

Consultații pentru elevii de liceu organizate de Facultatea de Matematică și Informatică  
pentru pregătirea concursului Mate-Info  
și concursului de admitere 2021  
30.10.2021, 09:00-10:15  
Asist. Mihai Andrei  
Algoritmi care lucrează pe numere (fără tablouri sau alte elemente structurate)  
Partea I

**Problema 1** – calcularea celui mai mare divizor comun a 2 numere.

**1A. Enunț:**

Se citesc 2 numere naturale nenule, să se afișeze cel mai mare divizor comun al acestor numere.

**Exemple:**

Dacă se citește 12 și 18 se afișează "cmmdc(12, 18) = 6"

Dacă se citește 15 și 105 se afișează "cmmdc(15, 105) = 15"

Dacă se citește 22 și 115 se afișează "cmmdc(22, 115) = 1"

**Specificații:**

Avem nevoie de o funcție care să calculeze cel mai mare divizor comun.

Funcția *cmmdc(a, b)*:

*Descriere*: calculează cel mai mare divizor comun a 2 numere

*Date intrare*: a, b numere naturale nenule

*Date ieșire*: cel mai mare divizor comun al celor 2 numere

Rezolvare – Folosim *Algoritmul lui Euclid*

Algoritmul prin metoda **scăderilor repetate**:

Varianta C++:

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

int cmmdc(int a, int b){
    while(a != b){
        if(a > b)
            a = a - b;
        if(b > a)
            b = b - a;
    }
    return a;
}
```

```

int main(){
    int a,b;
    cin >> a;
    cin >> b;
    int d = cmmdc(a, b);
    cout << "cmmdc(a,b) = " << d;

}

```

Varianta Pascal:

```

program CMMDC;
var
a, b, d: integer;

function cmmdc(a, b:integer):integer;
begin
    while a <> b do
    begin
        if (a > b) then
            a := a - b;
        if (b > a) then
            b := b - a;
    end;
    cmmdc := a;
end;

begin
    readln(a);
    readln(b);
    d := cmmdc(a,b);
    writeln('cmmdc(a,b) = ',d);
end.

```

Algoritmul prin **metoda împărțirilor repetate**:

Varianta C++:

```

int cmmdc(int a, int b){
    int r;
    while(b != 0) {
        r = a % b;
        a = b;
        b = r;
    }
    return a;
}

```

Varianta Pascal:

```
function cmmdc(a, b:integer):integer;
var
  r:integer;
begin
  while b <> 0 do
  begin
    r := a mod b;
    a := b;
    b := r;
  end;
  cmmdc := a;
end;
```

Algoritmul prin **metoda împărțirilor repetate – varianta recursivă**:

Varianta C++:

```
int cmmdc(int a, int b){
{
    if(b == 0)
        return a;
    return cmmdc(b, a%b);
}
```

Varianta Pascal:

```
function cmmdc(a, b:integer):integer;
begin
  if (b = 0) then
    cmmdc := a
  else
    cmmdc := cmmdc(b, a mod b);
end;
```

**1B. Enunț:** să se calculeze cel mai mare divizor comun a **3** numere.

**Exemple:**

Dacă se citește 12 și 18 și 24 se afișează "cmmdc(12, 18, 24) = 6"

Dacă se citește 15 și 105 și 45 se afișează "cmmdc(15, 105, 45) = 15"

Dacă se citește 22 și 115 și 20 se afișează "cmmdc(22, 115, 20) = 1"

### Specificații:

Avem nevoie de o funcție care să calculeze cel mai mare divizor comun.

Funcția *cmmdc*(a, b, c):

*Descriere*: calculează cel mai mare divizor comun a 3 numere

*Date intrare*: a, b, c numere naturale nenule

*Date ieșire*: cel mai mare divizor comun al celor 3 numere

Rezolvare: ne folosim de *cmmdc* a 2 numere implementat anterior

Varianta C++:

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

int cmmdc(int a, int b, int c){
    int d1 = cmmdc(a,b);
    return cmmdc(d1, c);
}

int main(){
    int a,b;
    cin >> a;
    cin >> b;
    int d = cmmdc(a, b);
    cout << "cmmdc(a,b) = " << d;
}
```

Varianta Pascal:

```
program CMMDC3;
var
a, b: integer;

function cmmdc(a, b, c:integer):integer;
var
d1:integer;
begin
    d1 := cmmdc(a,b);
    cmmdc := cmmdc(d1, c);
end;

begin
    readln(a);
    readln(b);
    d := cmmdc(a,b);
    writeln('cmmdc(a,b) = ',d);
end.
```

## Problema 2 – Ce face algoritmul?

### 2A. Enunț:

Mai jos este prezentată funcția *calculează*. Precizați care este scopul algoritmului – ce calculează acesta.

Varianta C++:

```
int calculeaza(int n){
    int mc = 0;
    int f = 0;
    int x = n;
    while (x > 0){
        int c = x % 10;
        if (c == mc) f++;
        if (c > mc){
            mc = c;
            f = 1;
        }
        x = x/10;
    }
    return f;
}
```

Varianta Pascal:

```
function calculeaza(n:integer):integer;
var
mc,f,x,c:integer;
begin
    mc := 0;
    f := 0;
    x := n;
    while x > 0 do
        begin
            c := x mod 10;
            if (c = mc) then
                f := f + 1;
            if (c > mc) then
                begin
                    mc := c;
                    f := 1;
                end;
            x := x div 10;
        end;
    calculeaza := f;
end;
```

Răspuns: calculează frecvența de apariție a celei mai mari cifre dintr-un număr.

E.g.: calculează(125345) → 2 (5 apare de 2 ori); calculează(11444332) → 3 (4 apare de 3 ori)

## 2B. Enunț:

Mai jos este prezentată o bucată de cod. Precizați ce se întâmplă la rularea acestui cod

Varianta C++:

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

int main(){
    int a,b,c1,c2;
    do{
        cin>>a>>b;
        c1 = a; c2 = b;
        while (c1 != c2){
            if (c1 > c2) c1 = c1 - c2;
            if (c2 > c1) c2 = c2 - c1;
        }
    } while(c1 == 1);
    cout<<a*b;
}
```

Varianta Pascal:

```
program PRODUS;

var
a,b,c1,c2: integer;

begin
    repeat
        begin
            readln(a);
            readln(b);
            c1 := a; c2 := b;
            while c1 <> c2 do
                begin
                    if (c1 > c2) then c1 := c1 - c2;
                    if (c2 > c1) then c2 := c2 - c1;
                end;
            end
        until c1 <> 1;

        writeln(a*b);
    end.
```

Răspuns: citește numere de la tastatură până când citește 2 care **nu** sunt prime între ele și apoi afișează produsul lor. (e.g. dacă citește 18 și 35 citește din nou dacă citește 4 și 6 afișează 24)

## 2C. Enunț:

Mai jos este prezentată funcția *expresie* și funcția *f*. Precizați care este scopul algoritmului – ce calculează acesta.

Varianta C++:

```
double expresie(int n, int k, int p){
    p = k*p;
    if (n == k)
        return 1.0/p;
    return 1.0/p + expresie(n, k+1, p);
}

double f(int n){
    if (n < 1) return 0;
    return 1 + expresie(n, 1, 1);
}
```

Varianta Pascal:

```
function expresie(n,k,p:integer):double;
begin
    p := p*k;
    if (n = k) then
        expresie := 1/p
    else
        expresie := 1/p + expresie(n, k+1, p);
end;

function f(n:integer):double;
begin
    if (n < 1) then f := 0
    else f := 1 + expresie(n,1,1);
end;
```

Răspuns: acest algoritm calculează o suma:

$$f(n) = 1 + \frac{1}{1} + \frac{1}{2} + \frac{1}{6} + \dots + \frac{1}{n!} = 1 + \sum_{k=1}^n \frac{1}{k!}$$

Este nevoie de 2 funcții pentru că funcția *expresie* care calculează suma are nevoie de parametrii *k* și *p* care **trebuie** să înceapă de la 1 (altfel nu calculează corect). Astfel, avem nevoie de funcția *f*, care e funcția pe care o apelăm când vrem să calculăm această sumă iar *f* se folosește de funcția *expresie* pentru calcul și se asigură că *expresie* este apelată corect, cu parametrii *p* și *k* inițial fiind 1.

Ca și curiozitate:

$$\lim_{n \rightarrow \infty} \sum_{k=1}^n \frac{1}{k!} = e$$

### Problema 3 – Scrieți un algoritm.

#### 3A. Enunț:

Scrieți un algoritm care să calculeze suma:

$$f_2(n) = \sum_{k=1}^n \frac{1}{T_k}$$

unde  $T_k = 1 + 2 + \dots + k$

Rezolvare:

Pentru început trebuie decis cum calculăm  $T_k$  pentru fiecare termen al sumei. Putem calcula iterativ  $1 + 2 + \dots + k$  de fiecare dată, dar asta e ineficient. Pentru a calcula eficient  $T_k$  putem face 2 observații:

1. Se poate folosi suma lui Gauss pentru a calcula  $T_k$ , adică:  $T_k = 1 + 2 + \dots + k = \frac{k(k+1)}{2}$ ,
2. La fiecare termen al sumei  $T_k$  crește cu  $k$ . Adică:  $T_k = T_{k-1} + k$ .

Varianta C++:

Implementare recursivă folosind observația 1.

```
double f2(int n) {  
    if (n == 0)  
        return 0;  
    double Tk = n * (n + 1.0) / 2.0;  
    return 1.0 / Tk + f2(n - 1);  
}
```

Implementare iterativă folosind observația 1.

```
double f2(int n) {  
    double s = 0;  
    for (int k = 1; k <= n; k++) {  
        double Tk = k * (k + 1.0) / 2.0;  
        s = s + 1.0 / Tk;  
    }  
    return s;  
}
```

Implementare iterativă folosind observația 2.

```
double f2(int n) {  
    double s = 0;  
    int Tk = 0;  
    for (int k = 1; k <= n; k++) {  
        Tk = Tk + k;  
        s = s + 1.0 / Tk;  
    }  
    return s;  
}
```



Varianta Pascal:

Implementare recursivă folosind observația 1.

```
function f2(n:integer):double;  
var  
Tk:double;  
begin  
  if (n = 0) then f2 := 0  
  else  
    begin  
      Tk := n * (n + 1) / 2;  
      f2 := 1/Tk + f2(n-1);  
    end;  
  end;
```

Implementare iterativă folosind observația 1.

```
function f2(n:integer):double;  
var  
Tk, s:double;  
k:integer;  
begin  
  s := 0;  
  for k := 1 to n do  
    begin  
      Tk := k * (k + 1)/2;  
      s := s + 1/Tk;  
    end;  
  f2 := s;  
end;
```

Implementare iterativă folosind observația 2.

```
function f2(n:integer):double;  
var  
s:double;  
Tk, k:longint;  
begin  
  s := 0;  
  Tk := 0;  
  for k := 1 to n do  
    begin  
      Tk := Tk + k;  
      s := s + 1/Tk;  
    end;  
  f2 := s;  
end;
```

Ca și curiozitate:  $\lim_{n \rightarrow \infty} \sum_{k=1}^n \frac{1}{T_k} = 2$

### 3B. Enunț:

Scrieți un algoritm care să calculeze suma:

$$f_3(n) = \sum_{k=1}^n \frac{1}{3^k * k} + \sum_{k=1}^n \frac{1}{4^k * k}$$

Varianta C++:

```
double f3(int n){
    double s1 = 0.0;
    double s2 = 0.0;
    for (int k = 1; k <= n; k++){
        s1 = s1 + 1.0/ (pow(3.0,k) * k);
        s2 = s2 + 1.0/ (pow(4.0,k) * k);
    }
    return s1 + s2;
}
```

Varianta Pascal:

```
uses Math;

function f3(n:integer):double;
var
    s1,s2:double;
    k:longint;
begin
    s1 := 0;
    s2 := 0;
    for k := 1 to n do
        begin
            s1 := s1 + 1/(power(3,k)*k);
            s2 := s2 + 1/(power(4,k)*k);
        end;
    f3 := s1 + s2;
end;
```

Ca și curiozitate:

$$\lim_{n \rightarrow \infty} \sum_{k=1}^n \frac{1}{3^k * k} + \sum_{k=1}^n \frac{1}{4^k * k} = \ln 2 \ (\log_e 2)$$