

2.1. Constante

Constantele (constante literale) sunt valori fixe (numerice, caractere sau șiruri de caractere), care nu pot fi modificate în timpul execuției programului. Tipul constantei este determinat de compilator pe baza valorii și sintaxei utilizate, putând avea oricare dintre tipurile predefinite de date. Ele rămân în memorie pe toată durata execuției programului.

2.1.1. Constante întregi

Tipul constantei este determinat pe baza valorii. Astfel, de exemplu:

- dacă valoarea numerică a constantei se încadrează în domeniul de valori al tipului `int`, atunci tipul implicit al constantei este **`int`** (chiar dacă valoarea ar putea fi reprezentată folosind tipul `char`, ca de exemplu 15);
- dacă valoarea depășește domeniul tipului `int` dar se încadrează în domeniul de valori al tipului `long`, reprezentarea va fi ca dată **`long`** (de exemplu valoarea 99999 este reprezentată ca dată de tip `long`);
- dacă valoarea constantei depășește domeniul de valori al tipului `long`, chiar dacă nu conține parte zecimală, ea va fi reprezentată în mod implicit ca o dată de tip `double` (de exemplu, constanta 98765432100).

Se poate forța preluarea constantei cu alt tip decât cel implicit prin utilizarea unui sufix (`U` sau `u` pentru `unsigned`, respectiv `L` sau `l` pentru `long`).

Constantele întregi pot fi **zecimale** (cu reprezentare în baza 10), **octale** (cu reprezentare în baza 8) sau **hexazecimale** (cu reprezentare în baza 16).

▪ Constante zecimale (baza 10)

Constantele zecimale se disting prin faptul că prima cifră este diferită de 0.



Exemple

23	// tip <code>int</code>
-555L	// tip <code>long int</code>
23u	// tip <code>unsigned int</code>
3276L	// tip <code>long int</code>
32768	// tip <code>int</code> sau <code>long int</code> , funcție de compilatorul folosit
32768U	// tip <code>unsigned int</code>
77UL	// tip <code>unsigned long int</code>

▪ Constante octale (baza 8)

Constantele octale sunt valori întregi cu reprezentare în baza 8. Convenția de scriere este ca prima cifră să fie 0 (zero). Fiind vorba de reprezentare în baza 8, cifrele 8 și 9 sunt ilegale.



Exemple

-067	// tip <code>int</code>
067u	// tip <code>unsigned int</code>

02000000000	// tip long int
055UL	// tip unsigned long
089	// eroare, prima cifră indică reprezentarea în octal
	// și numărul include cifrele 8 și 9

▪ Constante hexazecimale (baza 16)

Constantele hexazecimale, cu reprezentare în baza 16, se disting prin prefixul 0x sau 0X. Pot conține cifre mai mari de 9 (a...f, sau A...F).



Exemple

0x7FFF	// tip int
0X8A0L	// tip long int
0xffu	// tip unsigned int
0x1000L	// tip long int
0xFFul	// tip unsigned long int

2.1.2. Constante în virgulă mobilă

Constantele în virgulă mobilă sunt valori reale a căror reprezentare conține în general următoarele câmpuri:

- parte întreagă
- punct zecimal
- parte fracționară
- e sau E și un exponent cu semn (opțional)
- sufix de specificare a tipului: f sau F (forțează tipul float) sau l sau L (forțează tipul long double).

Se pot omite partea întreagă sau partea fracționară (dar nu amândouă), punctul zecimal sau litera e (E) și exponentul (dar nu amândouă).

Tipul implicit pentru constantele în virgulă mobilă este tipul double. Se poate forța reprezentarea ca tip float, dacă valoarea se încadrează în domeniul de valori corespunzător, prin atașarea sufixului f sau F.



Exemple

2.1	// valoare 2,1 (tip double)
11.22E5	// valoare $11,22 \times 10^5$ (tip double)
-.33e-2	// valoare $-0,33 \times 10^{-2}$ (tip double)
.5	// valoare 0,5 (tip double)
1.	// valoare 1 (tip double)
1.f	// valoare 1 (tip float)
0.L	// valoare 0 (tip long double)

2.1.3. Constante caracter

Constantele caracter sunt reprezentate de unul sau mai multe caractere încadrate de apostrofuri, de exemplu: 'a', 'A', '\n' și ele sunt de tip char.

Pentru a specifica o serie de caractere neafișabile, delimitatorii ('), ("), caracterul (\), etc. se utilizează așa numitele **secvențele escape** (secvențe de evitare) (vezi Tabelul nr. 2.1.) care sunt formate din caracterul backslash (\) însoțit de o literă sau o constantă numerică octală sau

hexazecimală. Valoarea numerică corespunde valorii din codul ASCII a caracterului reprezentat.

Tabelul nr. 2.1. Secvențe escape

Secvență	Valoare (hexa)	Caracter	Descriere
\a	0x07	alarm (bell)	semnal sonor
\b	0x08	BS	backspace
\t	0x09	TAB	tab orizontal
\n	0x0A	LF	linefeed
\v	0x0B	VT	tab vertical
\f	0x0C	FF	formfeed
\r	0x0D	CR	carriage return
\"	0x22	"	ghilimele
\'	0x27	'	apostrof
\?	0x3F	?	semnul întrebării
\\	0x5C	\	backslash
\o	-	orice caracter	șir de cifre octale
\xH	-	orice caracter	șir de cifre hexazecimale



Exemplu de utilizare a constantelor caracter

```
#include <stdio.h>
void main()
{
    putchar('?');           // se afișează caracterul '?'
    putchar(63);           // se afișează caracterul '?', 63 fiind valoarea
                           // corespunzătoare în codul ASCII
    printf("\n%c", '\077'); // se afișează caracterul '?'
    printf("\n%c", '\x3F'); // se afișează caracterul '?'
}
```

2.1.4. Constante șir de caractere

Constantele șiruri de caractere sunt alcătuite dintr-un număr oarecare de caractere, încadrate între ghilimele.

Șirurile de caractere se memorează în tablouri de tipul char, cu dimensiune egală cu numărul de caractere cuprinse între ghilimele, la care se adaugă terminatorul de șir '\0'.

De exemplu, șirul constant "STRING" este alocat în memorie ca în Fig. 2.1.

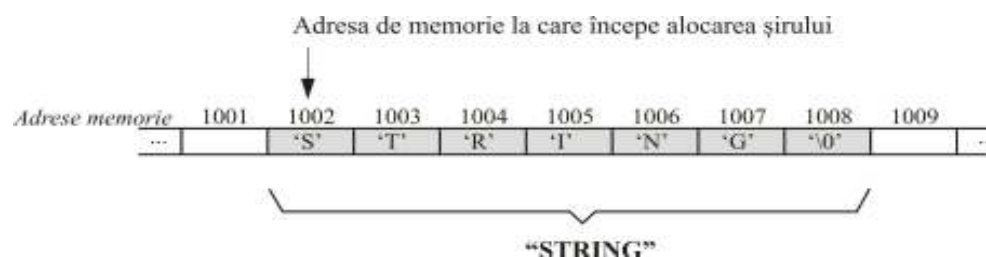


Fig. 2.1. Alocarea în memorie a unui șir de caractere constant

Caracterele care alcătuiesc șirul pot fi secvențe escape:

```
|| "\tMesaj 1\n\tMesaj 2"
```

Șirurile constante adiacente se concatenează și formează un singur șir:

```
|| "Programare" "orientata pe"  
|| "obiecte"
```

Pentru scrierea șirurilor lungi se poate utiliza simbolul (\) care semnalează continuarea șirului pe rândul următor:

```
|| "Exemplu de sir \  
|| scris pe doua rânduri"
```

2.2. Variabile

2.2.1. Declarații de variabile

Variabila este un identificator asociat cu o locație de memorie în care se memorează valori ce pot fi modificate pe parcursul existenței ei.

Toate variabilele trebuie declarate înainte de a fi folosite.

Declarația unei variabile (obiect) precizează tipul datei și numele (identificatorul) cu care va fi referită, sintaxa fiind:

tip_variabila nume_variabila;

unde:

- *tip_variabila* este un specificator de tip de date oarecare, standard (char, int, etc.), pointer sau definit de utilizator.

- *nume_variabila* este orice identificator care nu a primit altă semnificație în domeniul de existență al variabilei.

De exemplu, se poate declara o variabilă de tip int cu numele variabila_mea:

```
|| int variabila_mea;
```

alocarea ei în memorie fiind reprezentată în Fig. 2.2.

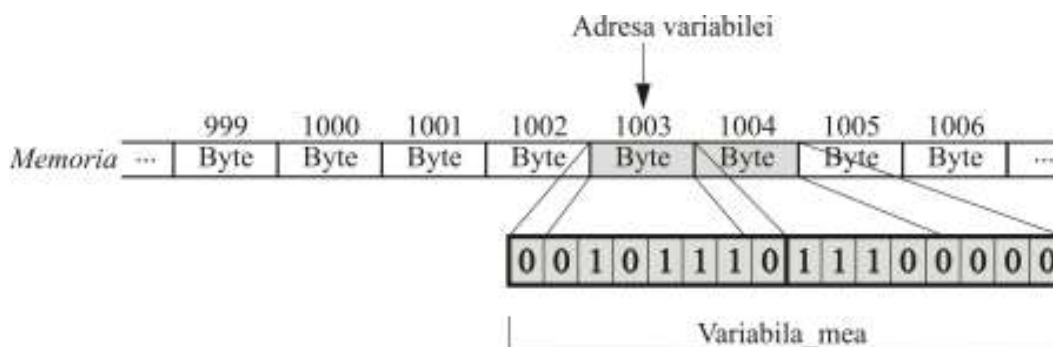


Fig. 2.2. Alocarea unei variabile în memorie

Alte exemple de declarare de variabile:

```
|| float r; // declararea variabilei r de tip float,  
|| unsigned int n; // declararea variabilei n de tip unsigned int
```

Se pot face declarații de variabile cu inițializare utilizând sintaxa:

tip_var nume_var= valoare_iniciala;

sau se pot declara mai multe variabile de același tip utilizând sintaxa:

tip_var nume_var1<=val_iniciala1>,< nume_var2<= val_iniciala2>>,...;

Exemple de declare a variabilelor:

```
int variabila_me=12000;    // declararea unei variabile de tip int inițializată cu
                           // valoarea 12000 (vezi Fig. 2.1.)
double real=2.5;          // declararea variabilei în virgulă mobilă de tip double,
                           // inițializată cu valoarea 2.5
char c1, c2='a', ch;       // declararea a trei variabile de tip char, c1, c2 și ch, variabila c2
                           // fiind inițializată cu valoarea 'a'
```

Declarația unei variabile are ca efect, la execuția programului, rezervarea unui număr de octeți în memorie necesar reprezentării tipului declarat al variabilei. Dacă declarația este cu inițializare, în acea locație se va înscrie valoarea specificată.



Observații

- Adresa de memorie la care se alocă o variabilă nu poate fi precizată de utilizator, ci compilatorul face alocarea în funcție de memoria disponibilă.
- Utilizatorul poate afla adresa la care a fost alocată o variabilă cu ajutorul operatorului & aplicat numelui variabilei. De exemplu: **&variabila_me**
- Chiar dacă nu poate impune adresa la care se alocă o variabilă, prin locul în program în care se face declarația sau specificatorii folosiți, utilizatorul poate stabili zona de memorie în care se face alocarea.

2.2.2. Atributele variabilelor

Atributele unei variabile sunt:

- **tipul datei** poate fi tip fundamental sau definit de utilizator și determină structura, gama de valori, dimensiunea spațiului ocupat în memorie;
- **clasa de memorare** stabilește zona de memorie în care se va plasa informația asociată identificatorului (segment de date, registru, stivă, heap) și delimitează durata sa de alocare;
- **domeniul** reprezintă porțiunea de program în care poate fi accesată informația asociată identificatorului, el fiind determinat de poziția declarației;
- **durata de viață** a identificatorului reprezintă perioada cât există efectiv în memorie și este corelată cu clasa de memorie;
- **legătura** precizează modul de asociere a unui identificator cu un anumit obiect sau funcție, în procesul de editare a legăturilor.

Atributele se pot asocia identificatorilor în mod implicit, în funcție de poziția și sintaxa declarației, sau explicit prin utilizarea unor specificatori.

Poziția declarației determină cele două domenii de existență fundamentale:

- **Domeniul bloc (local)** - Identificatorii cu domeniu bloc se numesc locali și sunt rezultatul unor declarații în interiorul unui bloc (au domeniul cuprins între declarație și sfârșitul blocului) sau sunt parametri formali din definiția unei funcții (au ca domeniu blocul funcției).
- **Domeniul fișier (global)** - Identificatorii cu domeniu fișier se numesc globali și sunt declarați în afara oricăror funcții (domeniul este cuprins între declarație și sfârșitul fișierului).

```
#include <stdio.h>

// zona declarațiilor globale
float functie (int, float);    // prototipul funcției
int a;                        // se declară variabila cu numele a, căreia i se rezervă o zonă de
                               // memorie de 2/4 octeți (16/32 biți) localizată în segmentul de
                               // date, deoarece declarația se află în zona declarațiilor globale

// definiții de funcții
void main (void)
{
    unsigned char c; // se declară variabila automatică cu numele c căreia i se rezervă
                     // un octet în segmentul de stivă, variabila fiind o variabilă locală
    float r=2.5;      // se declară variabila automatică cu numele r căreia i se rezervă 4 octeți în
                     // segmentul de stivă, variabila fiind o variabilă locală care este inițializată cu
//...                // valoarea 2.5
}
float functie (int n, float q) // se declară variabilele locale n și q, cărora li se alocă 2,
{
    // respectiv 4 octeți în segmentul de stivă
    a=100;           // variabila a este declarată global, deci poate fi utilizată de orice funcție
                     // definită în fișier
    r=1.5 *a;        // eroare, variabila r nu poate fi folosită în funcție(), ea fiind declarată local
                     // funcției main(), deci nu poate fi referită decât în aceasta
//...
}
```



Observații

- Într-un bloc inclus în domeniul unei declarații este permisă o declarație locală a aceluiași identificador, asocierea fiind însă făcută unui alt obiect cu alocarea altei zone de memorie.
- Spre deosebire de C, care impune gruparea declarațiilor locale la începutul unui bloc, C++ permite plasarea declarațiilor în interiorul blocului, bineînțeles înainte de utilizarea obiectului

Clasa de memorare se poate preciza prin specificatorii **auto**, **static**, **extern**, **register**. Acești specificatori precizează modul de alocare a memoriei și timpul de viață pentru variabile și legătură pentru funcții și variabile.

auto

Declarația **auto** specifică o variabilă automatică și se poate utiliza pentru variabile cu domeniul local (cu spațiu alocat pe stivă). Variabilele ce se declară în interiorul unui bloc sunt implicit automate, astfel că declarația auto este rar folosită.

```

void fct()
{   auto float r;           // declararea unei variabile automate cu declarare explicită
    double s;              // declararea unei variabile implicit automatice
    ...
}

```

static

Declarația **static** a unei variabile locale forțează durata statică fără a modifica domeniul de utilizare. Variabilele statice își păstrează valoarea între două apeluri succesive ale blocurilor care le conțin, asigurându-se în același timp o protecție, dar ele nu pot să fie accesate din blocuri care nu constituie domeniul lor de existență. Variabilele statice pot fi declarate cu inițializare, în caz contrar, implicit se inițializează cu valoarea 0, similar variabilelor globale.

```

#include <stdio.h>
int fct()
{   static int a=2;          // se declară o variabilă locală funcției, cu durată statică
    return (a++);
}

void main(void)
{   int n;
    n=fct();
    printf („\n Prima atribuire : n= %d”, n);
    n=fct();
    printf („\n A doua atribuire : n= %d”, n);
}

```

Programul afișează:

```

Prima atribuire : n= 3           // la primul apel al funcției, variabila a are valoarea
                                // inițială 2, ea fiind apoi incrementată
A doua atribuire : n= 4         // la al doilea apel al funcției, variabila a are la început
                                // valoarea 3 (valoare datorată apelului anterior al funcției),
                                // valoarea fiind apoi incrementată

```

register

Declarația **register** are ca efect memorarea variabilei într-un registru al procesorului și nu în memorie, având ca rezultat creșterea vitezei de execuție a programului. Aceste variabilele pot fi variabile locale, nestatice, de tip `int` sau `char`. Numai două variabile pot fi memorate simultan în registre, în cazul existenței mai multor declarații **register**, cele care nu pot fi onorate vor fi tratate de compilator ca variabile obișnuite.

```

register char c;                // declararea variabilei c cu memorare într-un registru al procesorului

```

extern

Specificatorul **extern** indică legătura externă și asigură durata statică pentru variabile locale și globale sau pentru funcții (acestea au implicit legătură externă și durată statică).

Pentru identificatorii cu legătură externă sunt permise mai multe declarații de referință, dar o singură definiție. De exemplu, în cazul unui proiect în care o variabilă se folosește în mai multe fișiere, ea se va defini global într-un fișier, în celelalte fiind declarată extern fără definire.

Compilatorul îi va aloca o singură dată memorie.

Ca exemplu, se consideră programul format din două fișiere sursă Ex_prj.cpp și Ex_prj1.cpp. Variabila m declarată în fișierul Ex_prj1.cpp este folosită în fișierul Ex_prj.cpp prin realizarea legăturii externe pentru această variabilă.

```
// Fișier Ex_prj.cpp
#include <stdio.h>

extern int m;                // se declară legătură externă pentru variabila m

void main(void)
{
    ...
    scanf("%d", &m);
    printf("\nm= %#x", m);
    ...
}
// Fișier Ex_prj1.cpp
...
int m ;                      // declarația variabilei m
...
```

În exemplul anterior, cele două fișiere, Ex_prj.cpp și Ex_prj1.cpp, se includ într-un proiect utilizând opțiunea “project” a meniului principal din mediul de programare C/C++. Variabila m este declarată global într-unul dintre fișiere, ea putând fi referită din celălalt fișier datorită specificării legăturii “extern”.

2.2.3. Modificatori de acces

Modificatorii de acces ce pot fi utilizați sunt **const** și **volatile** și ei pot controla modul de modificare a unei variabile.

const

Variabilele **const** (constante cu nume) nu pot fi modificate pe parcursul execuției unui program. Instrucțiunile care încearcă modificarea variabilelor const generează erori la compilare.

```
const int a=99;              // declarația variabilei a const
a++;                         // eroare, a nu se poate modifica fiind declarat const
a=5;                         // eroare, a nu se poate modifica fiind declarat const
```



Observații

- Variabilele const sunt în mod frecvent folosite atunci când:
 - în program se utilizează valori constante în mod repetat, fiind mai comod să fie precizate prin numele lor;
 - se dorește protejarea parametrilor funcțiilor care au fost transferați prin adresă sau referință.

volatile

Variabilele volatile pot fi modificate din exteriorul programului (de exemplu servirea unei întreruperi).

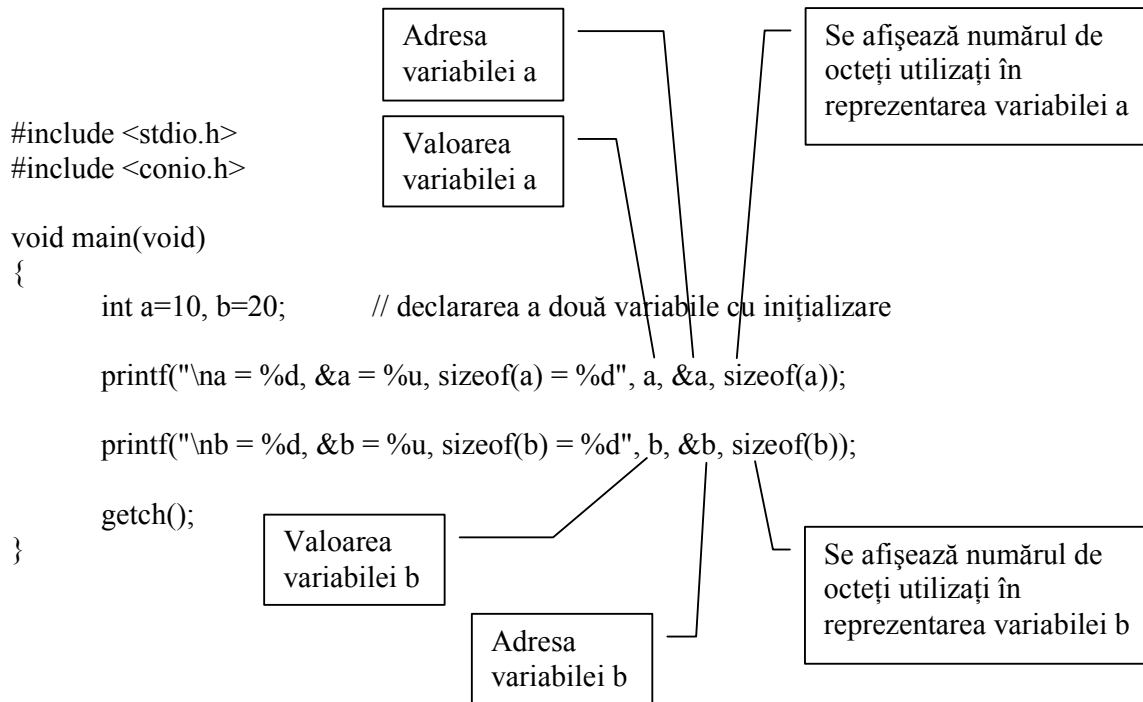


Exemple

Exemplul 2.1. - Se scrie un program în care se afișează informații referitoare la variabile.

Se afișează valoarea, adresa și numărul de octeți ocupați în memorie de către variabile.

*****/



*****/

La execuția programului se afișează:

```
C:\F:\==PROGRAMARE C++ ==\LABORATOR FR\LA...
a = 10, &a = 1310588, sizeof(a) = 4
b = 20, &b = 1310584, sizeof(b) = 4
```

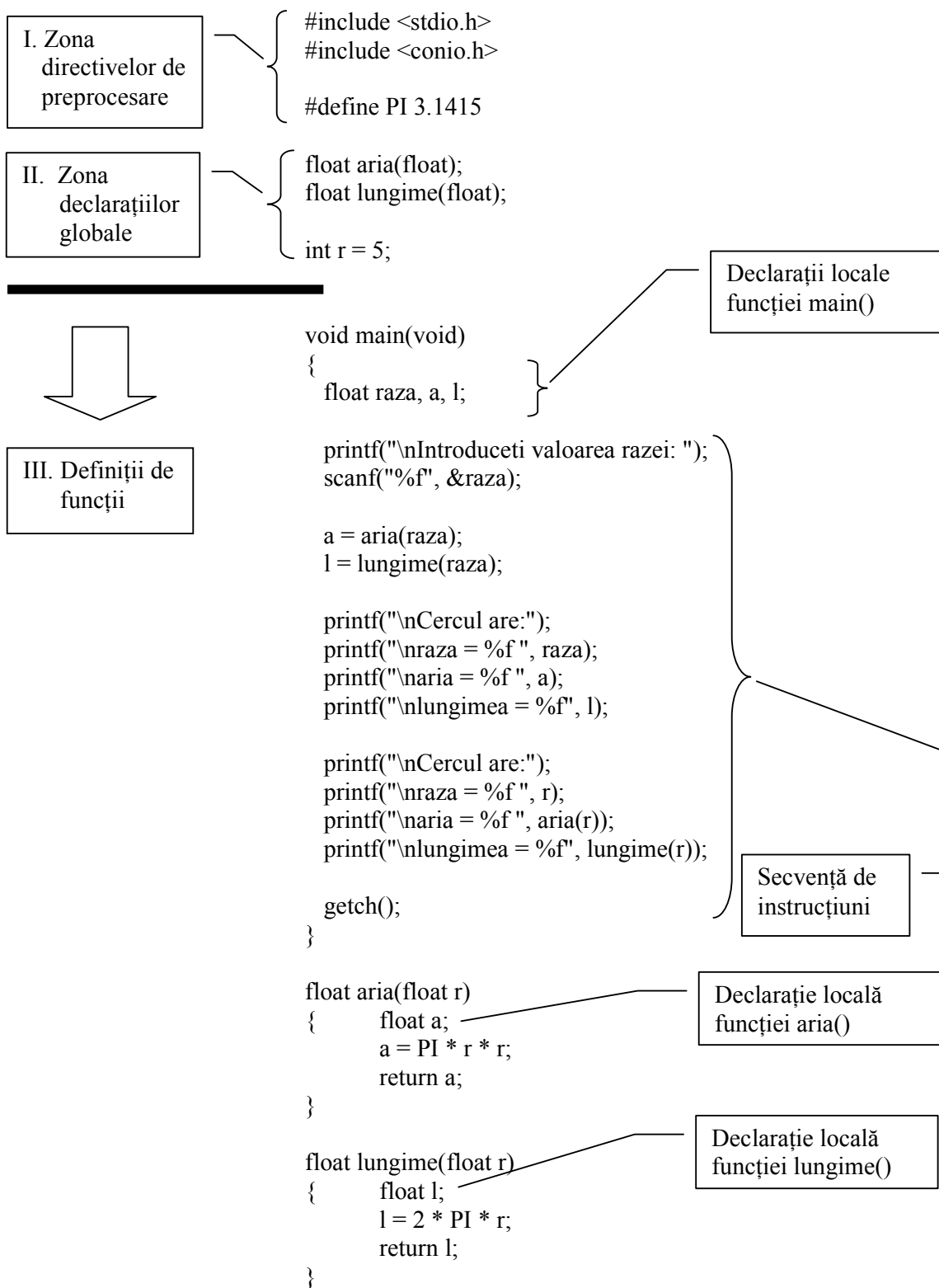
- valoarea afișată pentru variabile este valoarea de inițializare;
- adresa de memorie la care este alocată o variabilă poate fi aflată cu ajutorul operatorului `&` (adresă);
- numărul de octeți utilizați în reprezentarea unei variabile se poate afla cu ajutorul operatorului `sizeof`; numărul de octeți alocați unei variabile este determinat de tipul variabilei (în cazul de față tipul celor două variabile este `int` – se reprezintă pe 4 octeți).

/******

Exemplul 2.2. - Se scrie un program în care :

- se definesc funcții de calcul al ariei și lungimii unui cerc.
- în funcția main se va exemplifica folosirea funcțiilor

*****/



/*****

Exemplul 2.3. - Se scrie un program în care se exemplifică atributele variabilelor :

- clasa de memorare
- domeniu
- durata de viață

*****/

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
int a=5;
```

variabilă globală

- alocată în segmentul de date
- durata de viață - execuția programului

```
void f1(void)
```

```
{  
    printf("\na=%d",a);  
}
```

```
void f2(void)
```

```
{  
    int a=7;
```

variabilă locală

- alocată pe stivă
- durata de viață - execuția funcției
- domeniu: funcția

```
    printf("\na=%d", a);  
}
```

```
void main()
```

```
{  
    int b=10, c=20;
```

variabile locale funcției main()

- alocate pe stivă
- durata de viață - funcția main()
- domeniu: funcția main()

```
    printf("\na=%d, b=%d, c=%d", a, b, c);
```

```
    {  
        int b=100, d=1000;
```

variabile locale blocului de instrucțiuni

- alocate pe stivă
- durata de viață - execuția blocului de instrucțiuni
- domeniu: blocul de instrucțiuni

```
        printf("\na=%d, b=%d, c=%d, d=%d", a, b, c, d);  
    }
```

```
    printf("\na=%d, b=%d, c=%d", a, b, c);
```

```
    printf("\na=%d, b=%d, c=%d, d=%d", a, b, c, d);  //- eroare, variabila d nu mai exista
```

```
    puts("\nSe executa functia f1");  
    f1();
```

```
    puts("\nSe executa functia f2");  
    f2();
```

```
}
```

/******

Exemplul 2.4. - Exemplu de folosire a variabilelor statice

*****/

```
#include <stdio.h>
#include <conio.h>
```

```
void func(void)
{
    int a=10;
```

variabilă locală automatică
- alocată pe stivă
- durata de viață-execuția funcției
- domeniu: funcția

```
    static int b=10;
```

```
    a=a+1;
    b=b+1;
    printf("\na=%d, b=%d", a, b);
}
```

variabilă locală statică
- alocată în segmentul de date
- durata de viață - execuția programului
- domeniu: funcția

```
void main()
{
    func();
    func();
    func();

    getch();
}
```

/******/



Întrebări. Exerciții. Probleme.

1. Folosind Exemplul 2.2., să se scrie un program în care se definesc funcții de calcul al ariei, și respectiv al perimetrului unui dreptunghi. (Observații: Funcțiile vor avea câte doi parametri, corespunzători celor două laturi ale dreptunghiului).
2. Folosind Exemplul 2.2., să se scrie un program în care se definesc funcții de calcul pentru arii laterale și volum pentru diferite corpuri, cum ar fi paralelipiped, con, cilindru, etc.