



Programarea Calculatoarelor

Cursul 8: Recursivitate

Ion Giosan

Universitatea Tehnică din Cluj-Napoca

Departamentul Calculatoare



- 2



- 3



Terminarea recursivității

- O funcție recursivă trebuie să aibă în construcția ei o **condiție de terminare**
 - Altfel apelul recursiv poate să ducă la o buclă infinită (asemănătoare structurilor repetitive în care condiția de continuare este întotdeauna adevărată și care iterează la nesfârșit)!
- **Adâncimea recursivității** trebuie să nu fie una foarte mare
 - La fiecare apel recursiv se memorează pe stivă
 - Parametrii funcției
 - Variabilele locale automate
 - Adresa de revenire din apelul funcției
 - Aceasta variază în funcție de numărul de octeți conținuți în parametri și variabilele locale automate
 - În cazul depășirii dimensiunii maxime a stivei, programul se termină subit în urma unei erori – ***stack overflow***



- 5



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

- 6



Exemplu – Calculul lui $n!$

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

double factorial(int n){
    printf("Apel recursiv cu n=%d\n", n);
    double y;
    if (n<=0)
        y = 1;
    else
        y = factorial(n-1) * n; /* &2=adresa de intoarcere dupa apelul
                                lui factorial(n-1) */
    printf("Iesirea din apelul recursiv cu n=%d\n", n);
    return y;
}

int main() {
    int n;
    printf("n=");
    scanf("%d", &n);
    printf("%d!=%g\n", n, factorial(n));
    /* &1=adresa de intoarcere dupa evaluarea lui factorial(n) */
    return 0;
}
```



n=3

Apel recursiv cu n=2

Apel recursiv cu $n=0$

Iesirea din apelul recursiv cu $n=1$

Iesirea din apelul recursiv cu $n=3$

$$3! = 6$$



- 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 iesirea din apelul recursiv (pentru $n=0$)
- f. Imediat după iesirea din apelul recursiv (pentru $n=1$)
- g. Imediat după iesirea din apelul recursiv (pentru $n=2$)
- h. Imediat după iesirea 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;
}
```



Exemplu – Valoarea minimă dintr-un șir de întregi

```
#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;
}
```



24 septembrie 2024



Recursivitate neliniară

- Funcția se auto-apelează de mai multe ori de-a lungul firului execuției ei
- Exemple
 - Șirul lui Fibonacci

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

- Partițiile unui număr natural pozitiv
 - Exemplu pentru numărul 4:
 $\{1, 1, 1, 1\}, \{1, 1, 2\}, \{1, 2, 1\}, \{1, 3\}, \{2, 1, 1\}, \{2, 2\}, \{3, 1\}, \{4\}$



Exemplu – Șirul lui Fibonacci

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

int fib(int n)
{
    if (n==0)
        return 0;
    if (n==1)
        return 1;
    return fib(n-1)+fib(n-2);
}

int main()
{
    printf("%d\n", fib(20)); //6765
    printf("%d\n", fib(42)); //Timp de executie ridicat!!!
    return 0;
}
```



```
#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;
}
```



Bucă infinită !

- Nu există condiție de terminare:

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

- Există condiție de terminare, dar apelul recursiv nu modifică valoarea parametrului:

```
void afisare_cifre(int n) {  
    if (n < 10)  
        printf("%d\n", n);  
    else  
    {  
        afisare_cifre(n);  
        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(10); // Depasire de stiva !!!
    return 0;
}
```



- 18



- 19