Funcții C++

Sau cum modularizăm programul

Funcția

- Def. Este o porţiune de cod dintr-un program, care efectuează o anumită sarcină, este relativ independentă de restul codului, și este identificabilă și apelabilă prin intermediul unui nume.

Structura unei funcții

tip nume_funcție(parametri formali)
{
  <declarații>
  <instrucțiuni>
}

Primul rând este **antetul** funcției.

Partea inclusă între acolade, împreună cu acoladele, formează **corpul** funcției.

Tipul de dată returnat

- În C/C++ există două tipuri de funcții:
- Cele care la revenirea din apel returnează o valoare:
  - Pentru acestea **tip** reprezintă tipul de dată returnat, care poate fi orice tip de dată predefinit sau pointer
  - **int suma()**
- Cele care la revenire **nu** returnează o valoare
  - Pentru acestea **tip** trebuie să fie **void**
  - **void nimic()**

Numele funcției

- Numele funcției este un identificator, deci trebuie să respecte regulile de definiție a identificatorilor.
- Se recomandă ca numele funcțiilor să fie sugestive și clare

Parametrii funcției

- O funcție poate avea 0 sau mai mulți parametri. Lista parametrilor e vidă, dacă aceștia nu există. În acest caz declarația antetului funcției ar arăta astfel:

  tip nume() sau **tip** nume(void)
- În cazul în care funcția are parametri aceștia se includ între parantezele rotunde și se separă prin virgulă dacă sunt mai mulți.
- Exemplu:
- int modul(int x)
- float suma(float a, float b)
- int operatie(char a, int nota, float media)

Parametrii funcției

- Parametrii se folosesc pentru a transmite/prelua date funcției la momentul apelului.
- Utilizarea parametrilor crește independența funcției de restul programului.
- Parametrii declarați în antetul funcției se numesc formali.
- Parametrii transmiși în momentul apelului funcției se numesc efectivi.
- Parametrii efectivi trebuie să corespundă cu cei formali ca număr, ordine și tip.

Parametrii funcției

- Parametrii unei funcții pot fi transmiși prin valoare, adresă (pointer) sau prin referință.
- Dacă parametrul a fost transmis prin valoare atunci valoarea parametrului efectiv a fost transmisă către parametrul formal și orice modificare asupra parametrului formal **nu** este vizibilă în parametrul efectiv.
- Dacă parametrul a fost transmis prin referință modificările efectuate asupra parametrului formal în interiorul funcției **sunt** vizibile, fiind modificat parametrul efectiv transmis.
- Exemplu: int suma(int a, int &b, int *c)
- Parametrul transmis primul va fi prin valoare, al doilea prin referință, iar al treilea prin adresă.

Exemplu:

```
#include <iostream>
using namespace std;
int x=3,y=5;
int suma(int a, int &b) // a,b parametri formali, a transmis prin
    valoare și b prin referință
{   a++;b++;  return a+b; }
int main()
{   cout<<suma(x,y)<<endl; // apelul funcției suma. x,y sunt
    parametri efectivi
   cout << x<<" "<<y << endl;
   return 0;
}
```

Parametri cu valoare implicită

- Putem să decidem că anumiți parametri sunt opționali și pentru aceștia să punem o valoare implicită, dar **doar** pentru ultimii.

```
#include <iostream>
using namespace std;
int x=3, y=7;
int suma(int x, int y=5){    return x+y;}
int main()
{
   cout<<suma(x,y)<<endl;
   cout<<suma(y)<<endl;
   return 0;
}
```

Funcții cu număr variabil de parametri

- Limbajul C permite definirea funcțiilor cu număr variabil de parametri
- Pentru a folosi această facilitate trebuie inclusă librăria <cstdarg>
- Definire **tip** nume(**tip1** arg1, **tip2** arg2, …)
- Primul argument e obligatoriu, restul sunt substituiți de …
- Trebuie găsită o metodă de a determina numărul de parametri
- Este indicat ca primul argument să ne dea numărul de parametri sau ultimul parametru să fie o valoare cunoscută (ex. 0)

Funcții cu număr variabil de parametri - exemplu

```
#include <iostream>
#include <cstdarg>
using namespace std;
int suma(int nr, ...)
{
    va_list arg; // lista de argumente
    int s=0, i;
    va_start(arg, nr);//populeaza lista
    for(i=1;i<=nr;i++)
        s+=va_arg(arg, int); // extrage un argument din lista
    va_end(arg); // elibereaza memoria alocata listei
    return s;
}
int main()
{
    cout << suma(5, 6, 1, 3, 6, 1);
    return 0;
}
```

Parametri vector

```
int suma1(int *x, int n)
{
    int i, s=0;
    for(i=0;i<n;i++)s+=x[i];
    return s;
}
int suma2(int x[], int n);

int suma3(int x[100], int n);
```

Cele 3 declarații de funcții sunt echivalente.

Parametri matrice

```
typedef int matrice[100][100];
int suma(int x[100][100],int n, int m);
int suma2(int x[][100],int n, int m);
int suma3(matrice x, int n, int m)
{
    int i,j,s=0;
    for(i=1;i<=n;i++)
        for(j=1;j<=m;j++)s+=x[i][j];
    return s;
}
```

Cele trei variante de mai sus sunt echivalente.

Parametri structuri

```
typedef struct nr_mare{int a[1000],n;};
nr_mare x,y;
nr_mare suma(nr_mare a, nr_mare b);
```

- În exemplul de mai sus funcția **suma** primește 2 parametri de tip nr_mare și returnează un nr_mare.

Exercițiu

- Determinați ce afișează secvența de program

```
int a;
void f(int x){x=x+1;}
int main()
{a=1;
 f(a);
 cout<<a;
}
```

Raspuns: 1

Exercițiu

- Determinați ce afișează secvența de program

```
int a;
void f(int &x){x=x+1;}
int main()
{a=1;
 f(a);
 cout<<a;
}
```

Raspuns: 2

Corpul funcției

- În corpul funcției putem declara tipuri de dată, variabile și scrie instrucțiuni.
- Variabilele declarate în interiorul unei funcții (și parametrii transmiși prin valoare) sunt alocate pe stivă și există doar pe perioada execuției funcției.
- Tipurile de dată declarate în funcție sunt vizibile/utilizabile doar în aceasta.
- Dacă funcția returnează o valoare trebuie ca în corpul funcției să existe **return** *expresie*, unde *expresie* este de tipul returnat de funcție.

Exemplu

- Fie funcția modul

```
int modul (int x)
{
  if(x<0) return –x;
   else return x;
}
```

Funcționare: dacă x e negativ returnează numărul cu semnul schimbat, altfel returnează numărul așa cum este.

Exercițiu

- Determinați ce afișează secvența de program de mai jos:

```
int fun(int a, int b)
{
 a=a+3;b=b+1;
 return a+b;
}
int main()
{ cout<<fun(3,5);
 return 0;
}
```

Răspuns: 12

Apelul unei funcții

- Apelul unei funcții se poate realiza în două moduri:
  - Printr-o instrucțiune de apel
  - Ca operand într-o expresie
- Apelul se face: nume(listă_parametri_actuali)
- Parametrii actuali se mai numesc și parametri efectivi
- Lista parametrilor efectivi este formată din expresii separate prin virgulă.
- Parametrii actuali trebuie să corespundă cu cei formali ca număr, ordine și tip.
- La apelul unei funcții, valorile parametrilor efectivi sunt atribuite, în ordine, parametrilor formali corespunzători.
- Parametrii actuali care sunt transmiși prin referință trebuie să fie variabile.

Exerciții

- Care din apelurile de mai jos ale funcției op sunt corecte ?

```
int op(int a, int &b){a++;b++;return a+b;}
int main()
{ int x=1,a=2,b=3,y=4; float z=6,k=7;
  cout<<op(a,b);
  x=op(x,y);
  z=op(5,y);
  k=op(a+b,x+y);
  z=op(x+k, y);
  cout<<op(5,a)+op(x,y);
  cout<<op(op(4,a),b);
}
```

Domeniul de vizibilitate al identificatorilor

- Un identificator definit la începutul programului este vizibil în tot programul și în toate subprogramele.
- Un identificator definit într-o funcție este vizibil doar în acea funcție, după terminarea funcției el încetând să mai existe.

Domeniul de vizibilitate al identificatorilor

```
int a=4, b=8;                    // a,b variabile globale
int f(int x, int &y)       // x,y parametri formali
{int b=4;                  // b, variabila locală
 x++; y--;
 b=b+y; return b+x;
}
int main()
{ f(a,b);                  // a,b parametri efectivi
cout<<a<<“ ”<<b;
  b=f(a,b); cout<<a<<“ ”<<b;
}
```

Exercițiu

- Determinați ce afișează secvența de program de mai jos:

```
int a=5,b=7;
int f(int x)
{ int a;            //linia 3
  a=b+x;
  return a+b;
}
int main()
{cout<<f(b)<<“\n”;
 cout<<a<<” ”<<b;
return 0;}
```

Exercițiu

- Determinați ce afișează secvența de program de mai jos

```
int a=4, b=8;
int f(int x, int &y)
{int b;
 x++; y--;
 b=b+y; return b;
}
int main()
{ a=f(a,b); cout<<a<<“ ”<<b<<“\n”;
  b=f(a,b); cout<<a<<“ ”<<b;
}
```

Pointeri la funcții

- Numele unei funcții reprezintă adresa de memorie la care începe funcția.
- Numele funcției este, de fapt, un pointer la funcție.

- Dacă o funcție se declară:

  tip_returnat nume_functie(lista parametri)

- Un pointer la funcție se declară:

  tip_returnat (*nume_pointer_functie)(lista parametri)

Exemplu utilizare

```
#include <iostream>
#include <algorithm>
#include <cstdlib>
using namespace std;
typedef struct elev
{
    char s[20];
    double nota;
};
elev a[500];
int n,i;
int conditie(elev a, elev b)
{ //daca sunt bine ordonate returneaza 1
    if(a.nota>b.nota)return 1;
     else return 0;
}
int main()
{
    cin>>n;
    for(i=1; i<=n; i++)
    {
        a[i].nota=1.0*(rand()%101+1)/10;
        a[i].s[0]='A'+rand()%26;
    }
    sort(a+1,a+n+1,conditie);
    for(i=1; i<=n; i++)cout<<a[i].s<<" "<<a[i].nota<<endl;
    return 0;
}
```

Exemplu

```
#include <iostream>
using namespace std;
int suma(int a, int b){ return a+b;}
int produs(int a, int b){ return a*b;}
int cat(int a, int b){ return a/b;}
void afiseaza(int (*operatie)(int, int), int x, int y)
{
    cout<<operatie(x,y)<<endl;
}
int main()
{
    afiseaza(suma,5,8);
    afiseaza(produs,4,9);
    afiseaza(cat,7,3);
    return 0;
}
```