

1. Noțiunea de subprogram

Prin **subprogram** se înțelege un ansamblu alcătuit din tipuri de date, variabile și instrucțiuni scrise în vederea unei anumite prelucrări (calcul, citiri, scrieri) și care poate fi utilizat (rulat) doar dacă este apelat de un program sau de un alt subprogram.

Subprogramul este cea mai mică unitate de program care poate fi compilată separat.

Un **subprogram** se mai numește și **funcție**.

În general, o problemă complexă se rezolvă mai ușor dacă o descompunem în altele mai simple; șansele de a greși la scrierea unui subprogram sunt mai mici decât acelea de a greși la scrierea unui program mare. Acesta din urmă rezultă din asamblarea subprogramelor la care se adaugă, eventual, câteva linii scrise în programul principal. Avantajele utilizării subprogramelor sunt:

- reutilizarea codului : o dată scris, un subprogram poate fi utilizat de mai multe programe
- elaborarea algoritmilor prin descompunerea problemei în altele mai simple; în acest fel putem rezolva mai ușor problema, eventual prin colaborarea unei echipe de programatori
- reducerea numărului de erori care pot apărea la scrierea programelor
- depistarea cu ușurință a erorilor : verificăm subprogramele, apoi modul în care le-am asamblat

2. Structura subprogramelor și apelul lor

2.1 Definitia unui subprogram

În esență, un subprogram are structura:

antet

instrucțiune compusă

a) Antetul conține mai multe informații importante necesare compilatorului și anume: *numele funcției, lista parametrilor formali, tipul rezultatului.*

Structura antetului este:

tip nume(lista parametrilor formali)

Lista parametrilor formali este de forma:

parametru₁,parametru₂,.....,parametru_n

Parametrii formali sunt entități simbolice folosite pentru a descrie prelucrările realizate de funcție. Fiecare *parametru* are forma:

tip nume

Observație: Există și posibilitatea ca lista parametrilor formali să fie vidă, oricum prezența parantezelor rotunde în antet este obligatorie.

b) Instrucțiunea compusă cuprinde declarațiile variabilelor locale și instrucțiunile propriu-zise care descriu operațiile realizate de către funcție.

Observații:

- poate fi tip al unei funcții orice tip de dată cu excepția tablourilor
- dacă ținem neapărat, există posibilitatea ca funcția să întoarcă tablouri, dacă acestea sunt înglobate în tipuri declarate cu **struct**, așa cum se va vedea într-un paragraf ulterior.

Exemple de antete de funcții:

- **int suma(int a,int b)** – funcția se numește **suma**, returnează un rezultat de tip **int** și are doi parametri formali de tip **int**, numiți **a** și **b**
- **void t(int n,float v[20])** – funcția se numește **t**, este de tip **void** (nu returnează rezultat prin nume), are doi parametri formali, primul numit **n**, de tip **int**, al doilea numit **v**, de tip **vector**

- **ELEV mana(int n,char a[200])** – funcție de tip **ELEV**, unde **ELEV** este de tipul structură următor:

```
struct ELEV{  
    char nume[20];  
    int varsta;  
};
```

Încheierea unei funcții se poate face în mai multe moduri, după tipul rezultatului returnat de funcție.

În cazul funcțiilor de tip **void** (care nu returnează rezultat) încheierea funcțiilor se poate face în două moduri și anume:

- la încheierea instrucțiunii compuse care formează corpul funcției (încheiere normală)

Exemplu: funcția următoare afișează elementele vectorului **v** cu **n** elemente întregi

```
void afisare1(int n,int v[100])  
{  
    int i;  
    for(i=1;i<=n;i++)  
        cout<<v[i]<<" ";  
}
```

Evident, această funcție se încheie în momentul în care se întâlnește acolada închisă care încheie corpul funcției.

- la întâlnirea instrucțiunii **return ;** (încheiere forțată)

Exemplu: funcția următoare afișează doar primele elemente impare dintr-un vector **v** cu **n** elemente întregi. La întâlnirea primei valori pare, execuția funcției se încheie și se revine în modulul apelator (care a făcut apelul).

```
void afisare2(int n,int v[100])  
{  
    int i;  
    for(i=1;i<=n;i++)  
        if(v[i]%2)  
            cout<<v[i]<<" ";  
    else
```

```

        return ;
    cout<<"\n";
}

```

Dacă vectorul **v=(1,-9,13,19,5,-23)**, funcția va afișa toate valorile și se va încheia normal.

Dacă vectorul **v=(9,-17,21,6,11,24,9)**, funcția va afișa valorile **9,-17,21** și se va încheia forțat cu **return** pentru **v[4]=6**.

În cazul funcțiilor care returnează un rezultat, indiferent de natura acestuia, corpul funcției trebuie să conțină cel puțin o instrucțiune de forma **return expresie**; prin care se returnează rezultatul calculat de funcție, și în care **expresie** trebuie să fie de același tip cu tipul funcției sau să poată fi convertit implicit către acesta.

Exemplu: următoarea funcție returnează suma cifrelor unui număr întreg n

```

int suma(int n)
{
    int s=0;
    while(n)
    {
        s=s+n%10;
        n=n/10;
    }
    return s;
}

```

Exemplu: următoarea funcție verifică dacă numărul întreg x este prim sau nu returnând 1 în caz afirmativ și 0 în caz contrar.

```

int Prim(int x)
{
    int d;
    if(x==2) return 1;
    if(!(x%2)) return 0;
    for(d=3;d<=sqrt(x);d+=2)
        if(!(x%d)) return 0;
    return 1;
}

```

Evident, funcția folosește algoritmul cunoscut de verificare a primalității unui număr întreg, având patru instrucțiuni **return** care tratează toate cazurile posibile: **x=2**, **x** par diferit de 2, **x** impar compus, **x** impar prim.

2.2. Apelul unui subprogram

Se poate face din **main()** sau dintr-o altă funcție. Apelul se poate face de sine stător sau din interiorul unei expresii.

//apelul unei functii cu rezultat returnat

```
#include <iostream>
```

```
using namespace std;
```

```
int suma(int n)
```

```
{
```

```
    int s=0;
```

```
    while(n)
```

```
    {
```

```
        s=s+n%10;
```

```
        n=n/10;
```

```
    }
```

```
    return s;
```

```
}
```

```
int main()
```

```
{
```

```
    int n;
```

```
    cin>>n;
```

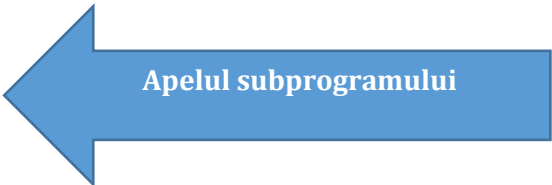
```
    cout<<suma(n);
```

```
    return 0;
```

```
}
```



Definitia subprogramului



Apelul subprogramului

//apelul unei functii cu rezultat de tip void

```
#include <iostream>
```

```
using namespace std;
```

```
void afisare1(int n,int v[100])
```

```
{
```

```
    int i;
```

```
    for(i=1;i<=n;i++)
```

```
        cout<<v[i]<<" ";
```

```
    }
```

```
int main()
```

```
{
```

```
    int n,v[100],i;
```

```
    cin>>n;
```

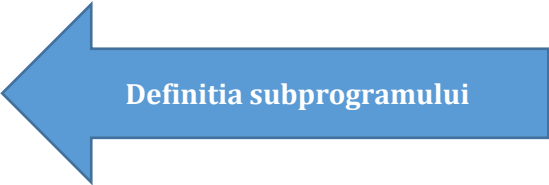
```
    for(i=1;i<=n;i++)
```

```
        cin>>v[i];
```

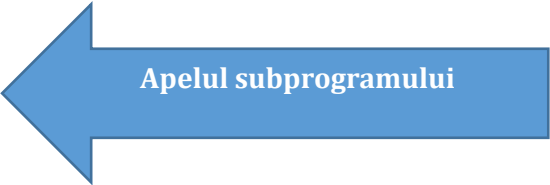
```
    afisare1(n,v);
```

```
    return 0;
```

```
}
```



Definitia subprogramului



Apelul subprogramului