

CURS 06 – FP

Algoritmi numerici fundamentali

3. Testarea primalității unui număr:

Un număr este *prim* dacă se divide doar cu 1 și el însuși.

Un număr este *prim* dacă nu are divizori proprii (diferiți de 1 și el însuși).

Un număr care nu este prim se numește *număr compus*.

```
#include <stdio.h>

int main()
{
    int n, d;

    printf("n = ");
    scanf("%d", &n);

    for(d = 2; d <= n/2; d++)
        if(n % d == 0)
            break;

    if(d == n/2 + 1)
        printf("Numarul %d este prim!", n);
    else
        printf("Numarul %d este compus!", n);

    return 0;
}
```

Variantă:

```
#include <stdio.h>
int main()
{
    int n, d, prim;

    printf("n = ");
    scanf("%d", &n);

    //presupunem ca numarul n este prim
    prim = 1;
    for(d = 2; d <= n/2; d++)
        if(n % d == 0)
```

```

    {
        //am gasit un divizor propriu al numarului n,
        //deci presupunerea devine falsa
        prim = 0;
        break;
    }

    if(prim == 1)
        printf("Numarul %d este prim!", n);
    else
        printf("Numarul %d este compus!", n);

    return 0;
}

```

Observație: Toți divizorii proprii ai unui număr natural n sunt cuprinși între 2 și $\left\lfloor \frac{n}{2} \right\rfloor$.

Exemplu: $n = 15 \Rightarrow \mathcal{D}_{15} = \{1, 3, 5, 15\} \Rightarrow$ toți divizorii proprii, adică 3 și 5, sunt mai mici sau egali decât $\lfloor 15/2 \rfloor = 7$.

Teoremă: Dacă n este compus, atunci el are cel puțin un divizor propriu mai mic sau egal decât \sqrt{n} .

Exemplu: $n = 15 \Rightarrow d_1 = 3 < \sqrt{15} \approx 3.87$, dar $d_2 = 5 > \sqrt{15}$!

Demonstrație:

Pp. prin absurd faptul că n este compus, dar nu are niciun divizor propriu mai mic sau egal decât $\sqrt{n} \Rightarrow n = d_1 * d_2$ și $d_1, d_2 > \sqrt{n} \Rightarrow n = d_1 * d_2 > \sqrt{n} * \sqrt{n} = n \Rightarrow n > n$ (imposibil!!!).

```
#include <stdio.h>
```

```

int main()
{
    int n, d;

    printf("n = ");
    scanf("%d", &n);

    //d * d <= n este echivalenta cu d <= sqrt(n),
    //dar mult mai rapida!!!
    for(d = 2; d * d <= n; d++)

```

```

        if(n % d == 0)
            break;

    if(d * d > n)
        printf("Numarul %d este prim!", n);
    else
        printf("Numarul %d este compus!", n);

    return 0;
}

```

Variantă optimizată (testăm doar posibili divizori impari):

```

#include <stdio.h>

int main()
{
    int n, d;

    printf("n = ");
    scanf("%d", &n);

    if(n < 2)
    {
        printf("Numarul %d nu este prim (primul numar prim este 2)!\n", n);
        return 0;
    }

    if(n == 2)
    {
        printf("Numarul 2 este prim!\n");
        return 0;
    }

    if(n % 2 == 0)
    {
        printf("Numarul %d este compus!\n", n);
        return 0;
    }

    //numarul n este impar si n >= 3, deci n
    //poate sa aiba doar divizori proprii impari
    for(d = 3; d * d <= n; d += 2)
        if(n % d == 0)
            break;

    if(d * d > n)
        printf("Numarul %d este prim!", n);
    else
        printf("Numarul %d este compus!", n);
}

```

```
    return 0;
}
```

4. Calculul cmmdc-ului și cmmmc-ului a două numere întregi

- **cmmdc(x, y)** = cel mai mare divizor comun al numerelor întregi x și y
- **cmmmc(x, y)** = cel mai mic multiplu comun al numerelor întregi x și y

Exemplu:

cmmdc(120, 18) = 6

cmmmc(120, 18) = 360

$$\left. \begin{array}{l} a = 120 = 2^3 * 3 * 5 \\ b = 18 = 2 * 3^2 \end{array} \right\} \Rightarrow \begin{cases} \text{cmmdc}(a, b) = 2 * 3 = 6 \\ \text{cmmmc}(a, b) = 2^3 * 3^2 * 5 = 360 \end{cases}$$

Descompunerea în factori primi a unui număr natural:

$$120 : 2 = 60$$

$$60 : 2 = 30$$

$$30 : 2 = 15$$

$$15 : 3 = 5$$

$$5 : 5 = 1$$

Pentru calculul cmmdc-ului există mai mulți algoritmi, cei mai cunoscuți fiind *algoritmul lui Nicomahus* (metoda scăderilor succesive) și *algoritmul lui Euclid*.

a) Algoritmul lui Nicomahus

$$\text{cmmdc}(a, b) = \begin{cases} \text{cmmdc}(a - b, b), & \text{dacă } a > b \\ \text{cmmdc}(a, b - a), & \text{dacă } a < b \\ a, & \text{dacă } a = b \end{cases}$$

Forma echivalentă: "Din numărul mai mare îl scad pe cel mai mic până cele două numere devin egale."

$$\begin{aligned}
\text{cmmdc}(120, 18) &= \text{cmmdc}(120 - 18, 18) = \text{cmmdc}(102, 18) \\
&= \text{cmmdc}(102 - 18, 18) = \text{cmmdc}(84, 18) \\
&= \text{cmmdc}(84 - 18, 18) = \text{cmmdc}(66, 18) \\
&= \text{cmmdc}(66 - 18, 18) = \text{cmmdc}(48, 18) \\
&= \text{cmmdc}(48 - 18, 18) = \text{cmmdc}(30, 18) \\
&= \text{cmmdc}(30 - 18, 18) = \text{cmmdc}(12, 18) \\
&= \text{cmmdc}(12, 18 - 12) = \text{cmmdc}(12, 6) \\
&= \text{cmmdc}(12 - 6, 6) = \text{cmmdc}(6, 6) = 6
\end{aligned}$$

Algoritmul efectuează multe operații de scădere când diferența dintre cele două numere este mare!!!

Observații generale referitoare la cmmdc:

- **cmmdc(x, 0) = x**
 $\text{cmmdc}(120, 0) = \text{cmmdc}(120 - 0, 0) = \text{cmmdc}(120, 0) = \dots = \text{ciclare infinită!}$
- **cmmdc(0, 0) = nu este definit**
- Algoritmii pentru calculul cmmdc-ului alterează valorile inițiale ale variabilelor a și b!

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int a, b, aux_a, aux_b;
```

```
    printf("a = ");
```

```
    scanf("%d", &a);
```

```
    printf("b = ");
```

```
    scanf("%d", &b);
```

```
    if(a == 0 && b == 0)
```

```
        printf("cmmdc(%d, %d) = indefinit\n", a, b);
```

```
    else
```

```
        if(a == 0)
```

```
            printf("cmmdc(%d, %d) = %d\n", a, b, b);
```

```
        else
```

```
            if(b == 0)
```

```
                printf("cmmdc(%d, %d) = %d\n", a, b, a);
```

```
            else
```

```
            {
```

```
                aux_a = a;
```

```

    aux_b = b;

    while(a != b)
        if(a > b)
            a = a - b;
        else
            b = b - a;

    printf("cmmdc(%d, %d) = %d\n", aux_a, aux_b, a);
}

return 0;
}

```

Observație:

$\text{cmmdc}(120, 18) = \text{cmmdc}(120 - 18, 18) = \text{cmmdc}(102, 18)$
 $= \text{cmmdc}(102 - 18, 18) = \text{cmmdc}(84, 18)$
 $= \text{cmmdc}(84 - 18, 18) = \text{cmmdc}(66, 18)$
 $= \text{cmmdc}(66 - 18, 18) = \text{cmmdc}(48, 18)$
 $= \text{cmmdc}(48 - 18, 18) = \text{cmmdc}(30, 18)$
 $= \text{cmmdc}(30 - 18, 18) = \text{cmmdc}(12, 18)$
 $= \text{cmmdc}(12, 18 - 12) = \text{cmmdc}(12, 6)$
 $= \text{cmmdc}(12 - 6, 6) = \text{cmmdc}(6, 6) = 6$

$120 : 18 = 6, \text{rest } 12$

b) Algoritmul lui Euclid

$a = 120, b = 18$

a	b	$a \% b$
120	18	12
18	12	6
12	6	0
6	0	?

$\text{cmmdc}(120, 18) = 6$

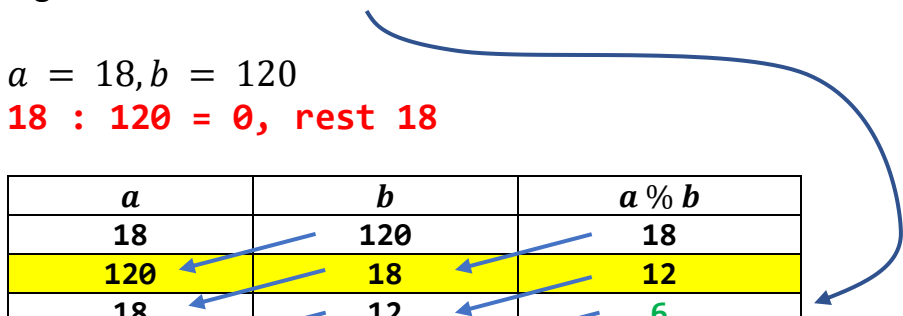
Forma echivalentă: "Înlocuiesc primul număr cu al doilea și al doilea număr cu restul împărțirii lor până când restul devine 0. Cmmddc-ul va fi ultimul rest nenul sau ultimul împărțitor".

Observație:

Algoritmul lui Euclid funcționează corect și dacă $a < b$, deoarece, la primul pas, algoritmul va interschimba a cu b !!!

$a = 18, b = 120$

18 : 120 = 0, rest 18



a	b	$a \% b$
18	120	18
120	18	12
18	12	6
12	6	0

```
#include <stdio.h>
int main()
{
    int a, b, aux_a, aux_b, r;

    printf("a = ");    scanf("%d", &a);
    printf("b = ");    scanf("%d", &b);

    if(a == 0 && b == 0)
        printf("cmmdc(%d, %d) = indefinit\n", a, b);
    else
        if(a == 0)
            printf("cmmdc(%d, %d) = %d\n", a, b, b);
        else
            if(b == 0)
                printf("cmmdc(%d, %d) = %d\n", a, b, a);
            else
            {
                aux_a = a;
                aux_b = b;

                r = a % b;
                while(r != 0)
                {
                    a = b;
                    b = r;
                    r = a % b;
                }

                printf("cmmdc(%d, %d) = %d\n", aux_a, aux_b, b);
            }

    return 0;
}
```

Pentru calculul cmmmc-ului se utilizează, de obicei, următoarea relație matematică:

$$\text{cmmdc}(a, b) * \text{cmmmc}(a, b) = a * b \Rightarrow \text{cmmmc}(a, b) = \frac{a * b}{\text{cmmdc}(a, b)}$$

Exemplu:

$a = 120, b = 18$

$\text{cmmdc}(120, 18) = 6 \Rightarrow \text{cmmmc}(120, 18) = 120 * 18 / 6 = 360$

$\text{cmmmc}(120, 18) = 360$

Observații generale referitoare la cmmmc:

- $\text{cmmmc}(x, 0) = \text{nu este definit}$
- $\text{cmmmc}(0, 0) = \text{nu este definit}$

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int a, b, aux_a, aux_b, r;
```

```
    printf("a = ");
```

```
    scanf("%d", &a);
```

```
    printf("b = ");
```

```
    scanf("%d", &b);
```

```
    if(a == 0 && b == 0)
```

```
    {
```

```
        printf("cmmdc(%d, %d) = indefinit\n", a, b);
```

```
        printf("cmmmc(%d, %d) = indefinit\n", a, b);
```

```
    }
```

```
    else
```

```
        if(a == 0)
```

```
        {
```

```
            printf("cmmdc(%d, %d) = %d\n", a, b, b);
```

```
            printf("cmmmc(%d, %d) = indefinit\n", a, b, b);
```

```
        }
```

```
    else
```

```
        if(b == 0)
```

```
        {
```

```
            printf("cmmdc(%d, %d) = %d\n", a, b, a);
```

```
            printf("cmmmc(%d, %d) = indefinit\n", a, b, b);
```

```
        }
```

```
    else
```



```

    {
        aux_a = a;
        aux_b = b;

        r = a % b;
        while(r != 0)
        {
            a = b;
            b = r;
            r = a % b;
        }

        printf("\ncmmdc(%d, %d) = %d\n", aux_a, aux_b, b);
        printf("\ncmmmc(%d, %d) = %d\n", aux_a, aux_b, aux_a
* aux_b / b);
    }

    return 0;
}

```

Calculul cmmdc-ului și cmmmc-ului utilizând forța brută:

```

#include <stdio.h>

int main()
{
    int a, b, min, max, cmmdc, cmmmc, d, m;

    //presupunem ca a si b sunt ambele diferite de 0

    printf("a = ");
    scanf("%d", &a);

    printf("b = ");
    scanf("%d", &b);

    // 1 <= cmmdc(a,b) <= min(a,b)
    min = a < b ? a : b;

    for(d = min; d >= 1; d--)
        if(a % d == 0 && b % d == 0)
        {
            cmmdc = d;
            break;
        }

    printf("\ncmmdc(%d, %d) = %d\n", a, b, cmmdc);

```

```

// max(a,b) <= cmmmc(a,b) <= a*b
max = a > b ? a : b;

for(m = max; m <= a*b; m++)
    if(m % a == 0 && m % b == 0)
    {
        cmmmc = m;
        break;
    }

printf("\ncmmmc(%d, %d) = %d\n", a, b, cmmmc);

return 0;
}

```

Observație: Dacă $\text{cmmdc}(a, b) = 1$ atunci spunem că *numerele a și b sunt prime între ele (coprime)!!!*