

Programarea calculatoarelor

11 C++
Functii

Adrian Runceanu
www.runceanu.ro/adrian

Curs 11

Capitolul 9. Functii

9.1. Declararea funcției

9.2. Apelul funcției

9.3. Prototipul funcției

9.4. Parametri formali și actuali

9.5. Variabile locale și variabile globale

9.6. Variabile statice și variabile automate

9.7. Funcții matematice

În limbajul C/C++, una din cele mai importante facilități o constituie folosirea funcțiilor.

De fapt, *funcțiile reprezintă locul unde sunt scrise și executate instrucțiunile oricărui program.*

Definiție: *O funcție în C/C++ este o construcție independentă care conține declarații și instrucțiuni și care realizează o anumită acțiune.*

Pentru a construi și folosi o funcție trebuie să cunoaștem trei elemente care sunt implicate în utilizarea funcțiilor:

- 1. Declarația funcției**
- 2. Apelul funcției**
- 3. Prototipul funcției**

9.1. Declararea funcției

Forma generală a unei funcții este următoarea:

```
tip nume_funcție (lista de parametri formali)  
{  
    declarații variabile locale  
    instrucțiuni  
}
```

9.1. Declararea funcției

Unde:

- **tip** – reprezintă tipul valorii returnate de funcție
- **lista parametri formali** – reprezintă variabilele folosite în cadrul funcției împreună cu tipul fiecăruia dintre ele, dar care au nume (denumiri) generice – **formale** – *care nu trebuie neapărat să coincidă cu denumirile variabilele folosite în alte funcții sau chiar în funcția principală.*

9.1. Declararea funcției

- **declarații variabile locale** - reprezintă zona de declarare a variabilelor folosite doar în cadrul corpului funcției respective și care nu se pot folosi în alte funcții.
- **instrucțiuni** - reprezintă secvența de instrucțiuni care formează funcția considerată.

Observație:

Dacă *lista parametrilor formali este vidă*, atunci reprezentarea declarării ei se face astfel:

tip nume_funcție(void)

Capitolul 9. Functii

9.1. Declararea funcției

9.2. Apelul funcției

9.3. Prototipul funcției

9.4. Parametri formali și actuali

9.5. Variabile locale și variabile globale

9.6. Variabile statice și variabile automate

9.7. Funcții matematice

9.2. Apelul funcției

2. Apelul funcției

Apelul unei funcții se realizează astfel:

```
nume_funcție( $p_{a1}$ ,  $p_{a2}$ ,  $\dots$ ,  $p_{an}$ );
```

Unde p_{a1} , p_{a2} , \dots , p_{an} , reprezintă lista parametrilor actuali (reali) cu care se folosește funcția respectivă.

De reținut:

1. *Apelul poate să apară într-o **instrucțiune de apel** dacă funcția returnează sau nu o valoare.*
2. *Apelul unei funcții poate să apară într-o **expresie** numai dacă funcția returnează o valoare.*

Capitolul 9. Functii

9.1. Declararea funcției

9.2. Apelul funcției

9.3. Prototipul funcției

9.4. Parametri formali și actuali

9.5. Variabile locale și variabile globale

9.6. Variabile statice și variabile automate

9.7. Funcții matematice

9.3. Prototipul funcției

3. Prototipul funcției

Pentru a funcționa corect programul, orice funcție trebuie declarată anterior folosirii ei.

Declararea este necesară dacă funcția este definită în altă parte decât în fișierul în care este apelată, sau dacă este definită în același fișier dar în urma apelării.

Prototipul unei funcții are următoarea formă generală:

tip nume_funcție(lista declarații parametri);

9.3. Prototipul funcției

Unde **tip** – reprezintă tipul valorii returnate de funcție:

- dacă nu se specifică, atunci implicit (automat) este considerat de către compilatorul C/C++ ca fiind tipul **int**.
- dacă **tip** este **void** atunci funcția nu returnează nici o valoare, și deci poate acționa ca o procedură.

9.3. Prototipul funcției

- **lista declarații parametri** poate avea una din următoarele patru forme:
 1. În listă apar **numai tipul de date** al parametrilor separați prin virgulă
 2. În listă apar **atât tipul de date al parametrilor cât și numele lor** separate prin virgulă
 3. Lista nu este specificată
 4. Lista este vidă – **void**

9.3. Prototipul funcției

Exemplu:

Prezentăm în continuare declararea unei **funcții de tip double** care are trei parametri: unul întreg, unul de tip double și al treilea de tip caracter, în cele patru forme amintite mai sus:

1. **double f(int, double, char);**
2. **double f(int i, double x, char c);**
3. **double f();** // nu înseamnă ca funcția nu are parametri, ci doar compilatorul nu va mai face, la apel, verificarea tipului parametrilor
4. **double g(void);** // nu are parametri

Se recomandă folosirea formelor 1) sau 2).

9.3. Prototipul funcției

Problema 1:

Următorul program conține prototipul și apoi definirea a două *funcții care ridică la cub o valoare întreagă*, respectiv *o valoare reală*:

9.3. Prototipul funcției

```
#include<iostream.h>
```

```
int intreg_la_cub(int);
```

```
float real_la_cub(float);
```

Prototipul funcției intreg_la_cub

Prototipul funcției real_la_cub

```
int main()
```

```
{
```

```
    cout<<"\n 3 la cub este "<< intreg_la_cub(3);
```

```
    cout<<"\n 5.2 la cub este "<< real_la_cub(5.2);
```

```
}
```

Apelul funcției intreg_la_cub

Apelul funcției real_la_cub

9.3. Prototipul funcției

Definiția funcției `intreg_la_cub`

```
int intreg_la_cub(int valoare)
{
    return (valoare * valoare * valoare);
}
```

Definiția funcției `real_la_cub`

```
float real_la_cub(float valoare)
{
    return (valoare * valoare * valoare);
}
```

Capitolul 10. Funcții

10.1. Declararea funcției

10.2. Apelul funcției

10.3. Prototipul funcției

10.4. Parametri formali și actuali

10.5. Variabile locale și variabile globale

10.6. Variabile statice și variabile automate

10.7. Funcții matematice

10.4. Parametri formali și actuali

4. Parametri formali și actuali

În definiția funcțiilor **parametrii formali** sunt de fapt *numele parametrilor care apar în construcția funcției*.

Exemplu: Definim o funcție care următorii parametri formali: vârstă, salariu, și nr_cod, astfel:

```
void info_angajat(int vârstă, float salariu, int nr_cod)
{
    // instrucțiunile funcției
}
```

10.4. Parametri formali și actuali

Atunci când o funcție apelează o altă funcție, **valorile transmise de funcția apelantă** sunt **parametrii actuali** (sau **parametrii reali**).

Astfel, dacă se apelează funcția cu valorile 34, 4500.00 și 101, aceste valori reprezintă parametrii actuali în apelul funcției:

```
info_angajat(34, 4500.00, 101);
```

Parametrii actuali pe care îi folosește o funcție pot fi valori constante sau variabile.

Valoarea și tipul parametrilor actuali trebuie să se potrivească cu valoarea și tipul parametrilor formali.

10.4. Parametri formali și actuali

Exemplu:

Următoarea secvență de program ilustrează modul de folosire a variabilelor ca parametri actuali:

```
int vîrsta_angajat = 34;  
float salariu_angajat = 4500.00;  
int număr_cod = 101;  
info_angajat(vîrsta_angajat, salariu_angajat,  
             număr_cod);
```

10.4. Parametri formali și actuali

Astfel, *atunci când se apelează o funcție folosind ca variabile parametri actuali, numele variabilelor utilizate nu au nici o legătură cu numele parametrilor formali.*

Compilerul C/C++ va lua în considerare numai valorile pe care le au variabilele respective(parametrii actuali).

Capitolul 10. Funcții

10.1. Declararea funcției

10.2. Apelul funcției

10.3. Prototipul funcției

10.4. Parametri formali și actuali

10.5. Variabile locale și variabile globale

10.6. Variabile statice și variabile automate

10.7. Funcții matematice

10.5. Variabile locale și variabile globale

Din punctul de vedere al vizibilității variabilelor avem:

1. **variabile locale**

2. **variabile globale**

10.5. Variabile locale și variabile globale

1. Variabilele locale se declară în cadrul funcțiilor și se numesc *locale* deoarece numele și valorile lor *sunt valabile doar în cadrul funcției respective*.

10.5. Variabile locale și variabile globale

Exemplu:

Următoarea funcție numită `var_locale` conține

- trei variabile locale numite x, y și z
- cărora le atribuie valorile 10, 20 și 30
- și apoi execută afișarea valorilor lor,
- iar în programul principal, **compilatorul va genera cod de eroare** la încercarea nereușită de a afișa aceleași valori ale variabilelor locale funcției considerate:

10.5. Variabile locale și variabile globale

```
#include<iostream.h>
void var_locale(void)
{
    int x = 10, y = 20, z = 30;
    cout<<"Variabila x = "<<x<<" variabila y = "<<y<<"variabila
    z = " <<z<<"\n";
}
int main()
{
    var_locale();
    cout<<"Variabila x = "<<x<<" variabila y = "<<y<<"variabila
    z = " <<z<<"\n";
}
```

10.5. Variabile locale și variabile globale

The screenshot shows a C++ IDE with a project named `curs11_1`. The `FileView` pane on the left shows the project structure: `Source Files`, `Header Files`, `Resource Files`, and `Other Files`. The main editor displays the file `curs11_1.cpp` with the following code:

```
1  #include<iostream.h>
2  void var_locale(void)
3  {
4      int x = 10, y = 20, z = 30;
5      cout<<"Variabila x = "<<x<<" variabila y = "<<y<<" variabila z = "<<z<<endl;
6  }
7  int main()
8  {
9      var_locale();
10     cout<<"Variabila x = "<<x<<" variabila y = "<<y<<" variabila z = "<<z<<endl;
11 }
12
```

The `Build` pane at the bottom shows the compilation output for `curs11_1 - Debug`. The output indicates that the source file `curs11_1.cpp` was compiled, but it resulted in 4 errors and 1 warning. The errors are related to undeclared variables `x`, `y`, and `z` in the `main` function. These error messages are circled in red:

```
-----Configuration: curs11_1 - Debug-----
Compiling source file(s)...
curs11_1.cpp
In file included from C:\MinGWStudio\MinGW\bin\..\lib\gcc\mingw32\3.4.2\..\..\..\include\c++\3.4.2\backward\ic
from curs11_1.cpp:1:
C:\MinGWStudio\MinGW\bin\..\lib\gcc\mingw32\3.4.2\..\..\..\include\c++\3.4.2\backward\backward_warning.h:32:2:
curs11_1.cpp: In function 'int main()':
curs11_1.cpp:10: error: 'x' undeclared (first use this function)
curs11_1.cpp:10: error: (Each undeclared identifier is reported only once for each function it appears in.)
curs11_1.cpp:10: error: 'y' undeclared (first use this function)
curs11_1.cpp:10: error: 'z' undeclared (first use this function)

curs11_1.exe - 4 error(s), 1 warning(s)
```

10.5. Variabile locale și variabile globale

2. Variabilele globale sunt *acele variabile care se declară înaintea oricăror declarații de funcții, iar numele, valorile și existența lor este recunoscută în întregul program.*

Exemplu:

Următorul program conține două funcții, una numită *var_globale()* și cea de-a doua funcția principală *main()* și trei variabile x, y și z:

10.5. Variabile locale și variabile globale

```
#include <iostream.h>
int x = 10, y = 20, z = 30;
void var_globale(void)
{
    cout<<"\nVariabila x = "<<x<<" variabila y = "<<y<<"variabila z
    = " <<z;
}
int main()
{
    var_globale();
    cout<<"\nVariabila x = "<<x<<" variabila y = "<<y<<" variabila z
    = "<<z;
}
```

10.5. Variabile locale și variabile globale

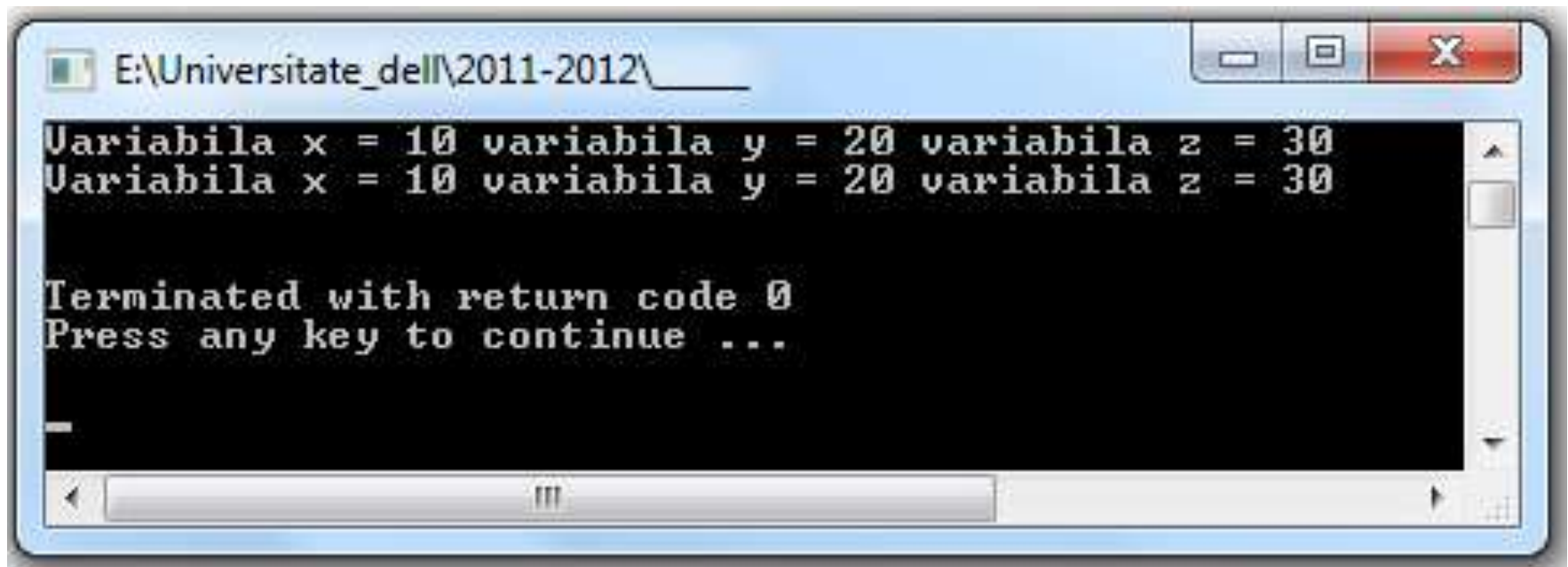
După compilarea și execuția acestui program, ambele funcții, *var_globale()* și *main()*, vor afișa pe ecran valorile variabilelor globale.

Observație:

Declararea variabilelor s-a făcut în afara funcțiilor.

Atunci când se declară ca variabilă globală, toate funcțiile programului pot folosi și modifica valorile acelei variabile prin simpla referire la numele său.

10.5. Variabile locale și variabile globale



A screenshot of a Windows command prompt window. The title bar shows the path "E:\Universitate_dell\2011-2012\". The command prompt displays two lines of text: "Variabila x = 10 variabila y = 20 variabila z = 30" and "Variabila x = 10 variabila y = 20 variabila z = 30". Below these lines, it says "Terminated with return code 0" and "Press any key to continue ...". The window has a standard Windows XP-style interface with a blue title bar and a scroll bar on the right.

```
E:\Universitate_dell\2011-2012\  
Variabila x = 10 variabila y = 20 variabila z = 30  
Variabila x = 10 variabila y = 20 variabila z = 30  
  
Terminated with return code 0  
Press any key to continue ...  
_
```

10.5. Variabile locale și variabile globale

Chiar dacă variabilele globale par convenabile la prima vedere, ele nu sunt recomandate de obicei.

Astfel, dacă se folosesc variabile globale se observă că nu mai trebuie utilizați parametri în cadrul funcțiilor și deci nu mai trebuie să se înțeleagă mecanismul *apelului prin valoare* și *apelului prin referință*.

10.5. Variabile locale și variabile globale

- Totuși în loc să reducă numărul de erori, folosirea variabilelor globale măresc numărul lor.
- Deoarece în sursa unui program se poate modifica valoarea unei variabile globale în orice loc al programului, este foarte dificil pentru un alt programator să găsească fiecare loc din program în care variabila respectivă se utilizează.

10.5. Variabile locale și variabile globale

- Astfel, alți programatori pot modifica programul dar fără ca să aibă vreun control asupra efectelor acestor modificări asupra variabilelor declarate global.
- Este o **regulă generală** ca *orice modificare a unei variabile să se reflecte doar asupra funcției care o folosește*.
- De aici **este recomandabil** ca orice program în C/C++ să aibă numai variabile locale și eventual doar câteva variabile globale (cât mai puține).

10.5. Variabile locale și variabile globale

Prezentăm în continuare un exemplu de program care evidențiază efectele folosirii numelor de variabile identice atât în funcții, cât și în funcția principală:

```
#include <iostream.h>
int a = 10, b = 20, c = 30;           // variabile globale
void valoarea_lui_a(void)
{
    int a = 100;
    cout<<"variabila a contine "<<a<<"variabila b contine
"<<b<<"variabila c contine "<<c<<"\n";
}
```

10.5. Variabile locale și variabile globale

```
int main()
{
    valoarea_lui_a();
    cout<<"variabila a contine "<< a <<"variabila b
    contine "<< b <<"variabila c contine "
    <<c<<"\n";
}
```

10.5. Variabile locale și variabile globale



```
E:\Universitate_del\2011-2012\__  
variabila a contine 100 variabila b contine 20 variabila c contine 30  
variabila a contine 10 variabila b contine 20 variabila c contine 30  
  
Terminated with return code 0  
Press any key to continue ...
```

Se observă că numele variabilei globale intră în conflict cu cel al variabilei locale și atunci *compilatorul C++, va folosi întotdeauna variabila locală.*

Deci, în funcție, se va afișa valoarea modificată a variabilei a și nu cea inițială, care a fost declarată global la începutul programului.

Capitolul 10. Funcții

10.1. Declararea funcției

10.2. Apelul funcției

10.3. Prototipul funcției

10.4. Parametri formali și actuali

10.5. Variabile locale și variabile globale

10.6. Variabile statice și variabile automate

10.7. Funcții matematice

10.6. Variabile statice și variabile automate

Din punctul de vedere al locului în care sunt alocate variabilele și al locului alocării avem:

1. **variabile automate (auto)**

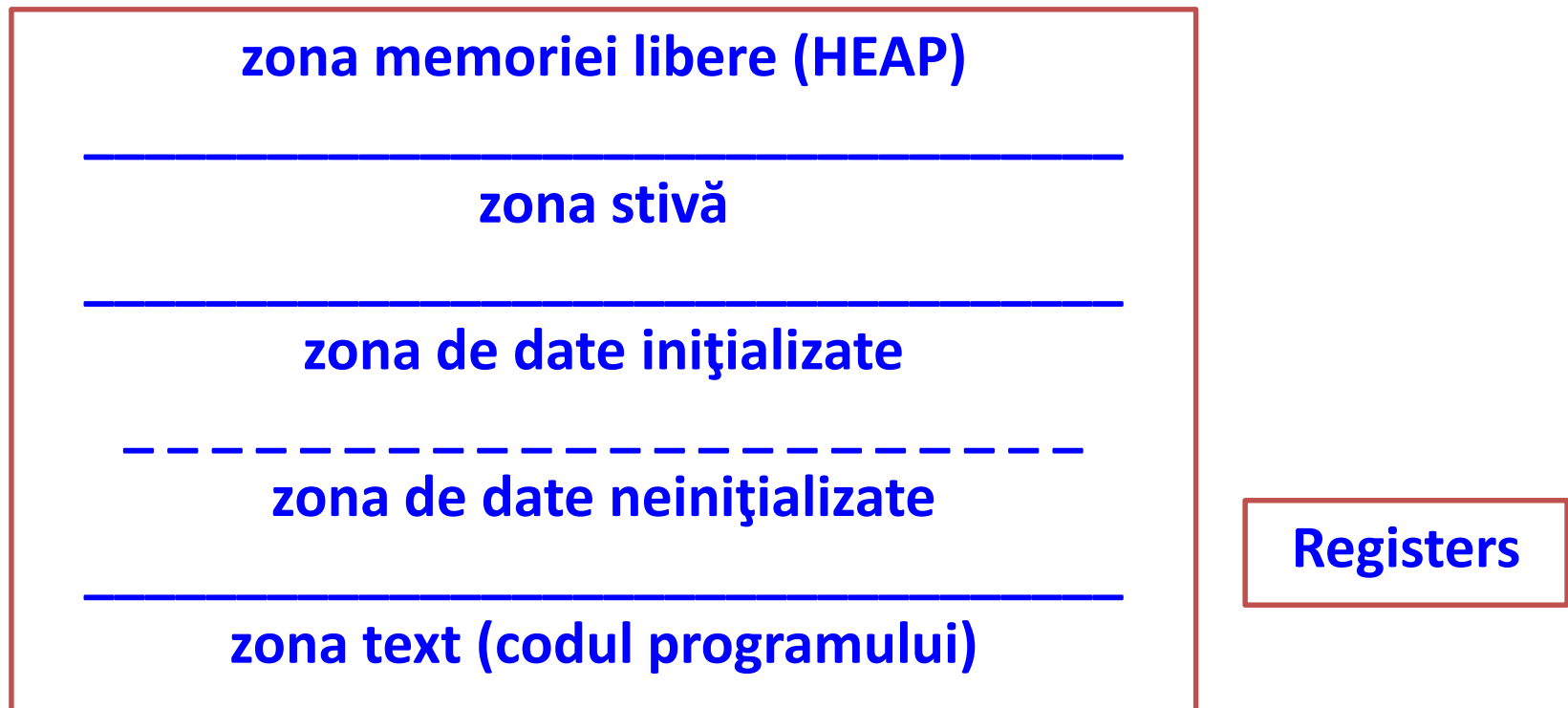
2. **variabile statice**

10.6. Variabile statice și variabile automate

1. **Variabilele automate** se alocă în regiștrii de stare sau pe stivă și **sunt disponibile numai în locul în care s-a făcut alocarea (ele se alocă la execuție)**
2. **Variabilele statice** se alocă în zona de date a programului la încărcarea programului în memorie. Ele **sunt disponibile pe toată durata de existență a programului în memorie.**

10.6. Variabile statice și variabile automate

- Prezentăm în continuare *harta simplificată a memoriei la încărcarea programului*:

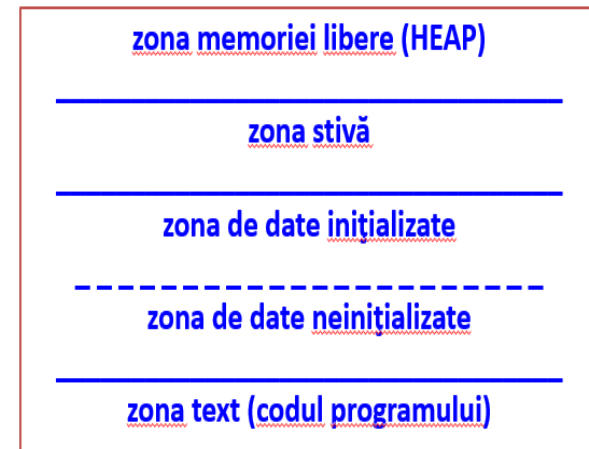


10.6. Variabile statice și variabile automate

- Acum putem specifica pentru fiecare zonă, variabilele și parametrii care se alocă acolo:

În zona stivă se alocă:

- Variabile locale automate
- Parametrii funcțiilor
- Adresa de retur a funcțiilor
- Variabilele temporare necesare evaluării expresiilor

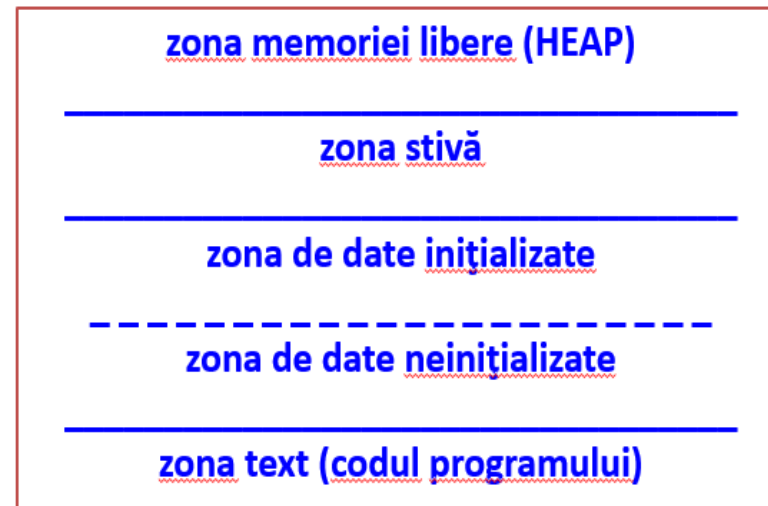


Registers

10.6. Variabile statice și variabile automate

În zona de date se alocă:

- variabilele globale
- șiruri inițializate și constante
- variabilele locale statice

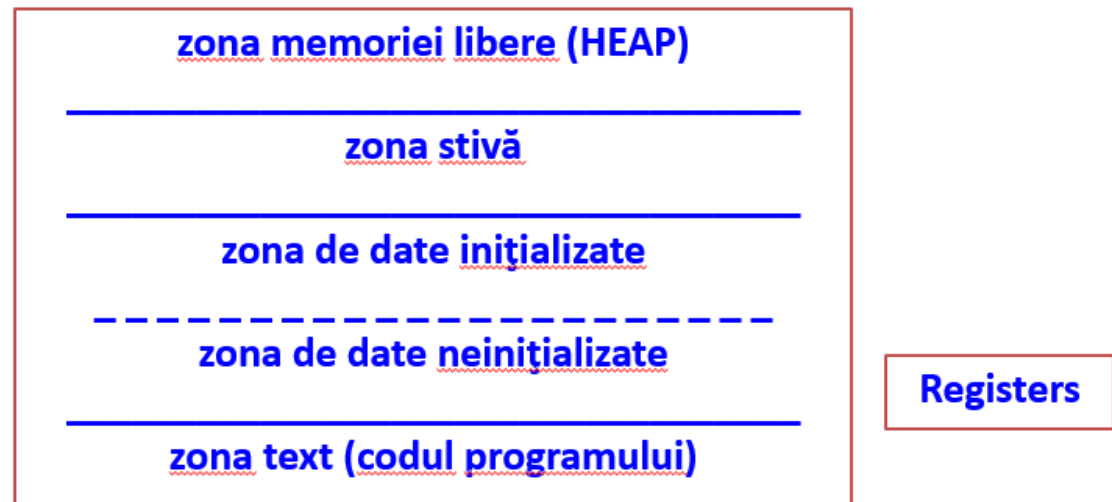


Registers

10.6. Variabile statice și variabile automate

În zona registers se alocă:

- variabilele locale automate
- parametri de apel ai funcțiilor



10.6. Variabile statice și variabile automate

- În limbajul C++, variabilele care se declară în cadrul funcției sunt adesea numite și **automatice**, deoarece *compilatorul C/C++ le creează automat când începe execuția funcției și apoi le distruge când ea se încheie*.
- Această caracteristică a variabilelor se explică prin faptul că variabilele funcțiilor sunt păstrate de compilator temporar în stivă.

10.6. Variabile statice și variabile automate

- Ca urmare, *funcția atribuie o valoare unei variabile în timpul unei apelări, dar variabila pierde valorile după ce funcția se încheie.*
- La următoarea apelare a funcției, valoarea variabilei este din nou nedefinită.
- În funcție de procesele executate de funcția respectivă, *este posibil ca variabilele funcției să memoreze ultima valoare care le-a fost atribuită în cadrul funcției.*

10.6. Variabile statice și variabile automate

Exemplu:

Prezentăm în continuare o funcție care afișează numărul matricol pentru fiecare student dintr-o facultate.

Funcția *afișează_matricol()*, folosește o **variabilă statică** *id_student* care păstrează numărul de identificare al studentului pentru care s-a tipărit ultima foaie matricolă.

În acest fel, fără nici o altă mențiune funcția va începe să tipărească foaia matricolă a următorului student:

10.6. Variabile statice și variabile automate

```
void afiseaza_matricol(int numar_print)
{
    static int id_student;

    // celelalte instrucțiuni
}
```

10.6. Variabile statice și variabile automate

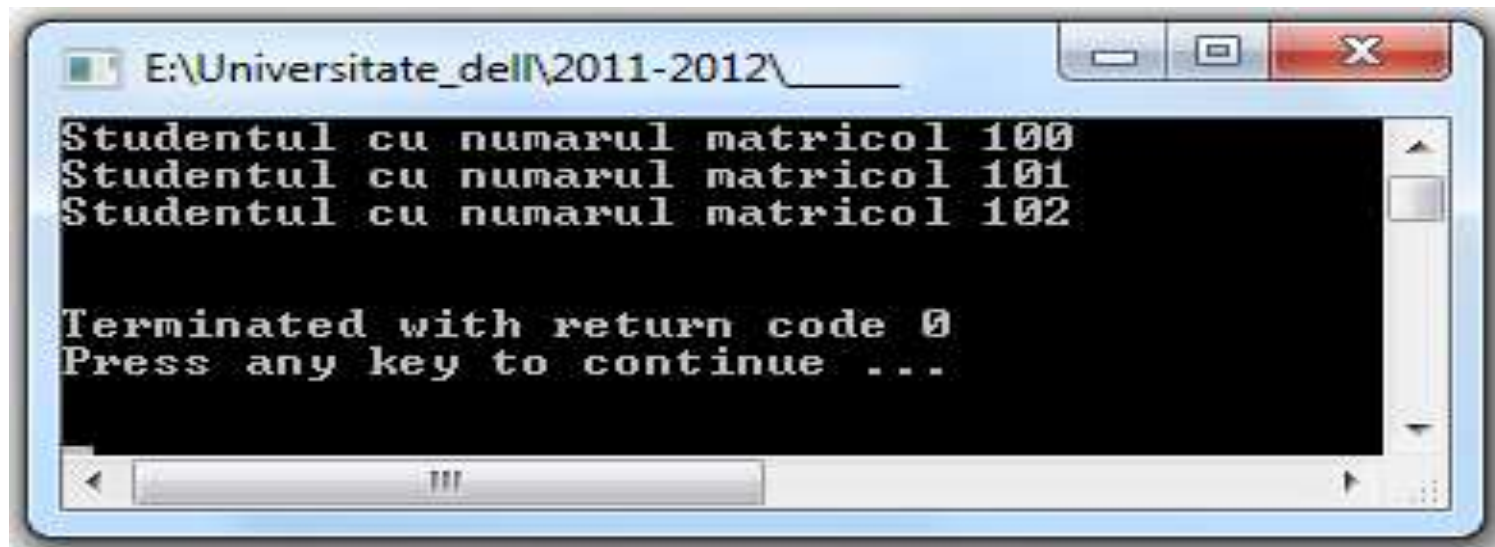
Următorul program ilustrează folosirea unei variabile *static*, astfel încât de fiecare dată când funcția este apelată se va afișa o valoare cu o unitate mai mare decât precedenta datorită folosirii variabilei statice *id_student*:

10.6. Variabile statice și variabile automate

```
#include<iostream.h>
void afiseaza_matricol(int numar_print)
{
    static int id_student = 100;
    cout<<"Studentul cu numarul matricol "<<id_student<<"\n";
    id_student++;
    // celelalte instructiuni
}
int main()
{
    afiseaza_matricol(1);
    afiseaza_matricol(1);
    afiseaza_matricol(1);
}
```

10.6. Variabile statice și variabile automate

După compilarea și execuția programului, pe ecran vor apărea următoarele valori:



```
E:\Universitate_del\2011-2012\  
Studentul cu numarul matricol 100  
Studentul cu numarul matricol 101  
Studentul cu numarul matricol 102  
  
Terminated with return code 0  
Press any key to continue ...
```

Se observă că variabila *id_student* își păstrează valoarea de la o apelare la alta.

Capitolul 10. Funcții

10.1. Declararea funcției

10.2. Apelul funcției

10.3. Prototipul funcției

10.4. Parametri formali și actuali

10.5. Variabile locale și variabile globale

10.6. Variabile statice și variabile automate

10.7. Funcții matematice

10.7. Funcții matematice

În programele în care avem nevoie să efectuăm anumite calcule matematice, putem folosi câteva funcții predefinite ale limbajului C/C++ care efectuează aceste calcule.

Prototipurile acestor funcții se află în fișierul sistem *<math.h>*, care trebuie inclus pentru compilarea programului.

10.7. Funcții matematice

Prezint în continuare câteva exemple de funcții matematice:

Valoarea absolută a unui număr întreg – **abs()**

```
#include<iostream.h>
```

```
#include<math.h>
```

```
int main()
```

```
{
```

```
    cout<<"Valoarea absoluta a lui " <<10<<" este " <<abs(10)<<"\n";
```

```
    cout<<"Valoarea absoluta a lui " <<0<<" este " <<abs(0)<<"\n";
```

```
    cout<<"Valoarea absoluta a lui " <<-10<<" este " <<abs(-10)<<"\n";
```

```
}
```


10.7. Funcții matematice

După compilarea și execuția programului se obțin următoarele valori:

Valoarea absolută a lui 10 este 10

Valoarea absolută a lui 0 este 0

Valoarea absolută a lui -10 este 10

10.7. Funcții matematice

Rotunjirea unei valori reale în virgulă mobilă –
ceil() și *floor()*

Funcția ***ceil()*** se folosește atunci când se dorește să se rotunjească valoarea unei variabile sau a unei expresii, înlocuind-o cu valoarea întreagă imediat următoare.

Funcția ***floor()*** se folosește atunci când se dorește să se rotunjească valoarea unei variabile sau a unei expresii, înlocuind-o cu valoarea întreagă imediat anterioară.

10.7. Funcții matematice

```
#include <iostream.h>
#include <math.h>
int main()
{
    cout<<"Valoarea "<<1.9<<" rotunjita cu functia ceil()
    este"<<ceil(1.9)<<"\n";
    cout<<"Valoarea "<<2.1<<" rotunjita cu functia ceil()
    este"<<ceil(2.1)<<"\n";

    cout<<"Valoarea "<<1.9<<" rotunjita cu functia floor()
    este"<<floor(1.9)<<"\n";
    cout<<"Valoarea "<<2.1<<" rotunjita cu functia floor()
    este"<<floor(2.1)<<"\n";
```

10.7. Funcții matematice

După compilarea și execuția programului se obțin următoarele valori:

Valoarea 1.900000 rotunjită cu funcția `ceil()` este 2.000000

Valoarea 2.100000 rotunjită cu funcția `ceil()` este 3.000000

Valoarea 1.900000 rotunjită cu funcția `floor()` este 1.000000

Valoarea 2.100000 rotunjită cu funcția `floor()` este 2.000000

10.7. Funcții matematice

Funcții trigonometrice:

1. sinus – `sin()`
2. cosinus – `cos()`
3. sinusul hiperbolic – `sinh()`
4. cosinusul hiperbolic – `cosh()`
5. tangenta – `tan()`
6. tangenta hiperbolică – `tanh()`
7. arcsinus – `asin()`
8. arccosinus – `acos()`
9. arctangenta – `atan()`

10.7. Funcții matematice

Exemplu:

Într-un triunghi dreptunghic, sinusul unui unghi este raportul între latura opusă și ipotenuză.

Pentru a folosi în programele C/C++, determinarea sinusului unui unghi, se poate utiliza funcția ***sin()***, care returnează o valoare de tip double ce reprezintă sinusul unui unghi specificat în radiani.

La fel, se pot calcula cosinusul unui unghi și tangenta unui unghi:

10.7. Funcții matematice

```
#include <iostream.h>
#include <math.h>
#define pi 3.14159265
int main()
{
    double radian;
    for(radian=0.0; radian<3.1; radian+=0.1)
        cout<< " Sinus de " <<radian<< " este " <<sin(radian)<<"\n";
    cout<<"Cosinus de pi/2 este " <<cos(pi/2.0)<<"\n";
    cout<<"Cosinus de pi este " <<cos(pi)<<"\n";
    cout<<"Tangenta de pi este " <<tan(pi)<<"\n";
    cout<<"Tangenta de pi/4 este " <<tan(pi/4.0)<<"\n";
```

10.7. Funcții matematice

```
cout<<"\n Arcsinus "<<"\n";
for(radian=-0.5; radian<=0.5; radian+=0.2)
    cout<<" "<<radian<<" "<<asin(radian);
cout<<"\n Arccosinus "<<"\n";
for(radian=-0.5; radian<=0.5; radian+=0.2)
    cout<<" "<<radian<<" "<<acos(radian);
cout<<"\n Arctangenta "<<"\n";
for(radian=-0.5; radian<=0.5; radian+=0.2)
    cout<<" "<<radian<<" "<<atan(radian);
}
```


10.7. Funcții matematice

Funcția exponențială - e^x

Pentru a calcula e^x trebuie să folosim funcția **exp()**:

```
#include <iostream.h>
#include <math.h>
int main()
{
    double valoare;
    for(valoare=0.0; valoare<=1.0; valoare+=0.1)
        cout<<"exp("<<valoare<<") este "<<exp(valoare);
}
```

10.7. Funcții matematice

După compilarea și execuția programului se obțin următoarele valori:

- `exp(0.000000)` este 1.000000
- `exp(0.100000)` este 1.105171
- `exp(0.200000)` este 1.221403
- `exp(0.300000)` este 1.349859
- `exp(0.400000)` este 1.491825
- `exp(0.500000)` este 1.648721
- `exp(0.600000)` este 1.822119
- `exp(0.700000)` este 2.013753
- `exp(0.800000)` este 2.225541
- `exp(0.900000)` este 2.459603
- `exp(1.000000)` este 2.718282

10.7. Funcții matematice

Restul împărțirii unui real la un număr real
fmod()

```
#include <iostream.h>
#include <math.h>
int main()
{
    double numarator=10.0, numitor=3.0;
    cout<<"fmod(10,3) este "<< fmod(numarator,
    numitor));
}
```

10.7. Funcții matematice

După compilarea și execuția programului se obțin următoarele valori:

fmod(10,3)=1.000000

10.7. Funcții matematice

Calculul părții întregi și fracționare dintr-un număr real – **modf()**:

```
#include <iostream.h>
```

```
#include <math.h>
```

```
int main()
```

```
{
```

```
    double valoare=1.2345;
```

```
    double parte_intreaga, fract;
```

```
    fract=modf(valoare, &parte_intreaga);
```

```
    cout<<"Valoarea "<<valoare<<" are partea intreaga egala cu
```

```
        "<<parte_intreaga<<" si partea fractionara egala cu
```

```
        "<<fract<<"\n";
```

```
}
```

Partea fracționară a unui număr
real - **modf**

După compilarea și execuția programului se obțin următoarele valori:

Valoarea 1.234500 are partea intreaga egala cu 1.000000 si partea

10.7. Funcții matematice

Calculul lui x^n – *pow()*

Ridicarea unei valori x la o putere dată n se poate efectua folosind funcția *pow()*, care utilizează o valoare de tip *double* și returnează o valoare de tip *double*:

```
#include<iostream.h>
```

```
#include<math.h>
```

```
int main()
```

```
{
```

```
    int putere;
```

```
    for(putere=-2; putere <= 2; putere++)
```

```
        cout<<"10 ridicat la puterea "<<putere<<" este " <<pow(10.0, putere);
```

```
}
```

pow() – calculul lui x^n

După compilarea și execuția programului se obțin următoarele valori:

10 ridicat la puterea -2 este 0.010000

10 ridicat la puterea -1 este 0.100000

10 ridicat la puterea 0 este 1.000000

10 ridicat la puterea 1 este 10.000000

10 ridicat la puterea 2 este 100.000000

10.7. Funcții matematice

Calculul rădăcinii pătrate a unei valori *sqrt()*

Pentru a calcula rădăcina pătrată dintr-o valoare de tip *double*, putem utiliza funcția ***sqrt()***:

```
#include <iostream.h>
#include <math.h>
int main()
{
    double valoare;
    for(valoare=0.0; valoare<10.0; valoare+=0.1)
        cout<<"Valoarea "<<valoare<<" are radacina patrata
            "<<sqrt(valoare);
}
```

Pentru alte informații teoretice și aplicative legate de acest capitol se recomandă următoarele referințe bibliografice:

1. Adrian Runceanu, Mihaela Runceanu, ***Noțiuni de programare în limbajul C++***, Editura Academica Brâncuși, Târgu-Jiu, 2012 (www.utgjiu.ro/editura)
2. Adrian Runceanu, **Programarea și utilizarea calculatoarelor**, Editura Academica Brâncuși, Târgu-Jiu, 2003 (www.utgjiu.ro/editura)
3. Octavian Dogaru, **C++ - teorie și practică**, volumul I, Editura Mirton, Timișoara, 2004 (www.utgjiu.ro/editura)

Întrebări?