Identificarea elementelor unui subprogram

Un program scris în limbajul C/C++ este un ansamblu de funcţii, fiecare dintre acestea efectuând o activitate bine definită.

Funcţiile comunică prin argumente: ele primesc ca parametri (argumente) datele de intrare, efectuează prelucrările descrise în corpul funcţiei asupra acestora şi pot returna o valoare (rezultatul, datele de ieşire). Execuţia programului începe cu funcţia principală, numită **main**. Funcţiile pot fi descrise în cadrul aceluiaşi fişier, sau în fişiere diferite, care sunt testate şi compilate separat, asamblarea lor realizându-se cu ajutorul link-editorului de legături.

În limbajul C++ există trei elemente implicate în utilizarea unui subprogram:

- **prototipul subprogramului;**
- **apelul subprogramului;**
- **definiţia subprogramului.**

Subprogramul se poate identifica printr-un **nume** care este folosit atât pentru definiţia subprogramului, cât şi pentru prototip şi activarea lui (apelarea lui).

```
# include <iostream>

using namespace std:

void functie();                                   Prototipul subprogramului

int main()                                        Antetul subprogramului
{
    functie();                  Apelul subprogramului     Definiţia subprogramului     Modulul apelant
    return 0;
}

void functie()                                    Antetul subprogramului
{
 cout<<"exemplu de functie";                      Definiţia subprogramului     Modulul apelat
}
```

1. Prototipul subprogramului

Este o linie de program, aflată înaintea modulului care apelează subprogramul, prin care se comunică compilatorului informaţii despre subprogram (se declară subprogramul). Prin declararea subprogramului, compilatorul primeşte informaţii despre modul în care se poate apela subprogramul şi poate face verificări la apelurile de subprogram în ceea ce priveşte tipul parametrilor folosiţi pentru comunicare şi a modului în care poate face conversia acestor parametri.

Prototipul unei funcţii se scrie astfel:

<tip_rezultat> **<nume_funcţie>** **(** <listă_parametr_formali> **);**

Unde:

<tip_rezultat> este tipul returnat de funcție. (**int** – funcția va returna un întreg, **void** – funcția nu returnează un tip anume, **float** – funcția va returna un număr real)

<nume_funcţie> este numele funcției. **OBS**. Numele funcției este un identificator deci **nu** trebuie să conțină caracterul **spațiu**.

<listă_parametr_formali> este lista parametrilor formali. Lista de parametri este de forma: **<tip$_1$> param$_1$, <tip$_2$> param$_2$, …., <tip$_n$> param$_n$**

În prototipul funcţiei se indică: **tipul** de dată returnat de funcţie, **numele** acesteia şi **lista** declaraţiilor parametrilor formali. La fel ca un operand sau o expresie, o funcţie are un tip, care este dat de tipul valorii returnate de funcţie în funcţia apelantă. Dacă funcţia nu întoarce nici o valoare, în locul **<tip_rezultat>** se specifică **void**. Dacă **<tip_rezultat>** lipseşte, se consideră, implicit, că acesta este **int. <nume_funcţie>** este un identificator.

<listă_parametr_formali> (încadrată între **paranteze rotunde**) constă într-o listă (enumerare) care conţine tipul şi identificatorul fiecărui parametru de intrare, despărţite prin virgulă. Tipul unui parametru poate fi oricare, chiar şi tipul pointer. Dacă lista parametrilor formali este vidă, în antet, după numele funcţiei, apar doar parantezele **()**, sau **(void)**.

Exemple:

```
void Afiseaza(void);
void Afiseaza();
int Calcul(int x, int y, float m);
unsigned long Medie(unsigned long x, double z);
```

În exemplul descris la început declarația de prototip este:

```
void functie();
```

OBS. Prototipul se termină obligatoriu cu caracterul **punct și virgulă.**

Alte exemple de prototipuri ale unor funcții:

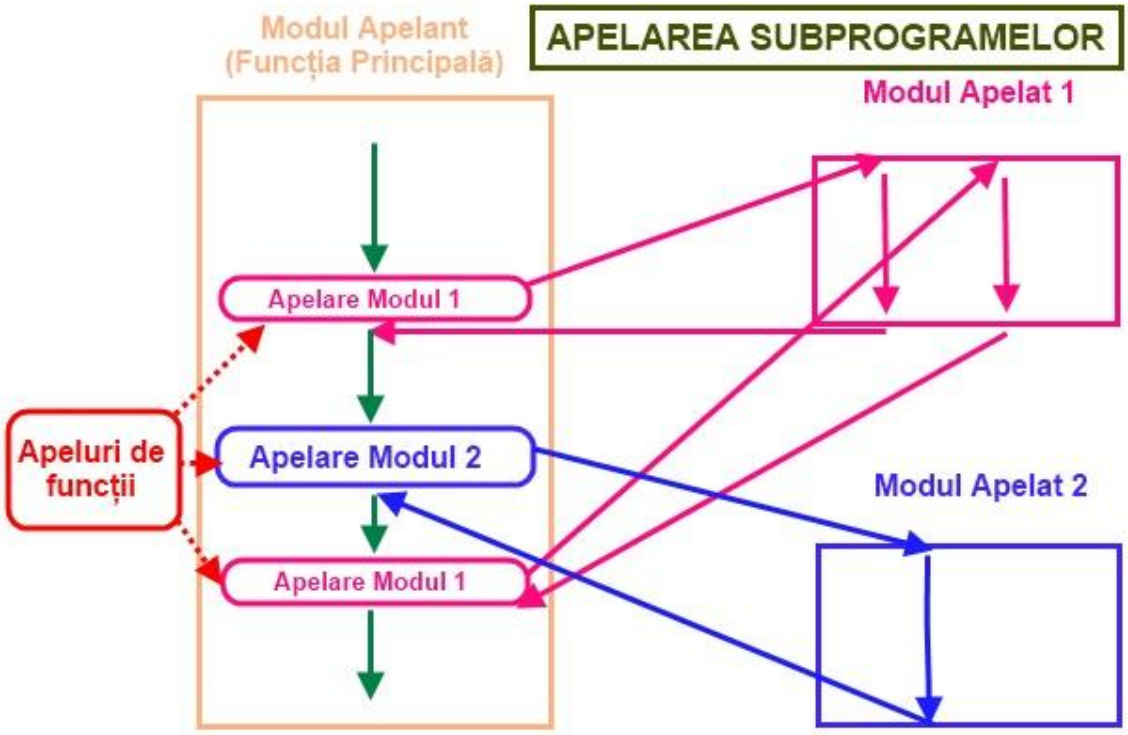
```
void Citeste(int , float);

void Citeste (int &n, float &m);
```

În primul exemplu de prototip în lista parametrilor formali acestora le lipsesc numele. Numele parametrilor sunt opționali în definiția de prototip dar **OBLIGATORIU** în definiția ce urmează a fi scrisă lista parametrilor trebuie să fie completă prin tipurile și numele aferente.

2. Apelul subprogramului

Apelarea subprogramului în cadrul unui bloc înseamnă activarea subprogramului, adică lansarea lui în execuţie. Subprogramul poate fi apelat ori de câte ori este nevoie (nu există restricţii pentru numărul de apeluri)



Modulul apelant se execută secvenţial (instrucţiune cu instrucţiune). La apelarea subprogramului, este părăsit blocul modulului apelant şi se trece la executarea instrucţiunilor din subprogram. După ce se termină executarea acestor instrucţiuni, se revine la blocul apelant şi se continuă execuţia cu instrucţiunea care urmează apelului.

În exemplul de la început apelarea subprogramului este:

```
functie();
```

Iar pentru celelalte exemple unele apeluri pot fi:

```
Afiseaza();
r= Calcul(a, b, c)+a%15;
a=b+ Calcul(a, b, c)+d%10;
cout<< Calcul(a, b, c);
r= s+Medie(a, b)+c%10;
cout<< Medie(a, b)+c;
```

3. Definiţia subprogramului

Definiţia unui subprogram este formată din antetul şi corpul subprogramului:

```
<tip_rezultat>  <nume_funcţie> ( <listă_parametr_formali> )
{
      <declaraţii proprii subprogramului>
      <instrucţiuni>
      [return <expresie>;]
}
```

a. **Antetul subprogramului**. Este o linie de recunoaştere a subprogramului, in care i se atribuie un nume. El specifică începutul subprogramului.

<tip_rezultat> **<nume_funcţie>** **(** <listă_parametr_formali> **)**

Unde:

<tip_rezultat> este tipul returnat de funcție. (**int** – funcția va returna un întreg, **void** – funcția nu returnează un tip anume, **float** – funcția va returna un număr real)

<nume_funcţie> este numele funcției. **OBS**. Numele funcției este un identificator deci **nu** trebuie să conțină caracterul **spațiu**.

<listă_parametr_formali> este lista parametrilor formali. Lista de parametri este de forma: **<tip$_1$>** **param$_1$**, **<tip$_2$>** **param$_2$**, …., **<tip$_n$>** **param$_n$**

În exemplul de la început antetele de subprograme sunt:

```
void functie()
int main()
```

OBS. Antetele nu se termină cu caracterul **punct și virgulă** deoarece urmează definirea corpului subprogramului respectiv.

b. **Corpul subprogramului**. La fel ca orice bloc C++, este încapsulat într-o instrucţiune compusă, delimitată de caracterele **{...}**

```
{
    <declaraţii proprii subprogramului>
    <instrucţiuni>
    [return <expresie>;]
}
```

Corpul subprogramului este format din două părţi:

- **Partea declarativă**. Conţine definiţii de elemente folosite numai în interiorul subprogramului: tipuri de date, constante şi variabile de memorie. Nu se pot defini şi alte subprograme (nu este valabilă tehnica de imbricare a subprogramelor existentă în alte limbaje de programare).

```
<declaraţii proprii subprogramului>
```

- **Partea executabilă**. Conţine instrucţiunile prin care sunt descrise acţiunile realizate de subprogram.

```
<instrucţiuni>
[return <expresie>;]
```

În exemplul de la început corpul subprogramelor sunt:

- pentru subprogramul **functie** avem:

```
{
    cout<<"exemplu de functie";
}
```

- pentru subprogramul **main** avem:

```
{
    functie();
    return 0;
}
```

Observăm că pentru ambele funcții lipsește partea declarativă a subprogramului și avem doar partea executivă.

Aşadar:

- **Prototipul** subprogramului **declară** subprogramul.
- **Apelul** subprogramului **execută** subprogramul.
- **Antetul** subprogramului **specifică** numele subprogramului şi tipul argumentelor şi al valorilor returnate, iar corpul subprogramului îl defineşte, adică specifică ceea ce trebuie să realizeze subprogramul.

OBS: Orice subprogram trebuie **declarat** şi **definit**. Declararea unui subprogram este necesară pentru ca subprogramul să fie cunoscut de subprogramele care îl apelează.

Declararea subprogramului poate fi făcută:

1. **prin definiţia lui** (antetul împreună cu corpul subprogramului) înaintea funcţiilor care îl apelează.

Exemplul de mai sus poate fi rescris utilizând definiția funcției înainte de a fi apleată:

```
# include <iostream>
using namespace std:
```

`void functie()`	**Antetul subprogramului**		**Modulul apelat**

`{`
`cout<<"exemplu de functie";`
`}`
		Definiţia subprogramului	

`int main()`	**Antetul subprogramului**		**Modulul apelant**

`{`
`functie();`
`return 0;`
`}`
	Apelul subprogramului	**Definiţia subprogramului**	

2. scrierea **prototipului** înaintea funcţiilor care îl apelează și apoi **definiţia** lui oriunde în program la fel cum s-a procedat în exemplul de la început