Definiția subprogramului

Blocul este unitatea de bază a oricărui program C++ şi este încapsulat într-o instrucţiune compusă delimitată de caracterele **{ ... }**.

El este format din două părţi:

- Partea declarativă conţine definiţii de elemente necesare algoritmuiui pentru rezolvarea problemei: constante (**const**), variabile de memorie şi tipuri de date (**typedef**). Definirea lor se face cu ajutorul instrucţiunilor declarative.

- Partea executabilă sau partea procedurală conţine instrucţiunile care descriu paşii algoritmului care trebuie implementat pentru rezolvarea problemei. Aceste instrucţiuni se numesc instrucţiuni imperative. Ele sunt:: **instrucţiunea expresie** (prin care se evaluează o expresie) şi **instrucţiunile de control** (prin care se modifică ordinea de execuţie a programului). Instrucţiunea expresie prin care se atribuie unei variabile de memorie o valoare se mai numeşte şi instrucţiune de atribuire, iar instrucţiunea expresie prin care se cere execuţia unui subprogram se mai numeşte şi instrucţiune procedurală.

Orice program C++ este o colecţie de definiţii de variabile şi funcţii.

Funcţia este un bloc precedat de un antet prin care se precizează numele ei şi, dacă este cazul, tipul rezultatului pe care-l întoarce prin chiar rezultatul său şi, eventual, parametrii de execuţie (valori care se transmit blocului şi care sunt necesare atunci când se execută blocul):

<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: **$<tip_1>$ $param_1$, $<tip_2>$ $param_2$, …., $<tip_n>$ $param_n$**

OBS. Fiecare parametru se delimitează în lista de parametri formali prin caracterul **virgulă** și trebuie să aibă definit obligatoriu **tipul** de dată. Este importantă și **ordinea** în care aceștia se declară.

Exemplu:

void Citire(int a[10], int n)

Funcția are numele **Citire**, returnează tipul **void** și are 2 parametri: vectorul cu numele **a** de lungime maximă 10 întregi și al doilea un număr întreg **n**.

Una dintre funcţiile programului C++ este funcţia rădăcină. Ea este obligatorie şi este primul bloc cu care începe execuţia programului. Numele său este **main**. Antetul acestei funcţii este:

int main()

ceea ce semnifică faptul că funcţia se numește **main**, întoarce un întreg (**int**) şi nu are parametri pentru apelare deoarece parantezele **(.....)** nu conţin listă de parametri.

Exemplu. *Se citesc de la tastatură 3 numere întregi a, b, c. Să se ordoneze cele 3 numere în ordine crescătoare și să se afișeze pe ecran.*

Rezolvare: Se vor executa următorii pași:

***Pasul 1**: Se declară cele trei numere.*

***Pasul 2**: Se citesc numerele.*

***Pasul 3**: Se compară numerele a și b dacă sunt în ordine crescătoare (a să fie mai mic decât b). Dacă nu sunt în ordine atunci se schimbă valorile între ele.*

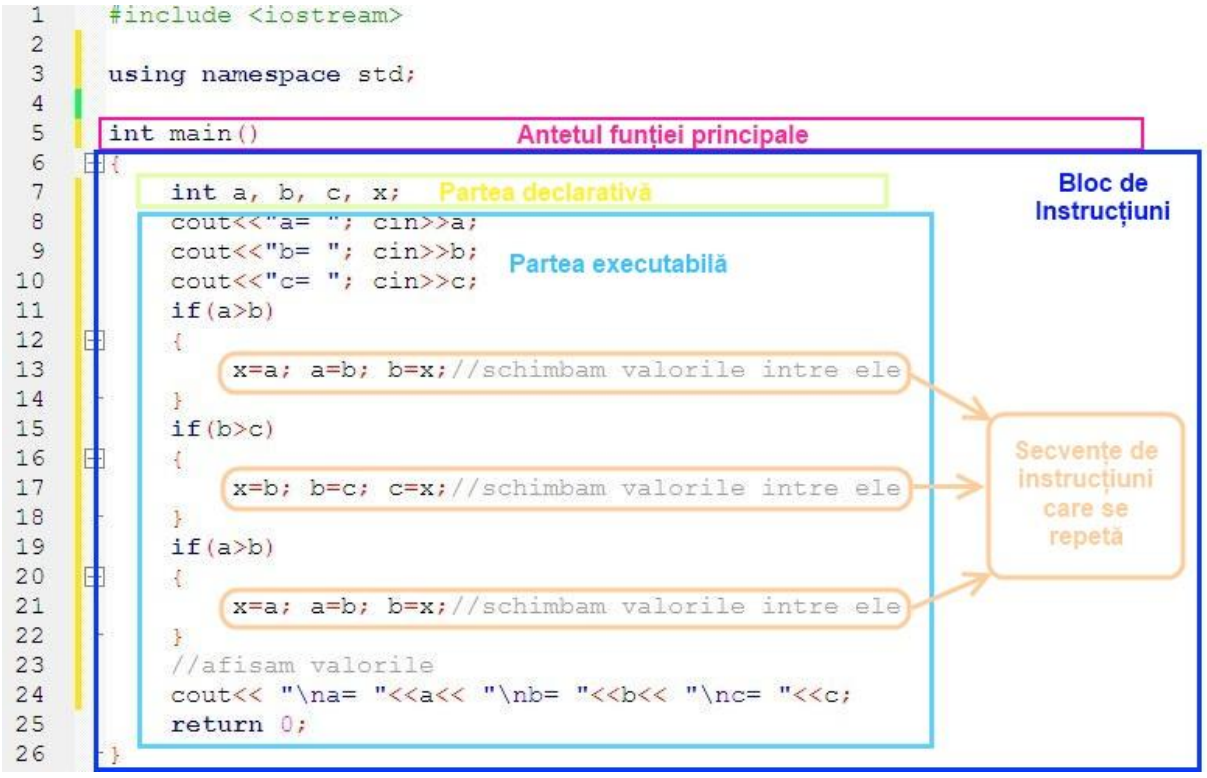
***Pasul 4**: Se compară numerele b și c dacă sunt în ordine crescătoare (b să fie mai mic decât c). Dacă nu sunt în ordine atunci se schimbă valorile între ele.*

***Pasul 5**: Se compară numerele a și b dacă sunt în ordine crescătoare (a să fie mai mic decât b). Dacă nu sunt în ordine atunci se schimbă valorile între ele.*

Se observă că acum numerele sunt în ordine crescătoare

***Pasul 6**: Se afișează valorile celor trei numere.*

Implementarea în C++ este următoarea:



În imaginea de mai sus sunt evidențiate:

- **antetul funcției principale**
- **-blocul de instrucțiuni** în cadrul căruia avem**:**
  - **Partea descriptivă**
  - **Partea executabilă**

Observăm că în cazul cel mai defavorabil (când a, b și c sunt în ordine descrescătoare), secvența de schimbare a valorile între ele (`x=a; a=b; b=x;`) se repetă de 3 ori.

Prin definiţie: **Subprogramul** este o secvenţă de instrucţiuni care rezolvă o anumită sarcină şi care poate fi descrisă separat de blocul rădăcină şi lansată în execuţie din cadrul unui bloc, ori de câte ori este nevoie.

În limbajul C++ subprogramele se mai numesc şi **funcţii**.

1. Necesitatea folosirii subprogramelor

În practica programării pot să apară următoarele cazuri:

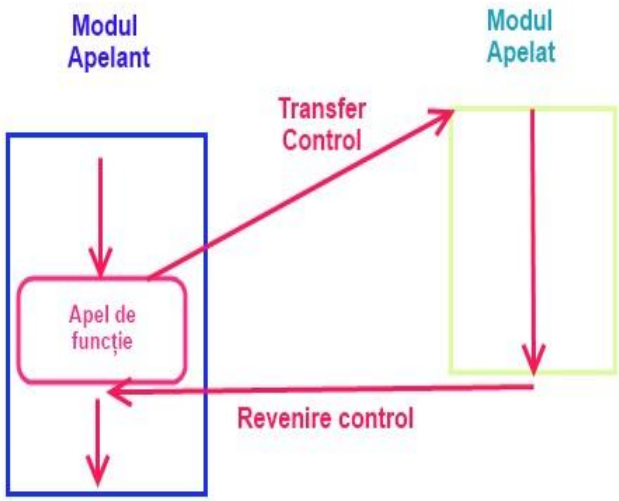
1. **Secvenţă de instrucţiuni care se repetă de mai multe ori în cadrul unui program** (ca în programul C++ din exemplul anterior). Secvenţa de instrucţiuni care se repetă poate fi implementată sub forma unui subprogram.

2. **Rezolvarea unei anumite sarcini este necesară în mai multe programe**, ca, de exemplu, diferite operaţii matematice (extragerea radicalului, extragerea părţii întregi sau a părţii fracţionare dintr-un număr real, ridicarea unui număr la o putere etc), diferite operaţii cu şiruri de caractere (extragerea unui subşir dintr-un şir, ştergerea unui subşir dintr-un şir, inserarea unui subşir într-un şir etc), diferite operaţii cu tablouri de memorie (crearea, parcurgerea şi sortarea tabloului de memorie, ştergerea sau inserarea unui element etc), diferite operaţii cu fişiere (deschiderea unui fişier, închiderea unui fişier, testarea sfârşitului de fişier etc). Secvenţa de instrucţiuni care rezolvă o anumită sarcină ce poate să apară în mai multe programe poate fi implementată cu ajutorul unui subprogram.

3. **Orice problemă poate fi descompusă în subprobleme**. Subproblemele în care este descompusă se numesc module. Descompunerea poate continua până când se obţine un modul cu rezolvare imediată. Această metodă de rezolvare a unei probleme se numeşte tehnica **top-down** de proiectare a algoritmilor. Ea este foarte utilă în cazul programelor care trebuie să rezolve probleme complexe (de exemplu: prelucrarea listelor liniare, prelucrarea vectorilor, prelucrarea şirurilor de caractere etc). în aceste cazuri se obţin programe foarte mari şi complexe. Pentru a obţine programe mai simple şi mai clare se poate fragmenta problema iniţială în subprobleme, fiecare subproblemă fiind descrisă printr-un subprogram.

4. Terminologie folosită pentru subprograme

Într-o structură modulară în care fiecare modul este descris printr-un subprogram, modulele se clasifică astfel:
- **Modul apelant**. Este modulul care, pentru rezolvarea propriei probleme, apelează la alte module, fiecare dintre ele rezolvând o anumită subproblemă. La apelare, el transferă controlul modulului apelat.
- **Modul apelat**. Este modulul apelat de un alt modul, pentru a-i rezolva o subproblemă. După ce îşi termină execuţia, el redă controlul modulului apelant.



Vom considera funcţia rădăcină `main()` ca fiind modulul principal sau subprogramul principal, iar celelalte funcţii (module) pe care le vom defini le vom numi subprograme (funcții).

5. Avantajele folosirii subprogramelor

În practică, pentru rezolvarea unor probleme complexe care ajută la îndeplinirea unor activităţi, cum sunt de exemplu prelucrările de texte, contabilitatea unei întreprinderi, inventarierea unor depozite de materiale, gestionarea unei biblioteci etc, trebuie să se conceapă programe sofisticate numite aplicaţii. în construirea unei aplicaţii folosirea subprogramelor oferă următoarele avantaje:

- **Se face economie de memorie internă**. Un grup de instrucţiuni care trebuie să se execute de mai multe ori într-o aplicaţie (chiar cu date de intrare şi de ieşire diferite) se va scrie o singură dată într-un subprogram şi se va executa prin apelarea subprogramului ori de câte ori este nevoie.
- **Se favorizează lucrul în echipă pentru aplicaţiile mari**. Fiecare programator va putea să scrie mai multe subprograme, independent de ceilalţi programatori din echipă. Pentru a realiza subprogramul, este suficient să i se precizeze programatorului specificaţiile subprogramului: datele de intrare, datele de ieşire şi problema pe care trebuie să o rezolve.
- **Depanarea şi actualizarea aplicaţiei se fac mai uşor**. După implementare şi intrarea în exploatare curentă, o aplicaţie poate necesita modificări ca urmare a schimbăriii unor cerinţe. Este mult mai simplu să se gândească modificarea la nivelul unui subprogram, decât la nivelul întregii aplicaţii.
- **Creşte portabilitatea programelor**. Subprogramele sunt concepute independent de restul aplicaţiei şi unele dintre ele pot fi preluate fără un efort prea mare şi în alte aplicaţii, în care trebuie să fie rezolvate sarcini similare.