# ***Laborator 9***

Un obiect sau un fenomen se defineste in mod recursiv daca in definitia sa exista o referire la el insusi. Utilitatea practica a recursivitatii: posibilitatea de a defini un set infinit de obiecte printr-o singură relatie sau printr-un set finit de relatii.

Recursivitatea s-a impus în programarea calculatoarelor odata cu aparitia unor limbaje de nivel înalt, care permit scrierea de subprograme ce se autoapelează:

Un exemplu ar fi definirea conceptului de stramos al unei persoane:
Un parinte este stramosul copilului. **("Baza"')**
Parintii unui stramos sunt si ei stramosi **("Pasul de recursie").**

O gandire recursiva exprima concentrat o anumita stare, care se repeta la infinit.

**Iterativitatea** inseamna executia repetata a unei portiuni de program, pana la indeplinirea unei conditii folosind structurilor de control repetitive, prin instructiuni ca: ***while, do while, for*** .

**Recursivitatea** inseamna:

- executia repetata a unui intreg subprogram, functie sau metoda
- in cursul executiei lui se verifica o conditie (***if*** din C/C++)
- nesatisfacerea conditiei implica reluarea executiei subprogramului de la inceput, fara ca executia curenta a acestuia sa se fi terminat
- in momentul satisfacerii conditiei se revine in ordine inversa in lantul de apeluri, reluandu-se si încheindu-se apelurile suspendate

**O functie se numeste recursiva daca ea se autoapeleaza**.

Dupa tipul apelului, o functie recursiva se autoapeleaza:

- fie direct (in definitia ei, se face apel la ea insasi),
- fie indirect (adica functia X apeleaza functia Y, care apeleaza functia X). Orice functie recursiva poate fi scrisa si in forma nerecursiva (folosind structurile de control repetitive).

1.Scrie o funcție recursivă **factorial(int n)** care calculează factorialul unui număr natural n citit de la tastatură.Folosește ca bază de recursie faptul că ***factorial(0) = 1,*** iar în rest ***factorial(n) = n * factorial(n - 1).***

```c
#include <stdio.h>

int factorial(int n) {
    if (n == 0) return 1; // caz de bază
    return n * factorial(n - 1); // pas recursiv
}

int main() {
    int n;
    printf("Introdu un numar: ");
    scanf("%d", &n);
    printf("Factorialul lui %d este %d\n", n, factorial(n));
    return 0;
}
```

2.Scrie o funcție recursivă suma(int n) care calculează suma numerelor de la 1 la n.
Fără a folosi bucle, implementează logica în care suma(1) = 1, iar pentru n > 1 avem **suma(n) = n + suma(n - 1).**

```c
#include <stdio.h>

int suma(int n) {
    if (n == 1) return 1; // caz de bază
    return n + suma(n - 1); // pas recursiv
}

int main() {
    int n;
    printf("Introdu un numar: ");
    scanf("%d", &n);
    printf("Suma primelor %d numere naturale este %d\n", n, suma(n));
    return 0;
}
```

3. Implementează o funcție recursivă este***_palindrom(char s[], int st, int dr***) care verifică dacă un șir de caractere este palindrom (se citește la fel de la stânga la dreapta și invers).Funcția va compara caracterele extreme și se va autoapela pentru porțiunea interioară a șirului **(st+1, dr-1)** până când se ajunge la baza recursiei.

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

int este_palindrom(char s[], int st, int dr) {
    if (st >= dr) return 1; // caz de bază
    if (s[st] != s[dr]) return 0;
    return este_palindrom(s, st + 1, dr - 1);
}

int main() {
    char s[100];
    printf("Introdu un sir: ");
    scanf("%s", s);
    if (este_palindrom(s, 0, strlen(s) - 1))
        printf("Sirul este palindrom.\n");
    else
        printf("Sirul NU este palindrom.\n");
    return 0;
}
```

4. Creează o funcție recursivă ***suma_cifre(int n)*** care calculează suma tuturor cifrelor unui număr natural n.

```c
#include <stdio.h>

int suma_cifre(int n) {
    if (n == 0) return 0; // caz de bază
    return n % 10 + suma_cifre(n / 10); // pas recursiv
}

int main() {
    int n;
    printf("Introdu un numar: ");
    scanf("%d", &n);
    printf("Suma cifrelor este: %d\n", suma_cifre(n));
    return 0;
}
```

5. Construiește o funcție recursivă ***combinari(int n, int k, int start, int comb[], int index***) care generează toate combinațiile posibile de k elemente distincte dintr-un șir de n numere naturale (de la 1 la n).La fiecare pas, funcția selectează următorul element posibil și continuă construcția combinației până când dimensiunea este atinsă (index == k), moment în care se afișează combinația.

```c
#include <stdio.h>

void afisare(int comb[], int k) {
    for (int i = 0; i < k; i++)
        printf("%d ", comb[i]);
    printf("\n");
}

void combinari(int n, int k, int start, int comb[], int index) {
    if (index == k) {
        afisare(comb, k);
        return;
    }

    for (int i = start; i <= n; i++) {
        comb[index] = i;
        combinari(n, k, i + 1, comb, index + 1);
    }
}

int main() {
    int n, k;
    printf("Introdu n si k: ");
    scanf("%d %d", &n, &k);
    int comb[100];
    combinari(n, k, 1, comb, 0);
    return 0;
}
```

Tema

- Scrie o funcție recursivă care primește un număr natural n și îl returnează cu cifrele în ordine inversă.Funcția trebuie să extragă ultima cifră a numărului (n % 10) și să construiască recursiv restul.Se poate transmite un parametru suplimentar pentru a construi rezultatul (ex: int invers(int n, int acc)).
- Scrie o funcție recursivă care calculează cel mai mare divizor comun (CMMD) a două numere naturale a și b, folosind algoritmul lui Euclid. Folosește relația:
  - cmmdc(a, b) = cmmdc(b, a % b)
  - Baza recursiei este când b == 0, caz în care cmmdc(a, 0) = a.