



Programarea Calculatoarelor

Cursul 8: Recursivitate

Ion Giosan

Universitatea Tehnică din Cluj-Napoca

Departamentul Calculatoare



- 2



- 3



- 4



- 5



- $$fact(n) = \begin{cases} 1 & , \text{dacă } n = 0 \\ n \times fact(n-1) & , \text{dacă } n > 0 \end{cases}$$

- 6



7



n=3

$$3! = 6$$



- a. Imediat după intrarea în apelul lui factorial(3)
- b. Imediat după intrarea în apelul recursiv al lui factorial(2)
- c. Imediat după intrarea în apelul recursiv al lui factorial(1)
- d. Imediat după intrarea în apelul recursiv al lui factorial(0)
- e. Imediat după ieșirea din apelul recursiv (pentru $n=0$)
- f. Imediat după ieșirea din apelul recursiv (pentru $n=1$)
- g. Imediat după ieșirea din apelul recursiv (pentru $n=2$)
- h. Imediat după ieșirea din apelul recursiv (pentru $n=3$) în funcția main()





Exemplu – Inversarea unui șir de întregi

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

void inverseaza(int *a, int stg, int dr)
{
    if (stg<dr)
    {
        int t=a[stg];
        a[stg]=a[dr];
        a[dr]=t;
        inverseaza(a, stg+1, dr-1);
    }
}

int main()
{
    int n=5;
    int a[]={3,6,4,1,2};
    inverseaza(a,0,n-1);
    for (int i=0; i<n; i++)
        printf("%d ",a[i]); // 2 1 4 6 3
    return 0;
}
```



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

int minim(int *a, int stg, int dr) {
    if (stg==dr)
        return a[stg];
    else
    {
        int m = minim(a, stg+1, dr);
        if (a[stg]<m)
            return a[stg];
        else
            return m;
    }
}

int main() {
    int n=5;
    int a[]={3,6,4,1,2};
    printf("%d",minim(a,0,n-1)); // 1
    return 0;
}
```



12



- $$fib(n) = \begin{cases} 0, & \text{dacă } n = 0 \\ 1, & \text{dacă } n = 1 \\ fib(n-1) + fib(n-2), & \text{dacă } n \geq 2 \end{cases}$$

- 13



27 septembrie 2020



Exemplu – Partițiile unui număr natural pozitiv

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

void partitii(int i, int *x, int n) {
    for (int j=1; j<=n; j++) {
        x[i]=j;
        if (j<n)
            partitii(i+1, x, n-j);
        else {
            for (int k=0; k<=i; k++)
                printf("%d ", x[k]);
            printf("\n");
        }
    }
}

int main() {
    int n=4;
    int x[n];
    partitii(0, x, n);
    return 0;
}
```



- ```
void afisare_cifre(int n) {
 afisare_cifre(n / 10);
 printf("%d\n", (n % 10));
}
```

- ```
void afisare_cifre(int n) {
    if (n < 10)
        printf("%d\n", n);
    else
    {
        afisare_cifre(n / 10);
        printf("%d\n", (n % 10));
    }
}
```




Depășirea stivei !

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

void f(int n) {
    double x[10000]; // Tablou alocat pe stiva la fiecare apel
    if (n==0)
        return;
    else
    {
        f(n-1);
        printf("%d ", n);
    }
}

int main() {
    f(10); // 1 2 3 4 5 6 7 8 9 10
    f(100); // Depasire de stiva !!!
    return 0;
}
```



Recursivitatea – important !

- Pentru a evita ciclul infinit scrieți un caz special în care funcția să nu se apeleze recursiv
- Atenție la funcțiile care se pot apela prin recursivitate indirectă
- Recursivitatea trebuie gândită bine – altfel se pot scrie programe total ineficiente
- Recursivitatea este o alternativă pentru structurile repetitive *for* si *while* utilizate în calcule ce necesită iterații multiple
- Funcțiile recursive sunt foarte utile când structurile de date (ex. arbori binari de căutare) sau paradigma de programare (ex. *divide et impera*) sunt recursive prin definiție



- 19