Divizibilitate

- x divizibil cu k(x multiplu de k/k este divizor al lui x):

```
if(x%k==0)
    //prelucrează divizorul k
```

- x nu e divizibilcu k:

```
if(x%k!=0)....
//sau
if(x%k)....
```

- criterii de divizibiliate cu 2,3,4,5,6,7,9,11 etc.(de văzut condițiile matematice specifice)

- parcurgerea tuturor divizorilor lui x:

```
for(int d=1;d<=x;d++)
        if(x%d==0) {prelucreaza d}
```

- parcurgerea tuturor divizorilor lui x eficient:

```
for(d=1;d*d<=x;d++)
  if(x%d==0)
     {prelucreaza d;
      d1=x/d;     //divizorul pereche
      if(d1!=d)   //altfel pentru patratele perfecte ar parcurge divizorul care este egal cu sqrt(x) de doua ori
         prelucreaza d1;
      }
```

1 si x se pot analiza separate înainte de for(vezi cazul cand x=1)

- parcurgerea divizorilor proprii ai lui x:

```
for(int d=2;d<=x/2;d++)
        if(x%d==0) {prelucreaza d}
```

- cel mai mare divizor propriu sau mesajul "nu exista" dacă nu are divizori proprii:

```
d=x/2;
while(x%d!=0 &&d>=2)
        d--;
if(d>=2) cout<<d;
else cout<<"nu exista";
```

- cel mai mare divizor diferit de x(îl găseşte măcar pe 1):

```
d=x/2;
while(x%d!=0)
        d--;
 cout<<d;
```

- cel mai mic divizor propriu(cel mai mic factor prim diferit de număr):

```
d=2;
while(x%d!=0 &&d<=x/2)
        d++;
if(d<=x/2) cout<<d;
else cout<<"nu exista";
```

- cel mai mic divizor diferit de 1:

```
d=2;
while(x%d!=0 )
        d++;
cout<<d;
```

Observație: pentru numărul x, odată identificat cel mai mic divizor propiu d1, dacă există, , cel mai mare divizor propriu d2 se va determina conform regulii matematice d1*d2=x deci d2=x/d1. Există posibilitatea ca d1 să fie diferit de d2, dar și posibilitatea să fie egale (în cazul numerelor care sunt pătratele unor numere prime ex. 49)

Factori primi:

- puterea factorului f în descompunerea în factori primi a lui x

```
cin>>x>>f;
p=0;
while(x%f==0)
{x=x/f;
p++;
}
cout<<p;
```

- verificaţi dacă x este putere a lui f

```
p=0;
while(x%f==0)
{x=x/f;
p++;
}
if(x==1) cout<<"da "<<f<<" la puterea "<<p;
else cout<<"nu";
```

- toți factorii primi

```
f=2;
while(x!=1)
{
  p=0;
  while(x%f==0)
    {x=x/f;
     p++;
    }
    if(p!=0) cout<<f<<' '<<p;
 f++;
}
```

Uneori se analizează mai întâi factorul 2 și apoi ceilalți factori pornind de la 3 și din 2 în 2

- număr liber de pătrate (nu conține nici un factor prim la putere pară)

```
f=2; ok=1;
while(x!=1)
{
  p=0;
  while(x%f==0)
    {x=x/f;
     p++;
    }
    if(p!=0 && p%2==0) ok=0;
 f++;
}
if(ok) cout<<"da";
else cout<<"nu";
```

Aplicație: se dau 2 nr naturale a și b. Afișați cel mai mare divizor comun al lui a cu b și care poate fi scris ca produs de numere prime

Ex: pentru 72 și 60 se va afișa 6.(cmmdc pentru 72 și 60 este 12 apoi pentru 12 gasim produsul a cel puțin doi factori primi 6=2*3. Deci răspunsul este 6)

```
cin>>a>>b;
x=cmmdc(a,b);
y=1;
nr=0;d=2;
while(x!=1)
{p=0;
 while(x%d==0)
    {x=x/d;p++;}
if(p!=0)   //am gasit um factor îl adăugăm la y și îl numărăm
   {y=y*d; nr++;}
d++;
}
if(nr>=2) cout<<y;  //daca y este produs de cel putin 2 factori
else cout<<"nu exista";
```

- număr perfect(este egal cu suma divizorilor până la jumătate)

```
s=1;
for(d=2;d<=x/2;d++)
   if(x%d==0)
      s=s+d;
if(x==s) cout<<"da";
else cout<<"nu";
```

sau eficient

```
cin>>x;
s=1; //orice numar nenul il are pe 1 ca divizor
for(d=2;d*d<=x;d++)
            if(x%d==0)
              {s+=d;
               dp=x/d;
               if(dp!=d)
                        s+=dp;
             }
if(s==x) cout<<"este perfect";
```

- număr prim

(*) numărând toţi divizorii

```
nr=0;
for(d=1;d<=x;d++)
   if(x%d==0)
      nr=nr+1;
if(nr==2) cout<<"da";
else cout<<"nu";
```

(*) numărând divizorii proprii

```
nr=0;
for(d=2;d<=x/2;d++)
   if(x%d==0)
      nr=nr+1;
if((x>=2) &&(nr==0)) cout<<"da";
else cout<<"nu";
```

(*) căutând cel mai mic factor prim diferit de număr(dacă există numărul nu este prim

```
nr=0;
for(d=2;d*d<=x;d++)
   if(x%d==0)
      nr=nr+1;
if((x>=2) &&(nr==0)) cout<<"da";
else cout<<"nu";
```

(*)Eficient: dacă găsesc un divizor(factor), nu are rost să căutăm mai departe

```
nr=0;
ok=1;
d=2;
while(d*d<=x && ok==1)
   if(x%d==0)
      ok=0;
  else d++;
if((x>=2) &&(ok==1)) cout<<"da";
else cout<<"nu";
```

sau

(*)

```
ok=(x<2);
if(x%2==0) ok=0;
d=3;
while(d*d<=x && ok==1)
  {if(x%d==0)
      ok=0;
   else d=d+2;
  }
if(ok==1) cout<<"este prim";
```

sau subprogram

```
int prim(int x)
{if(x<2) return 0;
 if(x==2) return 1;
 if(x%2==0) return 0;
 for(d=3;d8d<=x;d++)
      if(x%d==0) return 0;
return 1;
}
```

Alte teste pentru număr prim

- suma tuturor divizorilor să fie egală cu numărul plus 1
- cel mai mare divizor diferit de 1 să fie chiar numărul
- cel mai mic divizor diferit de număr să fie 1
- ciurul lui Eratostene pentru a determina eficient numerele prime <= n

- **Număr aproape prim(**să poată fi scris ca produs de două numere prime distincte)

```
cin>>x;
ok=0;
for(d=2;d*d<=x;d++)
     if(x%d==0)
          {nr=0;
           for(dd=2;dd<=d/2;dd++)
                 if(d%dd==0)
                      nr=nr+1;
           if(nr!=0) ok=0;
           else
                   {d1=x/d;
                    if(d1!=d)
                           {nr=0;
                            for(dd=2;dd<=d1/2;dd++)
                                      if(d1%dd==0)
                                             nr=nr+1;
                            if(nr!=0) ok=0;
                            else {cout<<d<<' '<<d1; break;}  //am gasit o solutie si o afisam
                     }
           }
```

Sau

```
cin>>x;
ok=0;
a=2;ok=0;
while(a*a<x && ok==0)
   {if((x%a==0) && (x/a)!=a &&prim(a) && prim(x/a)) ok=1;
     else a++;
   }
if(ok) cout<<a<<’’<<x/a;
```

- **Conjectura lui Goldbach**: orice număr par mai mare sau egal cu 4 se poate scrie ca sumă de două numere prime. Afişaţi toate descompunerile unui număr par mai mare sau egal cu 4 dat

```
cin>>x;
if(x<4 || x%2!=0) cout<<"nu exista";
else
{for(a=2;a<=x/2;a++)
           {nr=0;
            for(d=2;d<=a/2;d++)
                  if(a%d==0)
                       nr=nr+1;
                  if(nr==0)
                     {b=x-a;
                      nr=0;
                      for(d=2;d<=b/2;d++)
                             if(b%d==0)
                                       nr=nr+1;
                       if(nr==0) cout<<a<<'  '<<b;
                       }
           }
}
```

- **Număr supraprim**: el şi toate prefixele lui sunt prime/sau el şi toate sufixele lui sunt prime.

```
//cu prefixe;
cin>>x;
ok=1;
while(x!=0 &&ok==1)
   { nr=0;
     for(d=2;d<=x/2;d++)
          if(x%d==0)
                   nr=nr+1;
       if(nr!=0) ok=0;
    x=x/10;
}
if(ok==1) cout<<"da";
else cout<<"nu";
```

// cu sufixe

```
cin>>x;
p=1;
y=0;
ok=1;
while(x!=0 &&ok==1)
   {y=y+(x*10)*p;
     nr=0;
     for(d=2;d<=y/2;d++)
          if(y%d==0)
                   nr=nr+1;
       if(nr!=0) ok=0;
    x=x/10;
    p=p*10;
}
if(ok==1) cout<<"da";
else cout<<"nu";
```

- Număr extraprim număr care este prim și pentru care toate numerele obținute prin permutarea(circulară eventual)a cifrelor sale sunt prime

- Afişaţi cel mai mic număr care are aceeaşi divizori primi ca şi numărul dat(ex. pentru 75 obţinem 15, iar pentru 7 obţinem 7)

(*)parcurgând divizorii

```
cin>>x;
p=1;//produsul divizorilor primi inclusiv x poate că numărul este prim
for(d=2;d<=x;d++)
   if(x%d==0)  //daca d este divizor verificam si daca este prim
      { nr=0;
       for(d1=2;d1*d1<=d;d1++)
             if(d%d1==0)
                   nr=nr+1;
       if(nr==0)
          p=p*d;
      }
cout<<p;
```

(*)parcurgând factorii primi

```
p=1;
```

```
d=2;
while(x!=1)
{
  e=0;
  while(x%d==0)
     {x=x/d;
       e++;
   }
  if(e!=0) p=p*d;
  d++;
}
cout<<p;
```

- Numere prietene (a şi b sunt prietene dacă a este egal cu suma divizorilor lui b până la jumătate şi invers)

```
cin>>a>>b;
sa=0;
for(d=1;d<=a/2;d++)
   if(a%d==0)   sa+=d;
sb=0;
for(d=1;d<=b/2;d++)
   if(b%d==0)   sb+=d;
if(a==sb&&b==sa) cout<<"sunt prietene";
else cout<<"nu sunt prietene";
```

CMMDC

- prin scăderi repetate

```
cin>>a>>b;
if(a==0&&b==0) cout<<-1;
else
if(a*b==0) cout<<a+b;
else
{
while(a!=b)
{if(a>b) a=a-b;
  else b=b-a;
}
cout<<a;
}
```

- prin împărţiri repetate

```
cin>>a>>b;
if(a==0&&b==0) cout<<-1;
else
 if(a*b==0) cout<<a+b;
else
{
while(b!=0)
{r=a%b;
 a=b;
 b=r;
}
cout<<a;
}
```

- prin resturi repetate

```
if(a==0 &&b==0) cout<<-1;
else
while(a*b!=0)
  if(a>b) a=a%b;
  else b=b%a;
cout<<a+b;
```

CMMMC

- pe bază de cmmdc: se știe că cmmdc(a,b)*cmmcc(a,b)=a*b

```
cin>>x>>y;
if(x==0 || b==0) cout<<-1;
else
{a=x; b=y;
if(a*b==0) d=a+b;
else
{
while(b!=0)
{r=a%b;
 a=b;
 b=r;
}
d=a;
}
cout<<x*y/d;
}
```

- prin adunări repetate(numerele să fie nenule)

```
cin>>x>>y;
if(x==0||y==0) cout<<-1;
else
{a=x;b=y;
while(a!=b)
{if(a<b) a=a+x;
  else b=b+y;
}
cout<<a;
}
}
```

Criteriul de divizibilitate cu 2
Un număr natural este divizibil cu 2 dacă ultima cifră a sa este cifră pară (0,2,4,6,8)

Criteriul de divizibilitate cu 3
Un număr natural este divizibil cu 3 dacă suma cifrelor sale se divide cu 3.
Exemplu: 32139 se divide cu 3; 3+2+1+3+9=18

Criteriul de divizibilitate cu 9
Un număr natural este divizibil cu 9 dacă suma cifrelor sale se divide cu 9.
Exemplu. 21543057 se divide cu 9; 2+1+5+4+3+0+5+7=27

Criteriul de divizibilitate cu 4
Un număr natural este divizibil cu 4 dacă numărul format din ultimele două cifre ale numărului este divizibil cu 4.
Exemplu. 4 | 20**32** pentru că 4 | **32**
4 | 1**28** pentru că 4 | **28**.

Criteriul de divizibilitate cu 5
Un număr natural este divizibil cu 5 dacă ultima cifră a sa este 0 sau 5.

Criteriul de divizibilitate cu 25
Un număr natural este divizibil cu 25 dacă numărul format din ultimele două cifre ale numărului este divizibil cu 25.
Exemplu. 25 | 38**50** pentru că 25 | **50**

Criteriul de divizibilitate cu 11
Un număr natural este divizibil cu 11 dacă diferenţa dintre suma cifrelor situate pe locurile impare şi suma cifrelor situate pe locurile pare este multiplu al lui 11.
Exemplu: 1925 : 11=175; (9+5)-(1+2)=11
1012 : 11=92; (1+1)-(0+2)=0

7 CRITERII DE DIVIZIBILITATE CU 7
Numărul 7 se poate mândri cu numeroase zicători (măsoară de şapte ori şi taie o dată ;şapte dintr o lovitură; unul la muncă ,şapte la mâncare ;şapte zile -n săptămână;etc) dar şi cu diferite reguli de divizibilitate. Iată 7 dintre aceste reguli_
CRITERIUL NR 1 Se scrie numărul in baza 10 folosind puterile lui 10, se înlocuieşte numărul10 cu 3 şi se fac calculele; Dacă rezultatul obţinut se divide cu 7,atunci şi numărul iniţial se divide cu 7. Exemplu: fie numărul 5285; in baza 10 se scrie: 5.10 3 +2.10 2 +8.10+5 şi prin înlocuirea bazei 10 cu 3 se obţine 5.3 3 +2.3 2 +8.3+5 = 182 M7.deci 5285 M 7.
CRITERIUL NR 2 (o variantă a primei reguli) Se înmulţeşte prima cifră din stânga cu 3 şi se adună cu cifra următoare;rezultatul se înmulţeşte cu 3 şi se adună ci fra următoare ş.a.m.d. până la ultima cifră. Pentru simplificarea rezultatului se admite ca după fiecare operaţie să se scadă ,din rezultatul obţinut 7 sau multiplu de 7. Exemplu: fie numărul 5285; operaţiile sunt următoarele: 5. 3 =15 , 15+2=17,dar 17=7.2+3;se renunţă la 7.2 şi se continuă 3.3+8=17,17=7.2+3; 3.3+5=14 M7
CRITERIUL NR 3 Vom proceda ca la regula precedentă dar vom începe înmulţirea de la cifra unităţilor cu 5 de această dată:să exemplificăm pentru numărul 48902 2.5=10=7.1+3; (3+0).5=15=7.2+1; (1+9).5 =50 =7.7+1;(1+8).5=45 =7.6+3, 3+4 =7,deci numărul 48902 M7
CRITERIUL NR 4 Se dublează cifra unităţilor şi se scade din rezultat cifra zecilor; din nou se dublează rezultatul apoi se adună cu cifra sutelor;procedeul se continuă alternând scăderea a cu adunarea,Acolo unde este posibil rezultatul se poate micsora cu un multiplu al lui 7. Exemplu ;fie numărul 5943 3.2= 6 , 6- 4=2, 2 .2 =4, 4+9 =13, 13= 7+6 , 6.2 =12, 12-5 =7,deci numărul 5943 M . 7

CRITERIUL NR 5 Este o regulă comună a divizibilităţii cu 7, 11, 13. Se imparte numărul in clase: clasa unităţilor, clasa miilor, clasa milioanelor,etc. Dacă diferenţa sumelor grupelor numărului dat ,adunate din 2 în 2, se divide cu 7,cu 11 sau cu13, atunci numărul se divide cu 7, 11 sau13. Exemplu: aplicăm regula pentru numărul 55285783 (783+55) –285 =553 este divizibil cu 7
CRITERIUL NR 6 Este o regulă comună a divizibilităţii cu 7, cu 3 sau cu 19. Se dau deoparte ultimile două cifre ale numărului , iar la numărul rămas se adună numărul format din cele două cifre date deoparte înmultit cu 4; dacă e necesar se repetă procedeulpânăse obţine un rezultat a cărui divizibilitate cu 3, cu 7 cu 19 este evidentă. Exemplu: fie numărul 134064 64.4 = 256, 1340+256 = 1596; repetăm regula;96.4 =384, 15+384 = 399 numărul 399 se divide cu 7 şi cu 3
CRITERIU NR 7 Numarul natural N se divide cu 7 (cu 11 si cu 13) daca si numai daca diferenta nenegativa dintre cele doua numere obtinute din numarul natural dat prin taierea lui in doua, astfel ca la dreapta sa ramâna trei cifre, se divide cu 7 (cu 11 sau 13). Daca numarul are mai mullt de 6 cifre,impartim de la dreapta la stânga numărul in grupe de câte trei cifre .Dacă diferenţa dintre suma numerelor exprimate pringrupe de rang par şi suma grupelor de rang impar se divide cu7, 11, 13, numărul dat se divide cu 7, 11, 13. . Iată, in continuare alte propoziţii matematice ce pot fi folosite in rezolvarea problemelor : a) Dacă un număr de două cifre se divide cu 7, atunci numărul format din aceleaşi cifre scrise in ordine inversă, mărit cu cifra zecilor din numărul iniţial se divide cu 7. Exemplu 63 M7; prin urmare numărul 36+6 = 42 M7. b) Dacă un număr de trei cifre se divide cu 7 ,atunci numărul format din aceleaşi cifre scrise in ordine inversă, micşorat cu diferenta dintre cifra unităţilor si c sutelor numărului iniţial,se divide cu 7. Exemplu: numărul 126 se divide cu 7.Numărul 621—(6-1)=616 se divide cu 7 c) Dacă suma cifrelor unui număr cu trei cifre este egală cu 7,el se divide cu 7 numai dacă cifra zecilor este egală cu cifra unităţilor . Exemplu ;322 se divide cu 7deoarece 3+2+2 = 7.

Ciurul lui Eratostene

Matematicianul grec ERATOSTENE (275 - 194 î.Hr.) a aplicat o metodă inedită și ușoară pentru a determina dacă numerele sunt prime sau nu.
Se scrie șirul numerelor naturale de la 2 până la 100, ordonate crescător, sub forma unei liste.
Se taie din acest șir toți multiplii numerelor prime, astfel:

- numărul 2 este prim, vom tăia deci din acest șir toți multiplii lui 2;
- 3 este număr prim, deci vom tăia și toți multiplii lui 3;
- tot așa vom proceda și cu 5;
- apoi va urma 7;
- următorul număr prim este 11; însă, deoarece 7 × 7 = 49, este mai mic decât 100 și 11 × 11 = 121, este mai mare decât 100, toate numerele care au rămas în listă după procesul descris mai sus sunt numere prime;
- **Multiplii lui 2:** 4, 6, 8, 10, 12, 14, 16, 18, 20, ... 100
- **Multiplii lui 3:** 6, 9, 12, 15, 18, 21, 24, 27, ... 99
- **Multiplii lui 5:** 10, 15, 20, 25, 30, 35, 40, 45, ... 100
- **Multiplii lui 7:** 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91, 98
- **După ce eliminăm multiplii de 2, 3, 5 și 7, mai rămân:** 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, adică exact lista numerelor prime de la 2 până la 100

Varianta 1 – numarul de numere prime <=1000000

```
int MAXSIZE = 1000001;
int p[MAXSIZE];         //p[i] == 0 daca i este prim
int getTheNumber(int n) {
  int i, j, nr = 0;
p[0]=p[1]=1;
  for (i = 2; i <= n; ++i) {
    if (p[i] == 0) {
      nr++;
      for (j = i + i; j <= n; j += i) {
        p[j] = 1;
      }
    }
  }
  return nr;
}
```

Varianta 2 O prima idee de optimizare ar fi sa nu mai luam in calcul numerele pare pentru ca stim ca singurul numar prim par e 2. Deci sa vedem noua varianta a programului:

```
int MAXSIZE = 1000001;
int p[MAXSIZE];      //p[i] == 0 if i is prime
int getTheNumber(int n) {
  int i, j, nr = 1;
  for (i = 3; i <= n; i += 2) {
    if (p[i] == 0) {
      nr++;
      for (j = i + i + i; j <= n; j += i << 1) {
        p[j] = 1;
      }
    }
  }
  return nr;
}
```

Varianta 3 Putem incerca o optimizare de memorie, pentru ca nu mai avem nevoie de elementele cu index par din sirul p. Acum semnificatia lui p_i s-a schimbat p_i fiind 0 daca 2*i+1 e numar prim si 1 daca nu.

```
int MAXSIZE = 1000000/2+1;
int p[MAXSIZE];      //p[i] == 0 if 2*i + 1 is prime
 int getTheNumber(int n) {
  int i, j, nr = 1;
  for (i = 1; (i << 1) + 1 <= n; i += 1) {
    if (p[i] == 0) {
      nr++;
      for (j = i + i + i + 1; (j << 1) + 1 <= n; j += (i << 1) + 1) {
        p[j] = 1;
      }
    }
  }
  return nr;
}
```

Varianta 4 Urmatoarea optimizare va fi marcarea multiplilor numarului prim i de la i*i nu de la 2*i cum am facut in prima varianta sau de la 3*i cum am facut in a 2-a. Aceasta optimizare este evidenta: orice numar prim compus multiplu de i mai mic decat i*i are un factor prim mai mic decat i, si acel factor l-a marcat mai devreme, deci nu are rost sa il marcam si la pasul i.

```
int MAXSIZE = 1000000/2+1;
int p[MAXSIZE];    //p[i] == 0 if 2*i + 1 is prime
int getTheNumber(int n) {
  int i, j, nr = 1;
  for (i = 1; ((i * i) << 1) + (i << 1) <= n; i += 1) {
    if (p[i] == 0) {
      for (j = ((i * i) << 1) + (i << 1); (j << 1) + 1 <= n; j += (i << 1) + 1) {
        p[j] = 1;
      }
    }
  }
  for (i=1; 2 * i + 1 <= n; ++i)
      if (p[i] == 0) nr++;
  return nr;
}
```

Varianta 5 Codul sursa arata putin urat pentru ca nu lucram direct cu i ci cu 2*i+1, am mai facut o optimizare, nu parcurgem numerele pana la n pentru marcarea multiplilor ci pana la sqrt(n) lucru care e evident dupa cele explicate mai sus.Ultima imbunatatire care o vom aduce este aceea de a folosi mai putina memorie. Cum pentru fiecare numar e necesara doar o informatie booleana, aceasta o putem tine intr-un bit, nu este necesar un char intreg.

```
int MAXSIZE = 100000000/2/8+1;
int p[MAXSIZE];      //(p[i>>3]&(1<<(i&7))) == 0 if 2*i + 1 is prime
int getTheNumber(int n) {
  int i, j, nr = 1;
  for (i = 1; ((i * i) << 1) + (i << 1) <= n; i += 1) {
    if ((p[i >> 3] & (1 << (i & 7))) == 0) {
      for (j = ((i * i) << 1) + (i << 1); (j << 1) + 1 <= n; j += (i << 1) + 1)
           p[j >> 3] |= (1 << (j & 7));
    }
```

```
  }
  for (i = 1; 2 * i + 1 <= n; ++i)
       if ((p[i >> 3] & (1 << (i & 7))) == 0)
           nr++;
  return nr;
}
```

Codul p[i >> 3] & (1 << (i & 7)) == 0 testeaza daca al i-lea bit din sirul de biti e 0 (deci daca 2*i+1 e prim). i >> 3 e echivalent cu i / 8 deci se gaseste pt bitul al i-lea in ce char e el stocat din sirul nostru de charuri. i & 7 e echivalent cu i % 8 ca sa aflam pe ce bit al charului e stocat numarul prim i.
Codul p[j >> 3] |= (1 << (j & 7)) seteaza bitul j % 8 al charului j / 8 in 1, pentru ca sa stim ca numarul 2 * j + 1 nu e prim.