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.Calculul lui n!

```cpp
//Calculul lui n!

#include <iostream>

using namespace std;

//metoda iterativa (clasica)
int fact_iterativ(int n) {
    int fact = 1;
    for(int i = 1; i <= n; i++) {
        fact *= i;
    }
    return fact;
}

//metoda recursiva
int fact_recursiv(int n)
{
   if(n==0)
      return 1;
   else
      return n*fact_recursiv(n-1);
}

int main()
{

    int n;

    cout<<"Dati N: ";
    cin >> n;

    cout<<"N! cu metoda iterativa este egal cu: "<< fact_iterativ(n)<<endl;
    cout<<"N! cu metoda recursiva este egal cu: "<< fact_recursiv(n);

    return 0;
}
```

2.Recursivitate indirecta

```
#include <iostream>
using namespace std;
int fa(int);
int fb(int);
int fa(int n){
    if(n<=1)
        return 1;
    else
        return n*fb(n-1);
}
int fb(int n){
    if(n<=1)
        return 1;
    else
        return n*fa(n-1);
}
int main(){
    int num=5;
    cout<<fa(num);
    return 0;
}
```

3.Recursivitate indirecta

```
//Recursivitate indirecta
#include <iostream>

using namespace std;

int i=0;

// declarare functii recursive
void fool(int);
void foo2(int);

void fool(int n)
{
  if (i <= n)
  {
    cout<<i<<" ";
    i++;
    foo2(n);
  }
  else
    return;
}

void foo2(int n)
{
  if (i <= n)
  {
    cout<<i<<" ";
    i++;
    fool(n);
  }
  else
    return;
}

int main(void)
{
  int n;
  cin >> n;
  fool(n);

  return 0;
}
```

4. Șirul Fibonacci recurisv

```cpp
#include <iostream>
using namespace std;
// Functie recursivă pentru calculul numerelor Fibonacci
int fibonacci(int n) {
    if (n <= 1) {
        return n; // Cazul de bază: fibonacci(0) = 0, fibonacci(1) = 1
    } else {
        // Pasul recursiv: fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}
int main() {
    int num;

    cout << "Introduceti pozitia dorita in sirul Fibonacci: ";
    cin >> num;

    // Afiseaza intregul sir Fibonacci pana la pozitia num
    cout << "Sirul Fibonacci pana la pozitia " << num << " este: ";
    for (int i = 0; i <= num; ++i) {
        cout << fibonacci(i) << " ";
    }
    return 0;
}
```