 400> 0 )
                cout<<"an obişnuit";
  else
          cout<<"an bisect";

 return 0;
}
```

Maximul a trei numere

1. Se citesc trei numere: a, b şi c. Să se afişeze valoarea maximă.

```
#include <iostream>
using namespace std;
int main()
{
        int a,b,c,max;//declararea variabilelor
        cin>>a>>b>>c;//citirea variabilelor
        max=a;//iniţializarea maximului cu primul număr
                if (b>max) {
                max=b;
                }
                if (c>max) {
                max=c;
                }
        cout<<maxx;//afişarea maximului
return 0;
}
```

Sortarea a trei numere

Se citesc trei numere a, b şi c, să se afişeze în ordine crescătoare. Iată o soluţie simplă, bazată pe interschimbarea variabilelor. Pentru aceasta vom interschimba două variabile dacă ordinea este greşită. Iată programul C++:

```
#include <iostream>
using namespace std;
int main() {
         int a, b, c, aux;      //declararea variabilelor
        cout<<"a=  "; cin>>a; // citirea variabilelor
        cout<<"b=  "; cin>>b;
        cout<<"c=  "; cin>>c;
                if ( a > b ) {
                aux = a;          // algoritm de interschimbare
                a = b;
                b = aux;  }
         if ( b > c ) {
                 aux = b;      // algoritm de interschimbare
                 b = c;
                c = aux; }
        if ( a > b ) {
                 aux = a;     // algoritm de interschimbare
                a = b;
                b = aux; }

 cout<<a<<"  "<<b<<" "<<c; // afişarea variabilelor

return 0;
}
```

Problema Suma_B_Numere – Pbinfo

Se dau 2 numere naturale, a şi b. Să se determine dacă a se poate scrie că suma de b numere naturale consecutive.
Date de intrare: Programul citeşte de la tastatură cele 2 numere a şi b.
Date de ieşire: Programul va afişa pe ecran numărul mesajul DA dacă a se poate scrie că suma de b numere naturale, iar NU în caz contrar.
Restricţii şi precizări : 1 ≤ a ≤ 100.000.000; 1 ≤ b ≤ 25.000
Exemplu:

Intrare	Ieşire	Explicaţie
12 3	DA	12 = 3 + 4 + 5 Pentru ca răspunsul sa fie DA trebuie ca a = x + (x + 1) + (x + 2) + ... + (x + b - 1) a = b*x + 1 + 2 + ... + (b - 1) a = b*x + ((b - 1) * b) / 2 b*x = a - ((b - 1) * b) / 2 x = (a - ((b - 1) * b) / 2) / b şi x trebuie sa fie întreg, deoarece aceasta este primul termen din suma de numere consecutive. Aşadar trebuie să testăm dacă restul împărţirii de mai sus este 0.

Rezolvare

```
#include <iostream>
using namespace std;
 int main()
{
  int a,b;
  cin>>a>>b;
  if((a-((b-1)*b)/2)%b==0)
     cout<<"DA";
  else
     cout<<"NU";
  return 0;
}
```

Cifra de control

Se citeşte un număr natural n. Să se scrie un algoritm prin care să se calculeze şi să se afişeze cifra de control al lui n. Cifra de control se obţine calculând suma cifrelor lui n, dacă suma este mai mare ca 10 vom calcula din nou suma cifrelor, s.a.m.d., până când se obţine o singură cifră.

Exemple: Pentru n=1234, se va afişa 1. Pentru n =92979, se va afişa 9.

```
int main()
{       int n; cin>>n;
        if(n==0)
                cout<<n;
        else if ( n%9==0 )
                cout<<9;
        else
                cout<<n%9;
        return 0;
}
```

Exersaţi:

Rezolvaţi următoarele probleme de pe www.pbinfo.ro:
Nota (#832), Interval2 (#469), 5numere (#559), calcul (#451), Minciuna (#1358).

Problema Gresie de pe Pbinfo

Se consideră o încăpere de formă dreptunghiulară cu dimensiunile a şi b. Încăperea trebuie pavată cu gresie de formă pătratică, de dimensiune d. Ştiind că o bucată de gresie se poate folosi întreagă sau tăiată, să se determine numărul minim de bucăţi de gresie sunt necesare pentru pavarea încăperii.

Date de intrare

Programul citeşte de la tastatură numerele naturale a, b, d.

Date de ieşire

Programul afişează pe ecran numărul minim de bucăţi de gresie necesare pentru pavarea încăperii.

Restricţii şi precizări

- $1 \le d \le a, b \le 32000$
- din motive estetice, pentru fiecare piesă de gresie tăiată se foloseşte o bucată întreagă.

Intrare

12 17 3

Ieşire

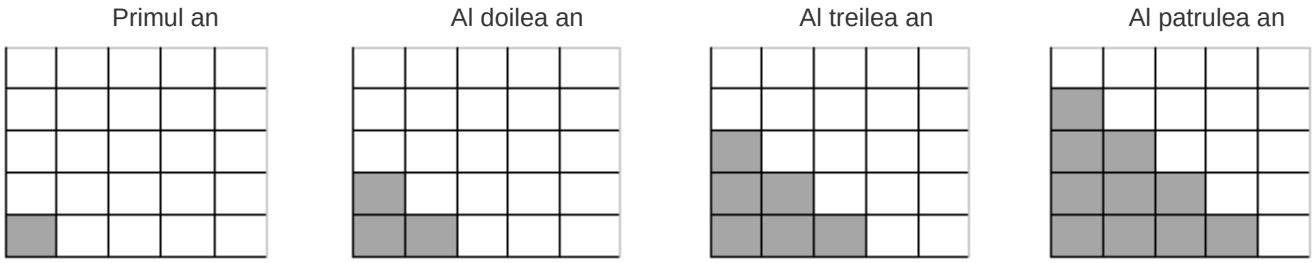
24

```
        #include<iostream>
        using namespace std;
int main()
{       int a, b, d;
        cin>>a>>b>>d;
        int L, l;
        if(a%d==0)
        L=a/d;
  else
        L=a/d+1;
        if(b%d==0)
        l=b/d;
  else
        l=b/d+1;
        cout<<l*L;
return 0;
}
```

Problema Bacteria de pe Pbinfo

O echipă de arheologi a descoperit o hartă străveche a Ţinutului de Nord, care era locuit de o civilizaţie condusă după reguli matematice foarte riguroase. Conform acestei hărţi, Ţinutul de Nord era împărţit în n rânduri a câte m comitate, fiecare comitat ocupând o suprafaţă pătrată de un hectar.

Însă descoperirile au mai arătat că această civilizaţie a fost atacată de la sud-vest de o bacterie periculoasă, ce a acţionat astfel: în primul an, a infectat comitatul din colţul din stânga jos al hărţii, în al doilea an a infectat cele două comitate vecine cu primul, în al treilea an a infectat cele trei comitate vecine cu anterioarele două şi aşa mai departe, infecţia oprindu-se când bacteria a ajuns la marginea de sus sau la marginea din dreapta a hărţii.



Cerința

Scrieți un program care să determine numărul de comitate rămase neinfectate după oprirea expansiunii bacteriei.

Date de intrare

Se citesc de la tastatura n și m, dimensiunile hărții.

Date de ieșire

Se va afișa numărul de comitate rămase neinfectate după oprirea expansiunii bacteriei.

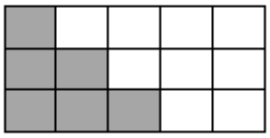
Restricții și precizări

- $1 \leq n \leq 1.000.000.000$, $1 \leq m \leq 1.000.000.000$, numere naturale

Exemplu

Intrare

3 5



Ieșire

9

Explicație

Harta Ținutului de Nord cuprinde 3 rânduri a câte 5 comitate fiecare, având în total 15 comitate. Expansiunea bacteriei s-a oprit după 3 ani, deoarece a atins marginea de sus; au rămas 9 comitate neinfectate.

Rezolvare

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

int main()
{
        long long n, m, minim, sol;
        cin >> n >> m;
        minim = n;
        if(m < minim)
        minim = m; //variabila minim va retine numărul de ani după care se opreşte expansiunea
sol = minim * (minim + 1) / 2; //sol retine numărul de comitate infectate după oprirea expansiunii
        cout << n * m - sol;

return 0;
}
```

4. Fișiere

1. Deschiderea fișierelor

Pentru a putea opera intr-un fișier, acesta trebuie deschis. Deschiderea unui fișier se face astfel:

- **Pentru citire** - valorile stocate in fișier vor fi citite in vederea prelucrării:
  ifstream <alias_fisier>("<nume_fisier>");

unde:

- alias_fisier - reprezintă numele cu care este recunoscut fișierul in cadrul programului
- nume_fisier - reprezintă numele fizic al fișierului pe disc
- ifstream - reprezintă fluxul de intrare

Exemple:

```
ifstream f("test.txt");
fstream f("test.txt", ios::in);
```

- **Pentru scriere** - valorile rezultate in urma prelucrării vor fi scrise in fișier:
  ofstream <alias_fisier>("<nume_fisier>");

unde:

- alias_fisier - reprezintă numele cu care este recunoscut fișierul in cadrul programului
- nume_fisier - reprezintă numele fizic al fișierului pe disc
- ofstream - reprezintă fluxul de ieșire

Exemple:

```
ofstream f("testare.txt");
fstream f("test.txt",ios::out);
```

2. Închiderea fișierelor

După terminarea operațiilor de intrare/ieșire cu fișierele utilizare, acestea trebuiesc închise, înainte de ieșirea din program. Acest lucru se realizează astfel:

<alias_fisier>.close();

Exemplu:

```
f.close();
g.close();
fin.close();
fout.close();
```

3. Citirea datelor din fișiere se realizează astfel: **<alias_fisier> >> <nume_var>;**

Exemplu:

```
f>>a;
fin>>a;
```

4. Scrierea datelor in fișiere se realizează astfel: **<alias_fisier> << <nume_var>;**

Exemplu:

```
f<<a;
fout<<a;
```

Exersaţi:

Rezolvaţi următoarele probleme de pe www.pbinfo.ro:
Catalin si elfii magigi (#1934), Parcare (#1470).

Problema Rapunzel de pe Pbinfo

Rapunzel s-a născut într-un ţinut îndepărtat într-un regat necunoscut. Mama vitregă a închis-o pe Rapunzel într-un turn foarte înalt ce avea N metri. Aici Rapunzel urma să-şi petreacă toată copilăria.

Pentru a-i trece timpul mai uşor, Rapunzel cânta din zori şi până în seară, cântecul ei auzindu-se în tot ţinutul. Atras de vocea de privighetoare a fetei, Flynn Rider şi-a propus să se cațere pe pereţii exterior ai turnului şi în fiecare zi să procedeze astfel: de la răsăritul până la asfinţitul soarelui să parcurgă M1 metri iar apoi, de la asfinţit până la miezul nopţii încă M2 metri.

Scrieţi un program care determină după câte zile ajunge Flynn Rider la Rapunzel.

Date de intrare

De pe prima linie a fişierului rapunzel.in se citesc trei numere naturale N, M1 şi M2, în această ordine, despărţite prin câte un spaţiu, având semnificaţia din enunţ.

Date de ieşire

În fişierul rapunzel.out se va afişa, pe prima linie, un număr natural ce reprezintă numărul de zile necesar lui Flynn Rider pentru a ajunge la Rapunzel.

Restricţii şi precizări

- 1 ≤ n ≤ 5 000 000 000
- 1 ≤ M1, M2 ≤ 2 500

Exemplul 1:

rapunzel.in	rapunzel.out
7 3 4	1

Explicaţie: Turnul are înălţimea de 7 metri. Flynn urcă până la asfinţit 3 metri iar apoi încă 4 metri. După o zi el va ajunge la Rapunzel.

Exemplul 2:

rapunzel.in	10 1 3
rapunzel.out	3

Explicaţie: Turnul are înălţimea de 10 metri. Flynn urcă până la asfinţit 1 metru iar până la miezul nopţii încă 4 metri. După 3 zile el va ajunge la Rapunzel.

```
        #include <fstream>
        using namespace std;
         ifstream f("rapunzel.in");
        ofstream g("rapunzel.out");
int main()
{
        long long N,M1,M2,t;
        f>>N>>M1>>M2;
        t=M1+M2;
        if (t>N)
        g<<1;
  else if ( N%t == 0)
         g<<N/t;
    else
         g<<N/t+1;

  return 0;
}
```

Problema Pinochio (clasa a 5-a) - Campion

În fiecare zi lucrătoare din săptămână, Pinochio spune câte o minciună datorită căreia nasul acestuia creşte cu câte p centimetri pe zi. Sâmbăta şi duminica, când vine bunicul Gepeto acasă, pentru a nu-l supăra prea tare, Pinochio reuşeşte să nu spună nici o minciună, ba chiar uitându-se în oglindă observă că în fiecare din aceste zile lungimea nasului său scade cu câte 1 centimetru pe zi. Când începe o nouă săptămână, rămânând singur acasă Pinochio continuă şirul minciunilor.

Cerinţă

Care este dimensiunea nasului lui Pinochio după k zile ştiind că iniţial nasul său măsura n centimetri.

Date de intrare

Din fişierul de intrare pinochio.in se citesc valorile n p k, care se găsesc pe prima linie a fişierului şi sunt separate prin câte un spaţiu.

Date de ieşire

În fişierul de ieşire pinochio.out se va afişa pe prima linie un singur număr natural, numărul de centimetri cerut de problemă.

Restricţii: 1 ≤ n ≤ 1000; 1 ≤ k ≤ 256; 1 ≤ p ≤ 100

Exemplu

pinochio.in	pinochio.out	Explicaţii
2 1 8	6	Zilele încep cu luni. Pentru exemplul dat zilele sunt luni, marţi, miercuri, joi, vineri, sâmbătă, duminică, luni.

Rezolvare: vom împărţi cele k zile în săptămâni întregi plus câteva zile rămase. Astfel vom avea:

- k / 7 săptămâni
- k % 7 zile rămase în ultima săptămână

Într-o săptămână lui Pinochio îi creşte nasul cu 5 * p - 2 centimetri. În ultima săptămână, cea incompletă, avem două cazuri:

- dacă zilele rămase, k % 7, sunt între 0 şi 5 numărul de centimetri cu care creşte nasul lui Pinochio în ultima săptămână este (k % 7) * p
- dacă zilele rămase sunt în număr de 6 avem şi o zi de final de săptămână, drept care numărul de centimetri cu care creşte nasul lui Pinochio în ultima săptămână este 5 * p - 1

Remarcaţi că nu este posibil ca numărul de zile rămase să fie 7, deoarece săptămâna ar fi plină. Algoritmul de mai jos implementează această soluţie:

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

ifstream fin("pinochio.in");
ofstream fout("pinochio.out");

int main() {
 int n, p, k, cm;
fin>>n>>p>>k ;
cm = n + (5 * p - 2) * (k / 7);
  k = k % 7;
  if ( k == 6 )
        cm = cm + 5 * p - 1;
  else
        cm = cm + k * p;
 fout<<cm;
                fout.close();
  return 0;
}
```

5. Structuri repetitive

Structurile repetitive execută o instrucţiune de un anumit număr de ori, sau cât timp o condiţie este adevărată. Se mai numesc şi bucle sau cicluri.

Structurile repetitive pot fi:

- cu număr cunoscut de paşi (iteraţii) – se cunoaşte de la început de câte ori se va execută instrucţiunea
- cu număr necunoscut de paşi (iteraţii). Instrucţiunea se execută cât timp o condiţie este adevărată. La fiecare pas se va evalua condiţia, iar dacă aceasta este adevărată se va executa instrucţiunea.

Structurile repetitive cu număr necunoscut de paşi pot fi:

- cu test iniţial: mai întâi se evaluează condiţia; dacă este adevărată se execută instrucţiunea şi procesul se reia.
- cu test final: mai întâi se execută instrucţiunea, apoi se evaluează condiţia; Dacă este adevărată, procesul se reia.

Instrucţiunea care se execută în mod repetat poartă numele de corp al structurii repetitive, corp al ciclului, corp al buclei şi de foarte multe ori este o instrucţiune compusă.

5.1 Instrucţiunea while

Instrucţiunea while este o structură repetitivă cu număr necunoscut de paşi şi test iniţial.

Sintaxa:

```
while (expresie) {
   Instrucţiuni;
}
```

Mod de execuţie:

1.Se evaluează expresie
2.Dacă expresie este adevărată ◦Se execută Instrucţiuni; ◦Se reia pasul 1.
3.Dacă Expresie este nulă, se trece la instrucţiunea de după while.
Observaţii

- Instrucţiuni; se execută cât timp expresie este nenulă – condiţie adevărată.
- Dacă expresie este de început vidă, Instrucţiuni; nu se execută deloc.
- Instrucţiuni; poate fi una singură sau instrucţiunea compusă.
- Este necesar ca cel puţin o variabilă care apare în expresie să-şi modifice valoarea în Instrucţiuni; Altfel se obţine o buclă infinită.

5.2 Algoritmi elementari

Prelucrarea cifrelor unui număr natural

Exerciţiul 1
Calculul şi afişarea sumei cifrelor unui număr natural n.

```
#include <iostream>
using namespace std;
int main() {
int n, s; // declararea variabilelor
cin>>n; // citirea numărului n
s=0; //iniţializarea sumei cu 0
while(n>0)
{ s=s+n%10; //adăugarea ultimei cifre la sumă
 n=n/10; // tăierea ultimei cifre a variabilei n
}
cout<<s; // afisarea sumei
return 0;
}
```

Exerciţiul 2
Aflarea primei cifre a numărului natural n.

```
#include <iostream>
using namespace std;
int main() {
int n; // declararea variabilei
cin>>n; // citirea variabilei n
while(n > 9)// cât timp n>9
{
n = n / 10;// tăierea ultimei cifre până când n ajunge de o cifră
}
cout<<n; //afisarea lui n (primei cifre a lui n)
return 0;
}
```

Exerciţiul 3
Determinarea inversului (oglinditului) lui n de ex: n=1234, inv=4321

```
#include <iostream>
using namespace std;
int main() {
int n, inv; //declararea variabilelor
cin>>n; // citirea lui n
inv = 0; // iniţializarea inversului cu 0 – deoarece este o sumă
while(n > 0) până când n ajunge < sau = cu 0
{
inv = inv * 10 + n%10; // se construieşte inversul pornind de la ultima cifră
n = n / 10; // golirea numarului n de cifre
}
cout<<inv;
return 0;
}
```

Exerciţiul 4. Numere de tip palindrom

Cum aflăm dacă un număr este de tip palindrom (simetric)? Exemplu: 22, 32123, 454, etc. Algoritmul clasic răstoarnă numărul şi testează dacă numărul este egal cu răsturnatul său:

```
#include <iostream>
using namespace std;
int main() {
int  p, c, r; // declararea variabilelor
 cin>>p; // citirea numărului p – presupus a fi de tip palindrom
  c = p; // salvarea numărului p într-o altă variabilă c (copie)
  r = 0; // initializarea lui r cu 0
  while ( p > 0 ) {
    r = r * 10 + p % 10; // se construieşte răsturnatul
    p = p / 10;// se goleşte numărul de cifre
  }
  if ( c == r ) // dacă răsturnatul este egal cu copia
    cout<<"DA" ;
  else
       cout<<"NU" ;
  return 0; }
```

Acest algoritm poate depăşi valoarea maximă reprezentabilă pe tipul long long (pentru numere mari).

O soluţie ar fi să răsturnăm numărul numai până la jumătate:

```
#include <iostream>
using namespace std;
int main() {
 long long p, r;
 cin>>p;    r = 0;
 while ( p > r ) {
   r = r * 10 + p % 10;
   p = p / 10;
 }// când numărul are un număr par de cifre testăm dacă p == r
 // când numărul are un număr impar de cifre testăm dacă p == r / 10
 if ( p == r || p == (r / 10) )
cout<<"DA" ;
 else
      cout<<"NU" ;
 return 0; }
```

Exerciţiul 5. Se dă un număr natural. Să se modifice acest număr, mărind cu o unitate fiecare cifră pară. Dacă numărul dat este 275 rezultatul va fi 375.

Rezolvare: Vom determina cifrele numărului dat şi vom construi rezultatul, inserând cifrele la început. Cifrele pare se inserează ca atare, cifrele impare se inserează micşorate.

- Fie n numărul dat şi r rezultatul. Vom utiliza o variabilă suplimentară, p, pentru a calcula puterile lui 10.
- Iniţial r = 0, p = 1
- Vom determina prin trunchieri succesive cifrele lui n în variabila uc, uc = n % 10.
    - Dacă uc este par, r = r+ p * uc, apoi p = p * 10.
    - Dacă uc este impar, r = r + p * (uc - 1), apoi p = p * 10.

```
#include <iostream>
using namespace std;
int main()
{
int n , r, p, uc;
cin >> n; r=0; p=1;
while(n>0)
{
        uc = n % 10;
        if(uc % 2 == 0)
                r= r+p * uc;
        else
                r= r+p * (uc - 1);
        p=p *10;
        n =/10;
}
cout << r;
return 0;}
```

Exerciţiul 6.

Eliminarea din număr a cifrelor cu o anumita proprietate. Acest algoritm construieşte noul număr începând cu cifra cea mai reprezentativă (cu prima cifră) şi utilizează o variabilă p care construieşte zecile, sutele, miile etc.

```
int n, uc, nou, p; cin>>n;
nou = 0;
p = 1; // orice produs se inițializează cu 1
while(n > 0)
{       uc = n % 10;// ultima cifră
        if (uc NU are proprietatea de șters) //uc trebuie păstrata
{
        nou = nou + uc * p;
        p = p * 10;
}
        n = n / 10;
}  cout<<nou;
```

Exerciţiul 7.

Descompunerea unui număr în cifre începând cu cifra cea mai semnificativă:

```
#include <iostream>
using namespace std;
int main()
{       int n, pc, p;
        p = 1;
while (p * 10 <= n)
{       p = p * 10;       }
while(n > 0)
{       pc = n / p;
        n = n % p;
        p = p /10;
cout<<pc<<"  ";}
return 0;
}
```

Exerciţiul 8.

Aflarea cifrei maxime dintr-un număr n:

```
#include <iostream>
using namespace std;
int main()
{
int n, cmax=0, u;//cifra maxima se iniţializează cu 0 sau cu -1
cin>>n;
while (n>0)
{       u=n%10; //ultima cifra
        if (u>cmax)
        {cmax=u;}
        n = n /10;// tăierea ultimei cifre până când n ajunge 0
}
cout<<cmax; // afişarea cifrei maxime
return 0;}
```

Exerciţiul 9.

Aflarea cifrei minime dintr-un număr n:

```
#include <iostream>
using namespace std;
int main()
{
int n, cmin=0, u;//cifra minima se iniţializează cu 10 sau cu 9
cin>>n;
while (n>0)
{       u=n%10; // ultima cifra
        if (u< cmin)
        { cmin =u;}
        n = n /10; // tăierea ultimei cifre până când n ajunge 0
}
cout<< cmin; // afişarea cifrei minime

return 0;}
```

Exerciţiul 10.

Se citeşte un număr natural n. Să se spună câte cifre pare are el. Exemple: dacă n=5428 el are 3 cifre pare (4, 2 şi 8), dacă n=49285640 el are 6 cifre pare (4, 2, 8, 6, 4, 0).

```
#include <iostream>
using namespace std;
int main() {
 int n, nrp;
 cin>>n;
 nrp = 0;
 while ( n > 0 ) {
   c = n % 10;
if ( c % 2 == 0 )
    { nrp = nrp + 1; }
   n = n / 10;   }
 cout<<"Nr. cifre pare: "<<nrp ;
 return 0; }
```

Răspuns 2: Putem simplifica această soluție:

1. n % 10 % 2 este totuna cu n % 2. Deci testul c % 2 == 0 poate fi înlocuit cu n % 2 == 0, scutind un calcul și o variabilă. Pentru aceasta trebuie să facem testul înainte de a-l împărți pe n la 10.

2. Ce valori poate lua restul împărțirii la 2? Zero sau unu. Când adunăm 1 la nrp? Atunci când restul este 0. Atunci când restul este unu nu adunăm nimic la nrp, ceea ce este totuna cu a aduna zero. Cu alte cuvinte putem întotdeauna să adunăm negarea restului. Cum calculăm acest număr? El este 1 - rest.

Iată o soluție bazată pe aceste două simplificări:

```
#include <iostream>
using namespace std;
int main() {
 int n, nrp;
 cin>>n;
 nrp = 0;
 while ( n > 0 ) {
   nrp = nrp + 1 - n % 2;
  n = n / 10;
 }
 cout<<"Nr. cifre pare:  "<<nrp ;
 return 0; }
```

Exercițiul 10.

Se citește un număr n. Să se afișeze cel mai mare divizor propriu al lui n (strict mai mic decât n). Exemplu: dacă n=24 cel mai mare divizor propriu este 12. Dacă n=7 cel mai mare divizor propriu este 1. Dacă n=125 cel mai mare divizor propriu este 25. Dacă n=175 cel mai mare divizor propriu este 35.

```
#include <iostream>
using namespace std;
int main() {
int n, d;
 cin>>n ;
 d = n - 1;
 while ( n % d > 0) 1 // cât timp nu mai dăm de niciun divizor
 {  d = d - 1;} // decrementez cel mai mare divizor
 cout<< "Cel mai mare divizor propriu: "<<d ;
 return 0; }
```

Exercițiul 11 - divizorii unui număr

Se citește un număr n, să se afișeze toți divizorii lui n. Spunem că d este divizor al lui n dacă n se împarte la d. Pentru a rezolva exercițiul vom varia un contor d de la 1 la n. El reprezintă potențialii divizori ai lui n. Pentru fiecare valoare a lui d, vom testa dacă n se împarte la d. Dacă da, vom afișa acel divizor.

```
#include <iostream>
using namespace std;
int main() {
int n, d;
 cin>>n ;
cout<<"Divizorii lui "<<n<<" sunt: ";
 d = 1;
 while ( d <= n/2 ) 2{
```

[1] Condiția de oprire este n%d<=0 (când am dat de un divizor)

```
  if ( n % d == 0 ) //dacă am găsit un divizor
    cout<<d<<" "; // îl afişăm
   d = d + 1;
 }
cout<<n;//la sfârşit îl afişăm şi pe n
 return 0; }
```

Exerciţiul 12 – număr prim

Se citeşte un număr n. Să se spună dacă n este prim. Un număr este prim dacă nu se împarte decât la 1 şi la el însuşi. Vom proceda similar cu afişarea divizorilor: vom căuta primul divizor al lui n, începând cu 2. Dacă găsim un divizor, numărul nu este prim. Dacă, în schimb, primul divizor găsit este chiar n, numărul este prim.

Putem face mai puţine iteraţii dacă observăm că dacă un număr nu are nici un divizor strict mai mare ca 1 dar mai mic sau egal cu radical din n atunci el nu va avea nici alţi divizori mai mari decât radical din n dar mai mici decât n. Putem, deci, înlocui condiţia d <= n/2 cu d * d <= n. În acest caz va trebui să modificăm condiţia finală de test de primalitate în d * d > n.

```
 #include <iostream>
using namespace std;
int main() {
int n, d;
 cin>>n ;
 d = 2;
 while ( d * d <= n  && 3 n % d > 0 ) /* condiţia de oprire este d * d > n or n % d == 0 adică ori nu am găsit {
d = d + 1;}                                  niciun divizor, ori am găsit primul divizor */
 if ( d * d > n )
   cout<<"d este prim ";
 else
   cout<<" d nu este prim ";
 return 0; }
```

Exerciţiul 13 – numere prime până la n

Se citeşte un număr n. Să se afişeze toate numerele prime mai mici sau egale cu n.

Vom folosi exerciţiul anterior. Vom varia un contor p de la 2 până la n. Aceste numere pot sau nu să fie numere prime, drept pentru care vom testa fiecare din ele dacă este prim.

```
 #include <iostream>
using namespace std;
int main() {
int n, p, d;
 cin>>n ;
 p = 2;
while (p<=n){
d=2;
 while ( d * d <= p  &&  p % d > 0 ) { d = d + 1;}
 if ( d * d > n )
   cout<<p<<" ";
p=p+1; }
return 0; }
```

Exerciţiul 14

---

[2] Între jumătatea numărului şi n, nu se mai află divizori

[3] Operatorul logic şi (**and)**

Se citesc două numere, n şi k. Să se afişeze toate numerele mai mici sau egale cu n care se divid cu k. Exemple: pentru n = 13 k = 3 se va afişa 3 6 9 12; pentru n = 30 k = 5 se va afişa 5 10 15 20 25 30.
Răspuns: am putea să variem un contor d, de la 1 la n şi, pentru fiecare valoare a lui d să testăm dacă se împarte la k. Dar există o rezolvare mai eficientă: ştim că numerele divizibile cu k sunt k, 2k, 3k, etc. Drept care putem să generăm direct aceste numere, pornind de la k şi adunând k la fiecare iteraţie.

```
#include <iostream>
using namespace std;
int main() {
        int n, d, k;
        cin>>n>>k ;
         d = k;
         while ( d <= n ) {
                 cout<<d<<" ";
                d = d + k;
 }
 return 0; }
```

Descompunerea în factori primi

Exerciţiul 15

Se citeşte un număr n. Să se descompună în factori primi. Exemple: dacă n = 45 vom afişa 45 = * 3^2 * 5^1; dacă n = 1008 vom afişa 1008 = * 2^4 * 3^2 * 7^1. Observaţie: nu este nevoie să testăm dacă un divizor este prim, putem testa toţi divizorii.

```
#include <iostream>
using namespace std;
int main() {
        int n, p, d;
        cin>>n; cout<< "n= ";
        d = 2;
         while ( n > 1 ) {
         p = 0;
        while ( n % d == 0 ) { dacă am găsit un divizor ...
                 p = p + 1;    // puterea unui divizor prim d – de cate ori se cuprinde d în n
                 n = n / d;   // ... împărţim numărul la d cât timp se mai poate
        }
                if ( p > 0 ) // dacă puterea este >0
                cout<<" *"<<d<<" ^"<<p ;// afişăm
                d = d + 1;
        }
 return 0;
}
```

Algoritmul lui Euclid (cel mai mare divizor comun a doua numere)

Exerciţiul 16

Cel mai mare divizor comun al două numere naturale n şi m poate fi determinat folosind descompunerea în factori primi a celor două numere. Această metodă este mai dificil de implementat. Există o metodă mai simplu de, numită **algoritmul lui Euclid**.

Algoritmul lui Euclid cu împărţiri se bazează pe ideea că cel mai mare divizor a două numere a, b divide şi restul împărţirii acestora, r, conform teoremei împărţirii cu rest. Algoritmul este:

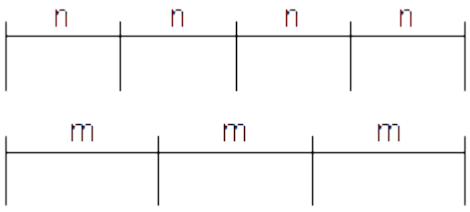
- Cât timp b != 0:
  - Determinăm restul împărţirii lui a la b.
  - În continuare a devine b, iar b devine restul calculat.
- Valoarea actuală a lui a este cel mai mare divizor comun a valorilor iniţiale.

```
#include <iostream>
using namespace std;
int main() {
        int a, b, r, p;
        cin>>a>>b;
        p=a*b;
         while ( b > 0 ) {
                  r = a % b;
                a = b;
                b = r;
         }
 cout<<a<<" ";        //cmmdc
 cout<<p/a;           //cmmmc
 return 0;
}
```

Observaţie: Pentru a determina cel mai mic multiplu comun a două numere naturale folosim următoarea teoremă: Produsul a două numere naturale este egal cu produsul dintre cel mai mare divizor comun al lor şi cel mai mic multiplu comun al lor: n * m = cmmdc * cmmmc

Exemplu

Doi prieteni, un iepure şi o broscuţă joacă un joc: pornesc de la o linie de start şi încep să sară. Broasca sare n centimetri, iar iepurele m centimetri. Cine este mai în spate vine la rând să sară. Jocul se termină atunci când iepurele şi broasca sunt iarăşi la egalitate. Câţi centimetri au parcurs cei doi prieteni?



Săriturile broscuţei şi iepurelui

Răspuns: după cum se vede şi din figură, broscuţa şi iepurele vor sări o lungime egală cu CMMMC(m, n).

Exerciţiul 17 – număr cu cifre distincte

Să se spună dacă un număr are toate cifrele distincte. Exemplu: 545453967 nu are cifre distincte, cifrele 5 şi 4 apar de două ori în număr; Numărul 5438 are cifre distincte, nici o cifră a sa nu se repetă.

```
#include <iostream>
using namespace std;
int main() {
        int n, cn, c;
        cin>>n;
         c n= 0;
        while ( n > 9 && cn == 0 ) {
                c = n % 10;
                cn = n / 10;
        while ( cn > 0 &&  cn % 10 != c )
                 cn = cn/ 10;
                n = n / 10;
                }
        if (cn==0)
                    cout<<"Cifre distincte";
        else
                     cout<<"Cifre nedistincte";
 return 0;
}
```

5.3 Instrucţiunea for

Atunci când anumite operaţii trebuie repetate de un număr de ori cunoscut (de obicei de un număr mare de ori, care nu permite scrierea repetată a operaţiilor în algoritm), se utilizează structura repetitivă pentru.

for (contor = expresie_iniţiala; contor<= expresie_finala; contor=contor+pas)
{ instrucţiuni; }

Execuţia instrucţiunii **for** are etapele următoare:

1. Se iniţializează contorul (variabila care numără paşii)
2. Se verifică daca valoarea contorului este mai mica sau egala cu valoarea finala
   - Dacă DA, se execută instrucţiuni, apoi contorul creste cu valoarea pasului şi revenim la pasul 2
   - Dacă NU este îndeplinită, se opreşte execuţia instrucţiunii for

Observaţii: Dacă sunt două sau mai multe instrucţiuni în **for**, se vor grupa între acolade {...}.
Instrucţiunea **for** este echivalentă cu instrucţiunea **while**. Sintaxa descrisă mai sus este echivalentă cu:

```
        contor = expresie_iniţiala;
        while (contor<= expresie_finala)
        {         Instructiuni;
                  contor=contor+pas;
}
```

Exemple:

Următorul program citeşte valoarea variabilei n şi calculează suma primelor n numere naturale. Rulaţi-l analizând rezultatul pentru diverse valori ale lui n, inclusiv 0.

```
#include <iostream>
using namespace std;
int main ()
{
    int n,i,s=0;
    cin >> n;
for (i = 1; i <= n ; i ++)// i=i+1
    {   s=s+i;  }
cout << s;
return 0;
}
```

```
#include <iostream>
using namespace std;
int main ()
{
   int n,i,s=0;
   cin >> n;
   i=1;
while (i <= n )
        {  s=s+i;
           i=i+1;    }
cout << s;
return 0;
}
```

Instrucţiunea break

Sintaxa:

```
break;
```

Mod de execuţie

Efectul instrucţiunii break când apare într-o instrucţiune repetitivă este întreruperea execuţiei acesteia şi trecerea la instrucţiunea care urmează celei repetitive.

Exemplu:

```
    #include <iostream>
    using namespace std;
    int main ()
{
    int n, i;
    cin >> n;
    int S = 0;
for( i = 1; i <= n ; i ++)
{
    S = S+i;
    if(i == 5) break;
}
cout << S << endl;
return 0; }
```

- Dacă valoarea lui n este cel mult 5, se va afişa suma numerelor de la 1 la n.
- Dacă n >= 5 se va afişa întotdeauna 15, deoarece execuţia lui for se întrerupe, datorită lui break, când i este 5.

Instrucţiunea continue

Sintaxa:

```
continue;
```

Mod de execuţie

Efectul instrucţiunii continue este ignorarea instrucţiunilor care îi urmează în corpul ciclului şi revenirea la evaluarea expresiei, în cazul lui while, respectiv la evaluarea expresiei_de_continuare, în cazul lui for.

Exemplu:

```
#include <iostream>
using namespace std;
        int main ()
{       int n, i;
        cin >> n;
        int S = 0;
        for(i = 1; i <= n ; i ++)
 {
        if(i % 2 == 0)
        continue;
        S = S+i;
 }
 cout << S;
 return 0;
}
```

Divizibilitate

Mihai este un mare amator de jocuri pe calculator. Deoarece nu s-a putut abţine si s-a jucat in timpul orei de informatica, doamna profesoară i-a dat un şir de **n** numere naturale. Ca să nu îi pună notă mică, el trebuie să numere câte elemente de pe poziţii pare din şir sunt divizibile cu numărul **k**. Ajutaţi-l pe Mihai!

Date de intrare : numerele naturale **n** si **k** iar apoi **n** numere naturale.

Date de ieşire

Afişaţi numărul de elemente de pe poziţii impare care sunt divizibile cu numărul **k.**

Restricţii şi precizări

- 1 ≤ n ≤ 1000
- 2 ≤ k ≤ 1 000 000
- 1 ≤ fiecare element din sir ≤ 1 000 000 000

Exemplu		**Explicaţii**
6 3 5 9 7 6 12 14	2	Numerele de pe poziţii pare sunt: **9 6 14** Dintre acestea doar 9 şi 6 sunt divizibile cu 3.

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

int main() {
        int i, n, k,  a, r;
        r=0;
        cin >> n >> k;
        for( i = 1; i <= n; i++) {
                cin >> a;
                if (i % 2!=0)  r = r +  (a % k == 0);
        }
        cout << r;       }
```

Cadouri - Centrul de pregătire "Hai La Olimpiadă!"

Moş Crăciun, harnic şi răbdător cu toţi copiii, se gândeşte să facă inventarul cadourilor care trebuiesc transmise elevilor de la CEX. Primeşte lista elevilor şi codifică cadourile cu numere **de exact trei cifre**, astfel:

1. fetele vor primi cadouri a căror primă cifra este 2 iar cadourile băieţilor vor începe cu cifra 1,
2. valoarea fiecărui cadou, este dată de ultima cifră a codificării cadoului,
3. elevii care sunt de la acelaşi liceu au cifra zecilor aceiaşi.

Deoarece elevii sunt de la mai multe scoli, fiecare şcoala are un cod unic, format dintr-o singură cifră!
Vă rugăm să îl ajutaţi pe Moş Crăciun să afle:

1. ce valoare au cadouri pregătite de Moş, pentru fetiţe.
2. câţi elevi de la *SCOALA TÂNĂRA SPERANŢĂ* , şcoală codificată cu **k**, sunt în grupa CEX.

Date de intrare

Fişierul de intrare **cadouri.in** conţine pe prima linie un număr natural **c (**1 sau 2) reprezentând numărul cerinţei ce trebuie rezolvată.
A doua linie conţine două numere:
n - un număr natural reprezentând numărul de elevi înscrişi la CEX şi
k - un număr natural reprezentând codul şcolii TÂNĂRA SPERANŢĂ.
Pe a treia linie se află **n** numere naturale **a1, a2, ..., an**, separate prin câte un spaţiu, cu semnificaţia din enunţ.

Date de ieşire

Fişierul de ieşire **cadouri.out** va conţine pe prima linie un singur număr reprezentând răspunsul la cerinţa citită din fişierul de intrare.
Pentru **c**=1 se va rezolva doar cerinţa 1, pentru c=2 se va rezolva doar cerinţa 2.

Restricţii şi precizări

- $1 \le n \le 1000000$
- $0 \le k \le 9$
- La punctul 1) se acordă 60 puncte iar la punctul 2 se acordă 40 puncte

cadouri.in	cadouri.out	Explicaţii
1 5 4 142 *225* 141 *267* *246*	18	c=1 n=5 deci avem 5 elevi înscrişi la CEX k=4, codul scolii este 4 In grupă sunt 3 fetiţe şi 2 băieţi cadourile fetiţelor au valorile 5, 7 şi 6 iar suma este 5+7+6=18
2 5 4 1**4**2 *225* 1**4**1 *267* *2**4**6*	3	c=2 n=5 deci avem 5 elevi înscrişi la CEX k=4, codul scolii este 4 3 elevi sunt de la şcoala codificată c, 4; elevii: 1, 3 şi 5
1 6 9 128 213 192 172 295 140	8	c=1 n=6 k=9 4 fetiţe ce au cadourile în valoare de 3 + 5 = 8
2 6 9 128 213 192 172 295 140	2	c=2 n=6 k=9 2 elevi învaţă la şcoala codificată cu 9, elevii 3 şi 5

```
#include <fstream>
using namespace std;
ifstream fin("cadouri.in");
ofstream fout("cadouri.out");
int n, k, c, i, a, v, s;
int main()
{
  fin>>c;
  fin>>n>>k;
  if(c==1)
  {
     s=0;
    for(i=1; i<=n; i++)
    {
       fin>>a;
       if(a/100==2)
       {
          v=a%10;
          s=s+v;
       }
    }
    fout<<s;
  }
  if(c==2)
  {
    v=0;
    for(i=1; i<=n; i++)
    {
       fin>>a;
       if(a/10%10==k)
       {
         v=v+1;
       }
    }
    fout<<v;
  }
   return 0;
}
```

Problema rachete * - Campion

În tabăra „Space Camp” copiii învaţă să lanseze rachete. În funcţie de înălţimea la care se ridică aceste rachete, unele devin roşii, altele albastre şi altele galbene. Cele albastre primesc eticheta 1, cele roşii eticheta 2, iar cele galbene eticheta 3.

Cerinţă: Cunoscând n, numărul de rachete lansate şi valoarea de pe eticheta fiecărei rachete, să se afişeze etichetele rachetelor în ordine crescătoare.

Date de intrare: Fişierul rachete.in conţine pe prima linie un număr natural n, reprezentând numărul de rachete. Pe cea de a doua linie se află n numere naturale din mulţimea {1, 2, 3}, reprezentând etichetele celor n rachete.
Date de ieşire: Fişierul rachete.out va conţine o singură linie, pe care vor fi scrise în ordine crescătoare etichetele celor n rachete, separate prin câte un singur spaţiu.

Restricţii

- $0 < n \leq 60000$
- $1 \leq r_i \leq 3$, pentru $1 \leq i \leq n$

Exemple

rachete.in	rachete.out
10 1 2 1 3 2 2 2 1 2 3	1 1 1 2 2 2 2 2 3 3

```
#include<fstream>

using namespace std;

ifstream fin("rachete.in");
ofstream fout("rachete.out");

int main (){
        int n, c, nr, a=0, b=0, d=0;
        fin>>n;
        for(c=1;c<=n; c++){
         fin>>nr;
         if(nr==1){
        a++;
        }
        if(nr==2){
           b++;
        }
        if(nr==3){
          d++;
        }
}
        for(c=1; c<=a; c++)
         {fout <<1<<" ";}

        for(c=1; c<=b; c++)
        { fout <<2<<" ";}

        for(c=1;c<=d; c++)
        {fout <<3<<" ";}

fin.close();      fout.close();

return 0;

}
```

6. Secvențe sau șiruri de numere

Elementul maxim într-o secvență

Dată o secvență de n numere să se afișeze elementul cel mai mare (maxim) din secvență. Exemple:

maxim.in	maxim.out
7 5 21 0 47 1 6 7	47
10 12 12 10 12 83 68 83 22 11 12	83

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

ifstream fin("maxim.in");
ofstream fout("maxim.out");

int main()
{
        int n, i, a, maxx;
        fin>>n;
        fin>>a;
        maxx=a;
for ( i = 2; i < =n; i++ ) {
        fin>>a;
   if ( max < a )
     max = a;
  }
        fin.close( );
        fout<<max ;
        fout.close( );

return 0;
 }
```

Observații:

- Elementul maxim se inițializează cu primul element al șirului **nu** cu valoarea zero. Întotdeauna vom inițializa maximul cu un element al secvenței. De ce?
    - Este bine să fim ordonați și consecvenți: maxx este doar un candidat la maxim, dintre elementele secvenței, până la final, când devine chiar maximul.
    - În viitor putem avea și elemente negative.
    - Dacă avem o greșeală în program detecția ei și corectura sunt mai grele atunci când maxx poate lua valori în afara elementelor din secvență.
- Pentru a calcula elementul *minim* într-o secvență singurul lucru care se modifică în schema de mai sus este semnul comparației maxx < a, din mai mic < în mai mare >.

Problema Poziții (clasa a 5-a) - Varena

Se citește o secvență de n numere. Să se spună câte din numere sunt egale cu poziția lor în secvență. Primul număr este pe poziția 0, ultimul pe poziția n–1.

Date de intrare: Fișierul de intrare pozitii.in conține pe prima linie numărul de numere, n. Pe următoarea linie conține cele n numere separate cu spații.
Date de ieșire: În fișierul de ieșire pozitii.out veți scrie un singur număr și anume numărul de numere din secvență egale cu poziția lor în secvență.
Restricții

- $1 \le n \le 100\,000$
- $1 \le a_i \le 100\,000$, unde a_i este un număr din secvență

Exemple

pozitii.in	pozitii.out	Explicații
4 0 3 2 5	2	Cele două numere egale cu poziția lor sunt 0 și 2
10 7 1 2 3 2 5 9 7 3 9	6	Cele șase numere sunt 1 2 3 5 7 și 9

Răspuns: problema este una introductivă în secvențe, deci nu foarte grea. Va trebui să folosim un *contor* pentru a număra poziția fiecărui număr citit. În algoritmul de mai jos el se numește i. De asemenea vom menține un al doilea contor care numără câte numere sunt egale cu poziția lor. După citirea numărului curent din secvență îl vom compara cu contorul de poziție i și vom aduna 1 la poz dacă sunt egale. În final poz este chiar numărul pe care trebuie să îl afișăm.

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

ifstream fin("pozitii.in");
ofstream fout("pozitii.out");

int main()
{
        int n, i, a, poz;
        fin>>n;
        poz = 0;
        i = 0;
  while ( i < n ) {
   fin>>a ;
   if ( a == i )
     poz = poz + 1;
   i = i + 1;
  }
        fin.close( );
        fout<<poz;
  fout.close( );
  return 0;
}
```

Secvenţă în ordine crescătoare

Dată o secvenţă cu n numere să se spună dacă cele n numere din secvenţă sunt in ordine crescătoare (fiecare număr este mai mic sau egal cu cel de după el).

crescatoare.in	crescatoare.out
6 2 7 7 10 15 15	da
8 3 3 6 8 8 10 12 11	nu

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

        fin ( "crescatoare.in");
        fout ( "crescatoare.out");

 int main() {
   int n, i, a, b, cresc;
fin>>n;
fin>>a;
   cresc = 1;
   i = 1;
   while ( i < n && cresc == 1) {
    fin>>b;
    if ( b < a )
      cresc = 0;
    a = b;
    i++;
   }
   fin.close( );

   if ( cresc == 1 )
    fout<<"da\n" );
   else
    fout<<"nu\n" );
   f.close(  );
   return 0;
 }
```

Observaţii:

- Nu folosim instrucţiunea *for* deoarece nu se ştie de câte ori se va executa bucla while (nu avem un ciclu cu număr cunoscut de paşi).
- Trebuie să ţinem minte elementul anterior în secvenţă, pentru a-l putea compara cu elementul curent.

Şirul lui Fibonacci

Definiţie: şirul lui Fibonacci este secvenţa de numere 0, 1, 1, 2, 3, 5, 8, 13... Regula acestei secvenţe este că primele două numere sunt 0 şi 1, iar următoarele numere sunt suma celor două numere din-înaintea lor. **Exerciţiu**: dat n, să se calculeze al n - lea termen din şirul lui Fibonacci. Exemple:

fibonacci.in	fibonacci.out
1	0
2	1
5	3
8	13

```
        #include <fstream>
        using namespace std;
        ifstream fin ( "fibonacci.in");
        ofstream fout ( "fibonacci.out");
int main() {
        int n, i, a, b, f;
        fin>>n;
        fin.close(  );
        a = 0;
        if ( n == 1 )
        b = a;
 else {
         b = 1;
  for ( i = 2; i < n; i++ ) {
        f = a + b;
        a = b;
        b = f;
  }
 }
 fout<<b;
 fout.close( );
 return 0;
}
```

Observaţii:

- Secvenţa de numere nu este *citită* ci *generată*.
- Este necesar să memorăm *ultimele două numere* din secvenţă pentru a genera următorul număr.

Numere identice

Se dă o secvenţa de n numere. Să se spună care este numărul maxim de numere identice consecutive în secvenţă. Exemple:

identice.in	identice.out	Explicaţie
10 6 2 2 5 8 8 8 2 2 5	3	Numărul 8 apare de trei ori la rând. Nici un alt număr nu apare de mai multe ori la rând.
14 8 8 3 3 3 3 1 1 1 5 5 5 5 2	4	Numărul 3 apare de patru ori la rând. De asemenea şi numărul 5 apare de patru ori la rând. Nici un alt număr nu apare de mai multe ori la rând.

```
#include <fstream>
using namespace std;
ifstream fin ( "identice.in");
ofstream  fout ( "identice.out");
 int main() {
  int n, i, a, b, l, lmax;
    fin>>n>>a;
  lmax = 1;
  l = 1;
  for ( i = 1; i < n; i++ ) {
    fin>>b ;
    if ( b == a ) {
      l++;
      if ( l > lmax )
        lmax = l;
    } else {
      a = b;
      l = 1;
    }
  }
  fin.close(  );
  fout<<lmax;
  fout.close(  );
  return 0;
}
```

Observaţii:

- Şi aici ţinem minte elementul anterior în secvenţă, pentru a-l putea compara cu cel actual.
- O alternativă mai eficientă ar fi să comparăm l > lmax numai atunci când b ≠ a. Se poate şi aşa, dar trebuie să avem grijă în cazul în care cea mai lungă secvenţă este chiar la final. Pentru acest caz particular va trebui ca imediat după bucla WHILE-DO să mai testăm o dată dacă l > lmax.

Numărare de cuvinte

Considerăm o secvenţă de numere. Să considerăm zerourile ca spaţii, iar numerele diferite de zero ca litere. Dorim să numărăm câte cuvinte avem în secvenţă. Exemple:

cuvinte.in	cuvinte.out	Explicaţie
10 3 5 0 0 2 9 7 0 1 3	3	Sunt trei cuvinte (subsecvente de numere diferite de zero), 3 5, 2 9 7 şi 1 3.
10 0 0 0 2 6 1 0 0 1 0	2	Sunt două cuvinte (subsecvente de numere diferite de zero), 2 6 1 şi 1.

O variantă de rezolvare ar fi să ne uităm la două numere consecutive în secvenţă şi să vedem dacă începe un cuvânt (sau dacă se termină). O a doua variantă este să ţinem o variabila incuv care să ne spună dacă ne aflăm în interiorul unui cuvânt sau nu. Vom prefera această variantă deoarece este mai simplă şi se poate generaliza pentru situaţii mai complicate.

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

        ifstream fin ( "cuvinte.in" );
        ofstream fout ( "cuvinte.out");

int main() {
         int n, i, a, incuv, nrcuv;
        fin>>n;
        nrcuv = 0;
        incuv = 0;
 for ( i = 0; i < n; i++ ) {
         fin>>a;
        if ( a == 0 )
         incuv = 0;
   else
         if ( incuv == 0 ) {
        nrcuv++;
        incuv = 1;
    }
 }
 fin.close( );
 fout<<nrcuv ;
 fout.close();

 return 0;
}
```

7. Lucrul cu caractere

În orice sistem de calcul, datele – de orice tip – se memorează sub formă de numere. Mai mult, acestea se reprezintă în baza 2. În consecinţă, pentru a memora în calculator **caractere** este necesară utilizarea unei reprezentări a caracterelor prin numere. O astfel de reprezentare este **Codul ASCII**.

Prin codul ASCII, fiecărui caractere reprezentat în acest cod i se asociază un număr. Aceste numere (numite chiar coduri ASCII) se găsesc în intervalul 0 .. 127. Caracterele ASCII se împart în două categorii:

- caractere imprimabile – cele cu codurile ASCII în intervalul 32 126, inclusiv capetele: aici se regăsesc toate caracterele care au o reprezentare grafică bine determinată:
  - literele mari: A ... Z,
  - literele mici: a ... z,
  - cifrele 0 .. 9,
  - semnele de punctuaţie .,:;!?'"
  - caractere ce reprezintă operaţii aritmetice sau de alt tip: + - / * <> = (){}[]
  - alte caractere: ~`@#$%^&_\|
  - **caracterul spaţiu**
- caracterele neimprimabile, sau de control – cu codurile 0 .. 31 şi 127. Ele erau folosit mai demult pentru a controla transmisiunea datelor. Dintre aceste caractere amintim două, de o importanţă mai mare în limbajele de programare studiate:
  - caracterul cu codul 0, numit şi **caracter nul**, notat în C++ cu '\0' – reprezintă finalul unui şir de caractere în memorie
  - caracterul cu codul 10, numit **Line Feed**, notat în C++ cu '\n' – produce trecerea la rând nou atunci când este afişat pe ecran sau într-un fişier.

Observaţii utile

- literele mari şi literele mici sunt diferite – au coduri ASCII diferite
- codurile ASCII ale literelor mari (sau mici) sunt în ordine: 'A' are codul 65, 'B' are codul 66, .. , 'Z' are codul 90. Două caractere consecutive în alfabet au coduri ASCII consecutive! De asemenea, litera 'a' are codul 97, etc.
- codurile ASCII ale literelor mici sunt mai mari decât codurile ASCII ale literelor mari ('a' > 'Z') şi diferenţă între codurile ASCII a două litere (mică – mare) este 32.
- cifrele au coduri consecutive: caracterul '0' are codul 48, caracterul '1' are codul 49, etc. *Observăm că caracterul '0' nu are codul ASCII 0, ci 48.
- caracterul spaţiu este un caracter imprimabil. **Spaţiul are codul ASCII 32**.

8. Instrucțiunea switch

Instrucțiunea switch permite executarea unor instrucțiuni, în funcție de egalitatea unei expresii cu anumite valori numerice constante:

Sintaxa:

```
switch ( Expresie )
{
   case Constanta_1:
        Grup_Instructiuni_1;
        break;
   case Constanta_2:
        Grup_Instructiuni_2;
        break;
   ...
   case Constanta_N:
        Grup_Instructiuni_N;
        break;
   default:
        Grup_Instructiuni_default;
        break;
}
```

Observații:

- Valorile din clauzele case trebuie să fie constante întregi.
- Prezență instrucțiunii break; nu este obligatorie, dar lipsa ei modifică modul de execuție al instrucțiunii. În exemplul de mai jos:

```
switch(n)
{
case 1: cout << "n = 1\n";
case 2: cout << "n = 2\n"; break;
case 3: cout << "n = 3\n"; break;
}
```

dacă valoarea lui n este 1, se va afișa:

```
n = 1
n = 2
```

Mai exact, se execută toate instrucțiunile de la clauza case corespunzătoare valorii expresiei până la prima instrucțiune break; întâlnită.

Exemple:

În exemplul de mai jos se afișează numele zilei din săptămână în funcție de numărul ei.

```
#include <iostream>
using namespace std;
int main(){
int zi; cin >> zi;
 switch(zi)
{case 1: cout << "Luni\n"; break;
 case 2: cout << "Marti\n"; break;
 case 3:  cout << "Miercuri\n"; break;
 case 4: cout << "Joi\n"; break;
 case 5: cout << "Vineri\n"; break;
 case 6:
 case 7: cout << "WEEKEND!!!\n"; break;
 default: cout << "Numarul zilei este incorect\n"; break; }
 return 0;}
```

Observați mai sus că în clauza case 6: nu avem instrucțiuni. Dacă variabila zi are valoarea 6 sau 7 se va afișa:

```
WEEKEND!!!
```

În exemplul de mai jos se afișează rezultatul unei operații în funcție de operatorul introdus: *+, -,* sau /*.

```
# include <iostream>
using namespace std;
int main()
{  char op;
   float nr1, nr2;
   cout << "Introdu un operator:  + sau - sau * sau /: ";  cin >> op;
   cout << "Introdu doi operanzi: ";   cin >> nr1 >> nr2;
   switch(op)
   {     case '+':
            cout << nr1+nr2; break;
      case '-':
            cout << nr1-nr2; break;
      case '*':
            cout << nr1*nr2; break;
      case '/':
            cout << nr1/nr2; break;
      default:
            // Daca operatorul este altul decât  +, -, * or /, se va afisa mesajul:
            cout << "Operator incorect!";  break;
   }
   return 0;}
```

Problema 2 – porumb* - OJI 2012

Locuitorii planetei *Agria*, numiţi agri, au hotărât ca în celebrul an 2012 să le explice pământenilor cum trebuie cules „eficient” un rând cu n porumbi, numerotaţi, în ordine, cu 1, 2, 3,..., n.

Cei n porumbi sunt culeşi de mai mulţi agri. Primul agri merge de-a lungul rândului, plecând de la primul porumb şi culege primul porumb întâlnit, al treilea, al cincilea şi aşa mai departe până la capătul rândului.

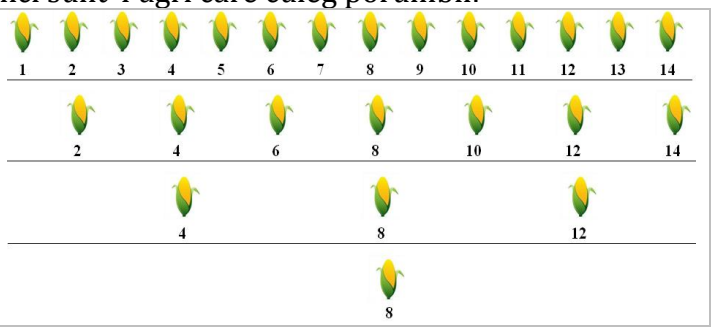
Atunci când ajunge la capătul rândului, porneşte al doilea agri şi culege porumbi respectând aceeaşi regulă ca şi primul agri.

Metoda se repetă până când toţi porumbii sunt culeşi.

Pământeanul Ionel încearcă să descopere ce ascunde această metodă şi se gândeşte câţi porumbi culege primul agri, câţi agri culeg un rând cu n porumbi, la a câta trecere este cules porumbul cu numărul x şi care este numărul ultimului porumb cules.

Exemplu: Dacă pe un rând sunt n=14 porumbi atunci sunt 4 agri care culeg porumbii:

- primul agri culege porumbii 1,3,5,7,9,11,13;
- al doilea agri culege porumbii 2,6,10,14;
- al treilea agri culege porumbii 4 şi 12;
- ultimul agri culege porumbul 8.



Cerinţe. Pentru a-l ajuta pe Ionel să descopere secretul acestei metode, scrieţi un program care citeşte cele două numere naturale n şi x şi care determină:

a) numărul de porumbi culeşi de primul agri;
b) numărul de agri care culeg şirul de n porumbi;
c) numărul trecerii la care este cules porumbul cu numărul x;
d) numărul ultimului porumb cules.

Date de intrare. Fişierul porumb.in conţine pe prima linie, separate printr-un spaţiu, cele două numere naturale n şi x cu semnificaţia din enunţ.

Date de ieşire. Fişierul de ieşire porumb.out va conţine patru linii:

- pe prima linie se va scrie un număr natural reprezentând numărul de porumbi culeşi de primul agri;
- pe a doua linie se va scrie un număr natural reprezentând numărul de agri care culeg cei n porumbi;
- pe a treia linie se va scrie un număr natural, reprezentând numărul trecerii la care este cules porumbul x;
- pe a patra linie se va scrie un număr natural, reprezentând numărul ultimului porumb cules.

Restricţii şi precizări:

- $1 \le x \le n \le 1000000000$
- Trecerile se numerotează în ordine, începând cu valoarea 1.

Exemplu:

porumb.in	porumb.out	*Explicaţii*
14 4	7 4 3 8	7 reprezintă numărul de porumbi culeşi de primul agri. Sunt 4 agri care culeg rândul cu n=14 porumbi. Porumbul x=4 este cules la a 3-a trecere iar ultimul porumb cules are numărul 8.

Rezolvare:

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

ifstream fin("porumb.in");
ofstream fout("porumb.out");

int main()
{       int n, x;
        int m=1,k=0,nr=0;
        fin>>n>>x;
        fout<<(n+1)/2<<'\n';
        while(m<=n){
                m = m*2;}
        m = m/2;
        fout<<m<<'\n';
        while(x%2==0)
        {
                x = x/2;
                k++;
        }
        fout<<k+1<<'\n';
        while(n!=0)
        {
                nr++;
                n = n/2;
        }
        fout<<nr ;
        return 0;
}
```

Problema 1 – beţe* - OJI 2013

Ana şi Bogdan au găsit la bunicul lor o cutie cu N beţe de aceeaşi lungime. După câteva minute de joacă urmează cearta. Bunicul le-a propus să rupă cele N beţe şi apoi Ana să primească fragmentele din mâna stângă, iar Bogdan fragmentele din mâna dreaptă. Zis şi făcut. Copiii au luat fragmentele, le-au numerotat fiecare cu numere de la 1 la N, le-au măsurat şi acum îşi doresc să lipească fragmentele primite, dar mai au nevoie de câteva informaţii.

Cerinţe

Cunoscând N numărul de beţe, $a_1,a_2,...,a_N$ lungimile fragmentelor primite de Ana şi $b_1,b_2,..., b_N$ lungimile fragmentelor primite de Bogdan, să se scrie un program care să determine:

a) lungimea iniţială a beţelor;
b) lungimea celui mai lung băţ care se poate obţine prin lipirea unui fragment aparţinând Anei cu un fragment care aparţine lui Bogdan;
c) numărul beţelor de lungime maximă care se pot obţine prin lipirea unui fragment aparţinând Anei cu un fragment care aparţine lui Bogdan.

Date de intrare: Fişierul de intrare bete.in conţine pe prima linie numărul natural N reprezentând numărul de beţe. Pe a doua linie sunt N numere naturale $a_1,a_2, ...,a_N$ reprezentând lungimile fragmentelor primite de Ana şi pe a treia linie sunt N numere naturale $b_1,b_2,...,b_N$ reprezentând lungimile fragmentelor primite de Bogdan.

Date de ieşire: Fişierul de ieşire bete.out va conţine trei linii. Pe prima linie se va scrie numărul natural L reprezentând lungimea iniţială a beţelor, pe a doua linie se va scrie numărul natural K reprezentând lungimea celui mai lung băţ care se poate obţine prin lipirea unui fragment aparţinând Anei cu un fragment care aparţine lui Bogdan, iar pe a treia linie se va scrie numărul natural P reprezentând numărul beţelor de lungime maximă care se pot obţine prin lipirea unui fragment aparţinând Anei cu un fragment care aparţine lui Bogdan.

Restricţii

- $1 \le N \le 1000$
- $1 \le a_i \le 10000$, $(1 \le i \le N)$
- $1 \le b_i \le 10000$, $(1 \le i \le N)$
- $1 \le L \le 20000$
- $1 \le K \le 20000$
- $1 \le P \le 1000$
- Odată lipite două fragmente, acestea nu se pot dezlipi.

Exemplu

bete.in	bete.out	*Explicaţii*
6 2 6 7 1 3 5 5 4 7 8 9 3	10 16 1	Lungimea iniţială este 10, lungimea maximă este 16 şi se poate forma un singur băţ de lungime 16.

Rezolvare:

```
#include<fstream>
using namespace std;
ifstream fin("bete.in");
ofstream fout("bete.out");
int n,l,k,a,b,i,j,s,ma,mb,na,nb;
int main()
{
  fin>>n;
  fin>>a;
  ma=a; na = 1;s = a;
  for(i=2; i<=n; i++)
       {
          fin>>a;
          if (ma<a) {ma=a; na = 1;}
          else if (ma==a) na ++;
          s =s + a;
       }
  fin>>b;
  mb = b; nb =1;s = s + b;
  for(i=2; i<=n; i++)
       {
          fin>>b;
          if(mb<b){mb = b; nb = 1;}
          else if(mb == b)nb ++;
          s = s + b;
       }
  l=s/n;
  fout<<l<<'\n'<<ma+mb<<'\n'<<min(na,nb)<<'\n';
  fout.close();
  return 0;}
```

Problema 1 – colier* - OJI 2016

Maria are în camera sa N mărgele aşezate una lângă alta. Pe fiecare dintre ele este scris un număr natural format din cifre nenule distincte. Pentru fiecare mărgea, Maria şterge numărul şi în locul său scrie altul, având doar două cifre, respectiv cifra minimă şi cifra maximă din numărul scris iniţial, în ordinea în care aceste cifre apăreau înainte de ştergere. Acum Maria consideră că mărgelele sunt de două tipuri, în funcţie de numărul de două cifre scris pe ele: tipul 1 (cele care au cifra zecilor mai mică decât cifra unităţilor) şi tipul 2 (celelalte). Folosind mărgelele, fetiţa doreşte ca prin eliminarea unora dintre ele (dar fără să le schimbe ordinea celorlalte) să obţină un colier circular cât mai lung care să respecte proprietatea că oricare două mărgele vecine ale sale sunt de tipuri diferite. În colierul format cu mărgelele rămase după eliminare se consideră că prima mărgea este vecină cu ultima.

Cerinţe:

1) determinaţi numărul de mărgele de tipul 1;

2) determinaţi numărul maxim de mărgele pe care le poate avea colierul;

Date de intrare

Fişierul colier.in conţine pe prima linie un număr natural T. Pe linia a doua se găseşte un număr natural N. Pe linia a treia sunt N numere naturale ce reprezintă, în ordine, valorile scrise iniţial pe mărgele. Aceste numere sunt separate prin câte un spaţiu.

Date de ieşire

Dacă valoarea lui T este 1, se va rezolva numai punctul 1) din cerinţe. În acest caz, fişierul de ieşire colier.out va conţine pe prima linie un număr natural reprezentând răspunsul la cerinţa 1).

Dacă valoarea lui T este 2, se va rezolva numai punctul 2) din cerinţe. În acest caz, fişierul de ieşire colier.out va conţine pe prima linie un număr natural reprezentând răspunsul la cerinţa 2).

Restricţii şi precizări

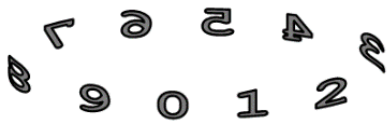
- $1 \le N \le 50\,000$;
- Numerele scrise iniţial pe mărgele au cifrele distincte, nu conţin cifra 0 şi sunt cuprinse între 12 şi 987654321;
- T va fi 1 sau 2;
- Pentru obţinerea colierului, Maria poate decide să nu elimine nici o mărgea;
- Colierul obţinut poate fi format şi dintr-o singură mărgea;
- Pentru teste în valoare de 20 de puncte avem T = 1 şi toate numerele scrise iniţial pe mărgele au două cifre;
- Pentru teste în valoare de 30 de puncte avem T = 1 şi dintre numerele scrise iniţial pe mărgele sunt şi unele cu mai mult de două cifre;
- Pentru teste în valoare de 50 de puncte avem T = 2.

Exemple:

colier.in	colier.out	Explicaţie
1 5 12 678 312 24 938	3	Numerele scrise de Maria pe mărgele vor fi, în ordine: 12 68 31 24 93. Trei dintre ele (12, 68 şi 24) sunt de tipul 1. (T fiind 1 se rezolvă doar cerinţa 1)
colier.in	colier.out	Explicaţie
2 5 12 678 312 24 938	4	Numerele scrise de Maria pe mărgele vor fi, în ordine: 12 68 31 24 93. Eliminând mărgeaua de pe poziţia 1 sau pe cea de pe poziţia 2 şi aşezându-le pe celelalte circular obţinem un colier cu 4 mărgele în care oricare două vecine sunt de tipuri diferite. (T fiind 2 se rezolvă doar cerinţa 2). Maria este obligată să elimine una din cele două mărgele, altfel ar exista mărgele vecine de acelaşi tip.

```
#include<fstream>
using namespace std;
ifstream f("colier.in");
ofstream g("colier.out");
int main()
{
   int n, nr=1, tip1=0, tip2=0,x,c,u,k=0,cmin=10,cmax=0,pcmin,pcmax,p,i,t;
   f>>t>>n>>x;
   while (x)
   {                    c=x%10; k++;
      if (c<cmin){
         cmin=c; pcmin=k;
      }
      if (c>cmax){
         cmax=c;pcmax=k;
      }
      x = x/10;
   }
   if (pcmin>pcmax){
      tip1++; u=p=1;
   }
   else
   {
      tip2++; u=p=2;
   }
   for (i=2; i<=n; i++) {
      f>>x;
      k=0;  cmin=10; cmax=0;
      while (x) {
         c=x%10; k++;
         if (c<cmin){
            cmin=c; pcmin=k;
         }
         if ( c > cmax){
            cmax=c; pcmax=k;
         }
         x= x/10;
      }
      if (pcmin >pcmax){
         tip1++;
         if (u!=1) {nr++; u=1;}
      }
      else
      {
         tip2++;
         if (u!=2) {nr++; u=2;}
      }
   }
if (u==p) nr--;
if (t==1) g<<tip1<<'\n';
else g<<nr<<'\n';
return 0;
}
```

Problema 2 robot **



Paul doreşte să înveţe cum să programeze un robot. Pentru început s-a gândit să construiască un robot format dintr-un mâner, 10 butoane aranjate circular şi un ecran. Pe butoane sunt scrise, în ordine crescătoare, cifrele de la 0 la 9, ca în figură. Un roboprogram va fi format dintr-o secvenţă de instrucţiuni. Instrucţiunile pot fi:

Instrucţiune	*Semnificaţie*
Dp	Mânerul robotului se deplasează spre dreapta cu p poziţii (p este o cifră)
Sp	Mânerul robotului se deplasează spre stânga cu p poziţii (p este o cifră)
A	Este apăsat butonul în dreptul căruia se află mânerul robotului şi pe ecran apare cifra scrisă pe buton
T	Terminarea programului (se utilizează o singură dată la final şi este precedată de cel puţin o instrucţiune A)

Iniţial mânerul robotului este plasat în dreptul butonului 0, iar ecranul este gol.
De exemplu, în urma executării roboprogramului D4AS1AAD6AT robotul apasă butoanele pe care sunt scrise cifrele 4, 3, 3, 9, iar pe ecran va apărea 4339.

Cerinţe
Să se scrie un program care rezolvă următoarele cerinţe:
1. citeşte un roboprogram şi determină numărul de cifre afişate pe ecran după executarea roboprogramului;
2. citeşte un roboprogram şi determină cifrele afişate pe ecran după executarea roboprogramului;

Date de intrare
Fişierul de intrare robot.in conţine pe prima linie un număr natural C, reprezentând cerinţa care urmează să fie rezolvată (1 sau 2). Dacă C=1 sau C=2, pe a doua linie a fişierului se află un roboprogram.

Date de ieşire
Fişierul de ieşire robot.out va conţine o singură linie. Dacă C=1, pe prima linie se va scrie un număr natural reprezentând numărul de cifre afişate pe ecran după executarea roboprogramului din fişierul de intrare.
Dacă C=2, pe prima linie vor fi scrise cifrele afişate pe ecran în urma executării roboprogramului din fişierul de intrare.

Restricţii
- $0 \le N \le 1000000000$
- Lungimea roboprogramului citit din fişierul de intrare sau scris în fişierul de ieşire este cel mult 1000 de caractere.
- Dacă mânerul este plasat în dreptul butonului 0 şi se deplasează spre dreapta, se va îndrepta către butonul 1; dacă deplasarea este spre stânga, se va îndrepta către butonul 9.

Exemple

robot.in	**robot.out**	**Explicaţii**
1 D1AD2AS1AT	3	C=1, pentru acest test se rezolvă cerinţa 1. Se afişează pe ecran 3 cifre (132)

robot.in	**robot.out**	**Explicaţii**
2 S0AD2AS1AT	021	C=2, pentru acest test se rezolvă cerinţa 2. Mânerul robotului se deplasează cu 0 unităţi la stânga, deci rămâne în dreptul butonului 0 şi apasă, apoi se deplasează 2 unităţi spre dreapta şi ajunge în dreptul butonului 2, apasă, apoi se deplasează 1 unitate la stânga şi ajunge în dreptul butonului 1 şi apasă acest buton $\Rightarrow$021.

Rezolvare:

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

ifstream fin("robot.in");
ofstream fout("robot.out");

int cerinta, n;
int main()
{char c;
 int nr, st, dr, poz, zero, cat, cifra;
 fin>>cerinta;
 if (cerinta==1)
   {c='*'; nr=0;
   while (c!='T')
        {
         fin>>c;
         if (c=='A') nr++;
        }
    fout<<nr<<'\n';
    fout.close();
    return 0;
   }
 if (cerinta==2)
   {c='*'; nr=0; poz = 0;
   while (c!='T')
        {
         fin>>c;
         if (c=='A') fout<<poz;
           else
           if (c=='D')
             {
              fin>>c;
              cat=c-'0';
              poz=(poz+cat)%10;
             }
             else
             if (c=='S')
               {
                fin>>c;
                cat = c-'0';
                poz = poz - cat;
                if(poz<0) poz = poz + 10;
               }
        }
    fout<<'\n';
    fout.close();
    return 0;
   }
```

Olimpiada Municipala de Informatica, Iaşi, 2015

Cristina şi Alina sunt eleve în clasa a V-a şi sunt foarte bune prietene. Le place ca în pauze să se provoace reciproc cu câte o problemă. De data aceasta, e rândul Cristinei să propună o problemă Alinei. Ea îi cere ca dintr-un set de mai multe numere naturale să le găsească pe cele centrale. Bineînţeles că mai întâi îi explică prietenei sale ce este un număr central: un număr care are proprietatea ca, după eliminarea primei şi a ultimei cifre, se obţine un nou număr care conţine numai cifre egale între ele. De exemplu, numărul **67771** este număr central pentru că, eliminând prima şi ultima cifră, se obţine numărul **777** care are toate cifrele egale între ele. Alina, care între timp a învăţat să programeze, intră imediat în jocul Cristinei, ştiind că va afla imediat rezultatul corect la problema propusă de prietena ei.
Cerinţa: Având la dispoziţie un set de numere pe care le primeşte pentru verificare, Alina trebuie să spună câte dintre acestea sunt numere centrale.
Date de intrare: Fişierul de intrare **centrale.in** conţine pe prima linie, numărul natural N care reprezintă numărul de numere ce trebuie verificate. Pe următoarea linie se găsesc cele N numere naturale, separate prin câte un spaţiu.
Date de ieşire: Fişierul de ieşire **centrale.out** va conţine o singură linie pe care va fi scris numărul de numere centrale găsite între cele N numere ce trebuie verificate.
Restricţii şi precizări : 1 ≤ N ≤ 100
• Fiecare număr din setul dat are cel puţin 3 cifre şi cel mult 9 cifre
Exemplu

centrale.in	centrale.out	**Explicaţie**
5 81318 71117 2258 933 21110	3	Dintre cele 5 numere din setul dat, sunt 3 numere centrale: 71117, 933 şi 21110.

Rezolvare:

```
        #include <fstream>
        using namespace std;
ifstream fin("centrale.in");
ofstream fout("centrale.out");
        int n,x,i,a,k;
        int main(){
        fin>>n;
        k=0; int nr=0;
        for(i=1;i<=n;i++)
{
        fin>>x;
        x=x/10;
        a=x%10;
while(x>9 && x%10==a)
        x=x/10;
        if(x<=9)
        k++;
 }
        fout<<k<<'\n';
        fout.close();
        return 0;
}
```

Bibliografie

1. Adrian Niţă, Carmen Popescu, Diana Nicoleta Chirilă, Maria Niţă, Informatică şi TIC, Editura Corint

2. LICA Dana, PAŞOI Mircea – "Informatica - Fundamentele Programării", Bucureşti, Editura L&S Soft, 2005;

3. BĂICAN Diana Carmen, CORITEAC Melinda Emilia, Informatică şi TIC, Manual pentru clasa a VI-a Editura Didactică şi Pedagogică;

4. MILOŞESCU Mariana, Informatică, Manual pentru clasa a IX-a, Editura Didactică şi Pedagogică R.A.

5. MINCĂ Carmen, BORIGA Radu, Informatica, Manual pentru clasa a IX-a, Editura L&S Info-mat

Bibliografie web:

www.pbinfo.ro

www.campion.ro

www.varena.ro

Anexa 1

Datele au următoarele caracteristici:

Nume (unic, primul caracter nefiind cifră): este o succesiune de caractere cu rol de identificare.

Tip: se referă la o anumită categorie de valori şi la operaţiile ce pot fi efectuate asupra acestora.

Valoare: în funcţie de tipul precizat.

Tipurile întregi

Tipurile întregi permit memorarea de valori întregi. Tipul de bază este int. O dată de tip int poate memora valori întregi cuprinse între -2^{31} şi $2^{31}-1$.

Tipurile întregi diferă prin numărul de octeţi necesari pentru memorarea datei, tipul datei (cu semn sau fără semn) şi implicit intervalul de valori pe care le pate lua respectiva dată. Tipurile întregi sunt:

Denumire tip	Reprezentare	Interval de valori	-
int	4 octeţi cu semn	-2^{31} ... $2^{31}-1$	-2147483648 ... 2147483647
unsigned int	4 octeţi fără semn	0 ... $2^{32}-1$	0 ... 4294967295
long int	4 octeţi cu semn	-2^{31} ... $2^{31}-1$	-2147483648 ... 2147483647
unsigned long int	4 octeţi fără semn	0 ... $2^{32}-1$	0 ... 4294967295
short int	2 octeţi cu semn	-2^{15} ... $2^{15}-1$	-32768 ... 32767
unsigned short int	2 octeţi fără semn	0 ... $2^{16}-1$	0 ... 65535
long long int	8 octeţi cu semn	-2^{63} ... $2^{63}-1$	
unsigned long long int	8 octeţi fără semn	0 ... $2^{64}-1$	
char	1 octet cu semn	-2^{7} ... $2^{7}-1$	-128 ... 127
unsigned char	1 octet fără semn	0 ... $2^{8}-1$	0 ... 255

Tipurile char şi unsigned char memorează valori întregi. La afişarea unei date de acest tip nu se va afişa numărul pe care îl memorează ci caracterul care are codul ASCII egal cu acel număr. Operaţia de citire a unei date de acest tip este similară.

Tipurile reale – în virgulă mobilă

Memorează valori reale, reprezentate prin mantisă şi exponent. În acest mod se pot reprezenta valori foarte mari, dar precizia reprezentării poate fi slabă – numărul de cifre semnificative memorate poate fi mult mai mic decât numărul de cifre din număr.

Tipurile reale sunt:

- float – se reprezinta pe 4 octeţi;
- double – se reprezinta pe 8 octeţi;
- long ouble – se reprezinta pe 10 octeţi;

Tipul logic (bool)

Anumite operaţii care se fac cu datele au ca rezultat valori de adevăr: **adevărat** sau **false**. În anumite limbaje de programare există un tip de date care memorează exact aceste două valori.

În limbajul C++ există tipul bool. Acest tip conţine două valori: literalii true şi false. De fapt, acestea sunt redenumiri ale valorilor 1 şi 0.

Anexa 2

O **expresie** este alcătuită din **operanzi** și **operatori**. Operanzii reprezintă datele cu care se fac operațiile, iar operatorul este simbolul care stabilește ce operație se face cu operanzii. Din punct de vedere a numărului de operanzi, operațiile (operatorii) pot fi:

- **unare** – se aplică unui singur operand (`-7`, operația de schimbare a semnului unui număr);
- **binare** – se aplică la doi operanzi (de exemplu adunarea numerelor, `2 + 5`);

Operatorii aritmetici binari: `+`, `-`, `*`, `/`, `%`

- `+` : adunarea a două numere
- `-` : scăderea a două numere
- `*` : înmulțirea a două numere
- `/` : împărțirea a două numere
- `%` : restul împărțirii a două numere întregi (**modulo**)
- În C++ nu există un operator pentru ridicarea la putere!

Adunarea, scăderea și înmulțirea se comportă conforma așteptărilor, ca la matematică. Operația de împărțire și operația modulo necesită niște explicații suplimentare.

Împărțirea întreagă și împărțirea zecimală

Operația de împărțire are două moduri de lucru, în funcție de tipul operanzilor.

- Dacă operanzii sunt de tip **întreg** (`int`, `short`, `char`, `unsigned`, etc.), se va realiza împărțirea întreagă, iar rezultatul operației `/` este câtul împărțirii întregi.
- Dacă operanzii sunt de tip **real** (`float`, `double`, `long double`), se va realiza împărțirea zecimală, iar rezultatul operației `/` este rezultatul acestei împărțiri, "cu virgulă".

Exemple

Operatorul modulo `%`

Operația modulo are sens numai dacă ambii operanzi sunt de tip întreg – împărțirea cu rest are sens numai în această situație. Iată câteva exemple:

Operatorul modulo este util în multe situații. El poate fi utilizat pentru a afla ultima cifră a unui număr natural: ultima cifră a lui `325` este `325 % 10` adică `5`, sau pentru a verifica dacă un număr `n` este divizor al lui `m`. În caz afirmativ, `m % n` este `0`.

Operatorii relaționali: `<`, `>`, `<=`, `>=`, `== (egal)`, `!= (diferit)`

Un operatori relațional stabilește dacă între două numere (operanzii) are loc o anumită relație. Rezultatul acestei operații este **adevărat** sau **fals**.

Un dintre cele mai frecvente erori este folosirea pentru operația de egalitate a operatorului `=`, în loc de `==`. Operatorul `=` reprezintă **operația de atribuire**!

Operatorii logici: `! (not)`, `|| (or)`, `&& (and)`

În C++, operatorii logici pot fi aplicați oricăror valori numerice, și au ca rezultat una din valorile `0` sau `1`. În exemplele de mai jos vom folosi literalii `true` și `false`, de tip `bool`.

Negația: **`not true` este `false`;**
`not false` este `true`.

Sau logic: or

or	false	true
false	false	true
true	true	true

or (+)	0	1
0	0	1
1	1	1

Şi logic: and

and	false	true
false	false	false
true	false	true

and (*)	0	1
0	0	0
1	0	1

Legile lui De Morgan

Fie p şi q două valori booleene (pot fi rezultatele unor expresii, de exemplu). Atunci:

- not (p and q) == not p or not q
- not (p or q) ==not p and not q

Operatorul de atribuire =

Atribuirea este operaţia prin care **o variabilă primeşte valoarea unei expresii**:

variabila = expresie

Exemple de expresii:

1. Evaluarea expresiei: (2+3*(5-7/2))*3+2*3%4 =(2+3*2)*3+6%4=(2+6)*3+2=8*3+2=24+2=26

2. Stabilirea valorii de adevăr a următoarei expresii: Calculele matematice au prioritate

2*(3+140/3%7) +12*7/3>10 or 5+2*(7/2+17%3/4) < =5
2*(3+46%7) +84/3>10 or 5+2*(3+2/4) <=5
2*(3+4) +28>10 or 5+2*3 < = 5
42>10 or 11< = 5
True or false **rezultă true**

Concluzii:

1. Într-o expresie informatică toate parantezele sunt rotunde. Calculele se efectuează începând cu paranteza cea mai din interior.

2. Pentru evaluarea expresiilor se respecta regulile de baza învăţate la matematica. Se evaluează întâi expresiile dintre parantezele rotunde, apoi se executa operaţiile în ordinea priorităţii lor. Daca exista operaţii cu aceeaşi prioritate, ele se executa în ordine, de la stânga la dreapta.

Prioritate	**Operatori**	**Simbol**
1 (cea mai mare)	Negaţia logică	NOT
2	Aritmetici multiplicativi	* / div % (mod)
3	Aritmetici aditivi	+ -
4	Relaţionali	< > <= >= == !=
5	SI logic	AND
6 (cea mai mica)	SAU logic	OR

Daca **a** şi **b** sunt doua variabile care conţin valori numerice, atunci avem relaţiile:
1. Expresia **NOT (a >b)** este echivalenta cu expresia **a<=b.**
2. Expresia **NOT (a>=b)** este echivalenta cu expresia **a < b.**

Exemple care folosesc instrucţiunea de atribuire

1. Ce se va afişa în urma executării următorului algoritm,ştiind ca de la tastatura se introduce valoarea **15749**?

```
x=23852;
y=x % 100 ;
x=x /1000;
x=x*100+y;
```

Variabila x	Variabila y
23852	**52**
23	
2352 (=**23***10+**52**)	

Observaţie: În calculul valorii finale a lui x se foloseşte valoarea: 23, care este cea mai recentă, nu valoarea iniţială care era **23852**.
Deci, la final, se va afişa valoarea **2352**.

2. Care sunt valorile variabilelor **x**, **y** si **z**, in urma executării secvenţei de instrucţiuni?

```
x=1334;
y=x/100;
x=x*2/100;
z=(x+2)/2;
```

x	y	z
1334		
	13	
26		
		14

3. Dacă valoarea iniţială a variabilei **x** este **275**, ce valoare va avea **x** în urma efectuării următoarelor atribuiri?

```
a =x % 10;
x =x / 10;
b = x % 10;
x = x /10;
x  =(x * 10 + a) * 10 + b;
```

x (275)	a	b
	5	
27		
		7
2		
257		

4. Care sunt valorile variabilelor **a** si **s**, in urma executării secvenţei de instrucţiuni?

```
s=0;
a=1;
s=s+a;
a=a+1;
s=s+a;
a=a+1;
s=s+a;
a=a+1;
s=s+a;
```

a	s=0
1	**1**
2	**3**
3	**6**
4	**10**