# ALGORITMI RECURSIVI

Recursivitatea a apărut din necesităţi practice, cum ar fi transcrierea directă a formulelor matematice recursive. Un exemplu de formulă matematică recursivă este următoarea:

$(n+1)! = (n+1) \cdot n!$, pentru $n \geq 0$

$0! = 1$ (prin convenţie)

Un subprogram este definit ca fiind *recursiv* dacă, pornind de la anumite valori ale ei, se pot calcula alte valori prin autoapelarea subprogramului. Apelul poate fi făcut *direct* sau *indirect* (prin intermediul altei funcţii).

Valorile unei funcţii recursive se calculează din aproape în aproape, pe baza valorilor cunoscute ale funcţiei pentru anumite argumente iniţiale. Pentru a calcula noile valori ale funcţiei recursive, trebuie memorate valorile deja calculate care sunt strict necesare. Acest fapt face ca implementarea în program a calculului unor funcţii recursive să necesite un volum mare de memorie.

Recursivitatea poate fi transformată în *iteraţie* (variantă nerecursivă). În general, forma nerecursivă a unui algoritm este de preferat formei recursive, fiind mai eficientă din punct de vedere al timpului de execuţie şi al memoriei ocupate. Forma recursivă este preferată datorită uşurinţei programării sau acolo unde transformarea recursivităţii în iteraţie cere un efort de programare deosebit, algoritmul pierzându-şi claritatea exprimării, testarea şi întreţinerea devenind astfel foarte dificile.

Exemplu de funcţie recursivă: funcţia $n!$ pe care o notăm cu *factorial*, ***factorial*:N→N** si care se defineşte recursiv astfel

$$factorial(n) = \begin{cases} 1, n = 0 \\ n \cdot factorial(n-1), n \geq 1 \end{cases}$$

Pentru a calcula *factorial*(4):

$$factorial(4) \overset{n=4}{=} 4 \cdot factorial(3) \overset{n=3}{=} 4 \cdot 3 \cdot factorial(2) \overset{n=2}{=} 4 \cdot 3 \cdot 2 \cdot factorial(1) \overset{n=1}{=} 4 \cdot 3 \cdot 2 \cdot 1 \cdot factorial(0) \overset{n=0}{=} 4 \cdot 3 \cdot 2 \cdot 1 \cdot 1 = 4!$$

De fiecare dată când o funcţie este apelată recursiv, pe *stiva sistemului* se reţin parametrii (de exemplu valoarea lui $n$ pentru determinarea factorialului) şi eventualele variabile locale ale funcţiei.

În cazul funcţiilor recursive se pune problema terminării calculelor. Apelul unei funcţii este terminat doar dacă cauzează un număr finit de apeluri la aceea funcţie sau la alte funcţii. În funcţiile recursive trebuie să se garanteze terminarea apelurilor recursive prin inserarea *condiţiilor de terminare*.

**Exemple de programe ce folosesc subprograme recursive:**

**R1.**

*Enunţul problemei*: Să se descrie un algoritm pentru determinarea sumei factorialelor până la $n \in \mathrm{N}^*$ dat, folosind o funcţie recursivă pentru calculul factorialului.

*Metoda de rezolvare*: Problema se reduce la calcul unei sume: $S = \sum_{i=1}^{n} i!$.

*Descrierea algoritmului în* C++ (folosind Codeblocks):

```
#include <iostream>
using namespace std;
int n,i;
float S;
float factorial(int n)
{
 if (n==0) return 1;
 //else
 return n*factorial(n-1);
}
int main() {
 cout<<"Dati valoarea lui n (n>=1): ";
 cin>>n; //* presupunem ca n>=1
 for (i=1;i<=n;i++) S += factorial(i);
 cout<<"Suma factorialelor pana la "<<n<<": "<<S<<endl;
 return 0;
}
```

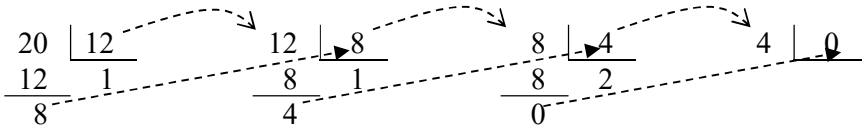
**R2.**

*Enunţul problemei*: Să se descrie un algoritm pentru determinarea celui mai mare divizor comun dintre *n* numere naturale nenule, folosind funcţie recursivă pentru determinarea cmmdc-ului.

*Metoda de rezolvare*: Determinarea cmmdc dintre mai multe numere se face iterativ prin determinarea cmmdc dintre primele două, apoi între rezultat şi următorul număr, ş.a.m.d. De exemplu, $cmmdc(x_1, x_2, x_3) = cmmdc(cmmdc(x_1, x_2), x_3)$. Astfel, cmmdc(2, 4, 6, 8) = 2.

Pe de altă parte, funcţia cmmdc poate avea următoarele două forme recursive:

1. folosind <u>împărţiri repetate</u>: $cmmdc(a,b) = cmmdc(b, a\%b)$, dacă $b \neq 0$; altfel este $a$.
   Pentru $a = 20$ şi $b = 12$ se proceda nerecursiv astfel:




2. folosind <u>scăderi repetate</u>: $cmmdc(a,b) = cmmdc(a-b, b)$, dacă $a>b$
   $cmmdc(a, b-a)$, dacă $b>a$
   $a$, $a=b$

   Pentru $a = 20$ şi $b = 12$ se proceda nerecursiv astfel:
   $a = 20 > b = 12 \Rightarrow a \leftarrow a-b \Rightarrow a = 20-12 = 8$
   $a = 8 < b = 12 \Rightarrow b \leftarrow b-a \Rightarrow b = 12-8 = 4$
   $a = 8 > b = 4 \Rightarrow a \leftarrow a-b \Rightarrow a = 8-4 = 6$
   $a = 4 = b = 4$. Stop. cmmdc(20,12) = 4.

*Descrierea algoritmului în pseudocod* C++ (folosind Codeblocks):

```
#include <iostream>
using namespace std;
int n,i,x[10],r;

int cmmdc1(int a,int b)
{  /*cu impartiri repetate*/
 if (b==0) return a;
 return cmmdc1(b,a%b);
}
```

```
int cmmdc2(int a,int b)
{     /*cu scaderi repetate*/
 if (a==b) return a;
 if (a>b) return cmmdc2(a-b,b);
 return cmmdc2(a,b-a);
}
int main()
{
 cout<<"Dati numarul de intregi pozitivi: ";
 cin>>n;
 cout<<"Dati cele n elemente intregi pozitivi: ";
 for (i=0;i<n;i++) cin>>x[i];
 r = cmmdc1(x[0],x[1]); //se det cmmdc dintre primele 2
 //sau r = cmmdc2(x[0],x[1]);
 for (i=2;i<n;i++)  //se iau in calcul si celelalte
   r = cmmdc1(r,x[i]);
 cout<<"Cmmdc="<<r<<endl;
 return 0;
}
```

**R3.**

*Enunţul problemei*: Descrieţi un algoritm recursiv pentru afişarea cifrelor unui număr întreg nenul.

*Metoda de rezolvare*: Ideea afişării cifrelor numărului *n*, de la cea a unităţilor până la prima este de a afişa ultima cifră a lui *n*, apoi de a repeta procedeul pentru numărul *n* din care se omite ultima cifră, ş.a.m.d.

*Descrierea algoritmului în* C++:

```
#include <iomanip>
#include <iostream>
using namespace std;
unsigned long n;
void cifre(unsigned long n) { //de la dreapta la stanga
 if (n) {  //sau n!=0
    cout<<setw(3)<<n%10; //se afiseaza ultima cifra
    cifre(n/10);  //apoi repeta pt numarul fara ultima cifra
  }
}
int main() {
 cout<<"n="; cin>>n;
 cout<<"Cifrele numarului (de la ultima la prima): ";
 cifre(n);
 return 0;
}
```

Pentru a afişa cifrele de la **stânga la dreapta** se schimbă funcţia C astfel (întâi se apelează procedeul similar pentru numărul n fără ultima cifră, apoi se afişează ultima cifră:

```
void cifre2(unsigned long n) {
 if (n) {  //n!=0
    cifre2(n/10);  //consideram numarul fara ultima cifra
    cout<<setw(3)<<n%10; //apoi se afişeaza ultima cifra
  }
}
```

cifre2(123) → cifre2 (12) → cifre2(1) → cifre2(0) ↘
afiş.3 ← afiş.2 ← afiş.1

**R4 (suplimentar).**

*Enunţul problemei*: Descrieţi un algortim pentru determinarea combinărilor de *n* luate câte *k*, folosind o funcţie recursivă.

*Metoda de rezolvare*: $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$, pentru $n>k$ şi $k>0$; altfel (pentru $n=k$ sau $k=0$) $C_n^0=1$, $C_n^n=1$. De exemplu: $C_4^2 = C_3^2 + C_3^1 = (C_2^2 + C_2^1) + (C_2^1 + C_2^0) = (1 + C_1^1 + C_1^0) + (C_1^1 + C_1^0 + 1) =$ $=(1+1+1) + (1+1+1) = 6$.

*Descrierea algoritmului în pseudocod*:

```
functie combinari(n,k)
daca n=k sau k=0 atunci
     returneaza 1
altfel
     returneaza combinari(n-1,k)+combinari(n-1,k-1)
sfarsit

citeşte n,k
scrie combinari(n,k)
```

*Descrierea algoritmului în* C++:

```
#include <iostream>
using namespace std;
int n,k;
int combinari(int n,int k)
{
 if (n==k || k==0) return 1;
 return combinari(n-1,k)+combinari(n-1,k-1);
}
int main() {
 cout<<"Dati valoarea lui n (n>=0): ";  cin>>n;
 cout<<"Dati valoarea lui k (k>=0, k<=n): "; cin>>k;
 cout<<"Combinari de "<<n<<" luate cate "<<k<<": "<<
          combinari(n,k)<<endl;
 return 0 ;
}
```

**R5.**

*Enunţul problemei*: Descrieţi un algoritm pentru determinarea valorii maxime dintre elementele unui vector, folosind o funcţie recursivă.

*Metoda de rezolvare*: $\text{maxim}\{x_1,\ldots,x_n\} = \text{maxim}\{\text{maxim}\{x_1,\ldots,x_{n-1}\}, x_n\}$.

*Descrierea algoritmului în C++* (folosind Codeblocks):

```
#include <iostream>
using namespace std;
#define maxim2(a,b) a>b ? a : b //maximul dintre 2 valori
int n,i,x[10];
int maxim(int x[10],int n)
{ /*dintre x[0]...x[n-1]*/
 if (n==2) return maxim2(x[0],x[1]);
 return maxim2(maxim(x,n-1),x[n-1]);
}
int main() {
 cout<<"n = "; cin>>n;
 cout<<"Dati valorile: ";
 for (i=0;i<n;i++) cin>>x[i];
 cout<<"Elementul maxim este: "<<maxim(x,n);
 return 0; }
```

**Temă (suplimentară)**:

1) (2*) Descrieţi un algoritm recursiv pentru determinarea numărului lui Fibonacci de ordin $n$ (şirul lui Fibonacci este definit astfel: $F_n = F_{n-1} + F_{n-2}$, pentru $n \geq 2$, iar $F_0 = F_1 = 1$; de exemplu $F_2 = 1+1 = 2$, $F_3 = F_2 + F_1 = 2+1=3$, $F_4 = F_3 + F_2 = 3+2=5$).
2) (3*) Descrieţi un algoritm recursiv pentru afişarea inversă a elementelor unui vector – folosind o funcţie recursivă (afişez ultimul elem, apoi apelez funcţia void pentru restul vectorului).
3) (3*) Folosind funcţii recursive, descrieţi algoritmul pentru afişarea elementelor unui vector – folosind o funcţie recursivă (apelez funcţia pentru restul vectorului, apoi afişez ultimul element).
4) (4*) Determinaţi o funcţie recursivă pentru calculul sumei $S_n = 1 + 3 + 5 + \ldots + (2n-1)$, $n \geq 1$ şi descrieţi un algoritm pentru calculul acestei sume.
5) (5*) Folosind recursivitatea, decrieţi un algoritm pentru a verifica dacă un vector conţine cel puţin un număr negativ în primele $n$ poziţii.
6) (4*) Descrieţi un algoritm recursiv pentru conversia unui număr din baza 10 în baza 2.