Șiruri recursive

• ***Șirurile Recursive***

- Un șir se numește recursiv dacă se autoapelează adică, dacă în corpul său există un modul care se autoapelează.
- Recursivitatea se realizează cu ajutorul subprogramelor.
- Un subprogram care se autoapelează se numește subprogram recursiv. Prin urmare un algoritm se numește recursiv sau un program se numește recursiv dacă conține cel puțin un subprogram care se autoapelează.
- Prin urmare orice subprogram recursiv trebuie să îndeplinească următoarele condiții:

  – să se poată executa cel puțin o dată fără a se autoapela (când apare condiția de oprire);
  – toate autoapelurile să se producă astfel încât la un moment dat să se ajungă la îndeplinirea condiției de oprire.

Cum se realizează recursivitatea?

- Recursivitatea se realizează în felul următor:

– Din afara subprogramului facem un apel al subprogramului;
– Dacă nu este îndeplinită o condiție (numită condiție de oprire), algoritmul se autoapelează de un anumit număr de ori (până când e îndeplinită condiția). La fiecare nouă autoapelare a subprogramului se reexecută secvența de instrucțiuni din corpul său, eventual cu alte date, creându-se un lanț de autoapeluri recursive.

Observatie! - Dacă condiția de oprire nu este îndeplinită niciodată sau nu există, algoritmul va avea un rezultat asemănător intrării într-un ciclu infinit.

• ***Șirul lui Fibonacci***

Ce este șirul lui Fibonacci?

- Șirul lui Fibonacci este o secvență infinită de numere în care fiecare număr este suma celor două numere anterioare. Forma sa standard este: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... Și continuă la infinit. Adică, fiecare număr din șir (începând cu al treilea număr) este suma celor două numere precedente.

Observații!

- Intuitiv, putem spune că un algoritm recursiv are același efect ca și o buclă (ciclu, instrucțiune repetitivă) – repetă execuția unui set de instrucțiuni. În mod analog, deducem că repetarea nu trebuie să se realizeze de un număr infinit de ori. De aici provine necesitatea existenței condiției de oprire.
- Cea mai mare parte a algoritmilor repetitivi se pot implementa într-o variantă nerecursivă (numită variantă iterativă) folosind instrucțiuni recursive, cât și într-o variantă recursivă atunci când conțin un modul definit recursiv.
- Varianta de implementare rămâne la latitudinea programatorului.
- Varianta recursivă este recomandată în cazul în care problemele sunt definite printr-o relație de recurență însă acest tip de algoritmi sunt mai greu de urmărit și de obicei necesită un timp de execuție mai mare.
- Execuția subprogramelor recursive seamănă cu execuția instrucțiunilor repetitive. Diferența constă în faptul că autoexecuția, în cazul programelor recursive, nu se poate realiza de un număr infinit de ori deoarece la fiecare autoapel se păstrează în stiva calculatorului variabilele locale și parametrii trimiși prin valoare iar stiva are o dimensiune limitată.
- Stiva reprezintă o succesiune ordonată de elemente, delimitată de două capete, în care adăugarea și eliminarea elementelor se poate face pe la un singur capăt, numit vârful stivei. Extragerea elementelor se realizează în ordine inversă introducerii lor.

Probleme:

1.

#255

Cerință:

Se dă un număr natural n. Să se afişeze în ordine crescătoare, primii n termeni ai şirului lui Fibonacci. Programul citește de la tastatură numărul n. Programul afișează pe ecran primii n termeni ai şirului lui Fibonacci, în ordine crescătoare, separați printr-un spaţiu.

Restricții: 1 =< n =< 40;

Exemplu: 5, se afișează: 1 1 2 3 4 5.

```
#include <iostream>
using namespace std;
int main()
{
```

```
    int n, a=1, b=1, c;
    cin>>n;
    if(n==1)
        cout<<1;
    else{
        cout<<1<<" "<<1<<" ";
        for(int i=3; i<=n; i++){
          c=a+b;
          cout<<c<<" ";
          a=b;
          b=c;
        }}
    return 0;
}
```

2.

Cerință:

Să se afișeze toți termenii șirului lui Fibonacci mai mici decât un număr natural n introdus de la tastatură.

```
#include <iostream>
using namespace std;
int main()
{ int n, a=1, b=1, c;
    cin>>n;
    cout<<a<<" ";
    while(b<n)
    {
        cout<<b<<" ";
        c=a+b;
        a=b;
        b=c;}
return 0;}
```