DATE, OPERATORI ŞI EXPRESII

2.1. LIMBAJELE C ŞI C++

Aşa cum comunicarea dintre două persoane se realizează prin intermediul limbajului natural, comunicarea dintre om şi calculator este mijlocită de un limbaj de programare. Limbajele C şi C++ sunt limbaje de programare de nivel înalt.

Limbajul C a apărut în anii 1970 şi a fost creat de Dennis Ritchie în laboratoarele AT&T Bell. Limbajul C face parte din familia de limbaje concepute pe principiile programării structurate, la care ideea centrală este ”structurează pentru a stăpâni o aplicaţie”. Popularitatea limbajului a crescut rapid datorită eleganţei şi a multiplelor posibilităţi oferite programatorului (puterea şi flexibilitatea unui limbaj de asamblare); ca urmare, au apărut numeroase alte implementări. De aceea, în anii ’80 se impune necesitatea standardizării acestui limbaj. În perioada 1983-1990, un comitet desemnat de ANSI (American National Standards Institute) a elaborat un compilator ANSI C, care permite scrierea unor programe care pot fi portate fără modificări, pe orice sistem.

Limbajul C++ apare la începutul anilor ’80 şi îl are ca autor pe Bjarne Stroustrup. El este o variantă de limbaj C îmbunătăţit, mai riguroasă şi mai puternică, completată cu construcţiile necesare aplicării principiilor programării orientate pe obiecte (POO). Limbajul C++ păstrează toate elementele limbajului C, beneficiind de eficienţa şi flexibilitatea acestuia. Limbajul C++ este un superset al limbajului C. Incompatibilităţile sunt minore, de aceea, modulele C pot fi încorporate în proiecte C++ cu un efort minim.

2.2. PROGRAME ÎN LIMBAJUL C/C++

Un *program* scris în limbajul C (sau C++) este compus din unul sau mai multe *fişiere sursă*. Un fişier sursă este un fişier text care conţine codul sursă (în limbajul C) al unui program. Fiecare fişier sursă conţine una sau mai multe *funcţii* şi eventual, referinţe către unul sau mai multe *fişiere header* (figura 2.1.).

Funcţia principală a unui program este numită ***main***. Execuţia programului începe cu execuţia acestei funcţii, care poate apela, la rândul ei, alte funcţii. Toate funcţiile folosite în program trebuie descrise în fişierele sursă (cele scrise de către programator), în fişiere header (funcţiile predefinite, existente în limbaj), sau în biblioteci de funcţii.

Un fişier header este un fişier aflat în sistem sau creat de către programator, care conţine declaraţii şi definiţii de funcţii şi variabile.

Acţiunile din fiecare funcţie sunt codificate prin *instrucţiuni* (figura 2.2.a.). Există mai multe tipuri de instrucţiuni, care vor fi discutate în capitolul următor. O instrucţiune este orice expresie validă (de obicei, o asignare sau un apel de funcţie), urmată de simbolul `;`. În figura 2.2.b. este dat un exemplu de instrucţiune simplă. Uneori, ca instrucţiune poate apare instrucţiunea nulă (doar `;`), sau instrucţiunea compusă (privită ca o succesiune de instrucţiuni simple, încadrate între acoladele delimitatoare `{}`.

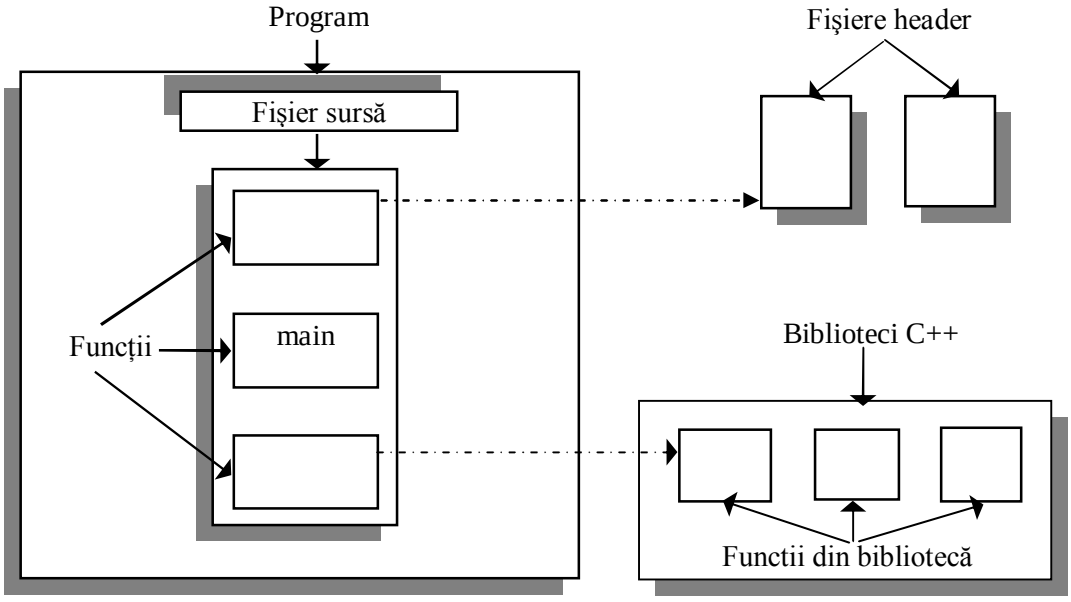

Figura 2.1. Structura unui program în limbajul C

Date, operatori şi expresii

O expresie este o structură corectă sintactic, formată din operanzi şi operatori (figura 2.2.c.).

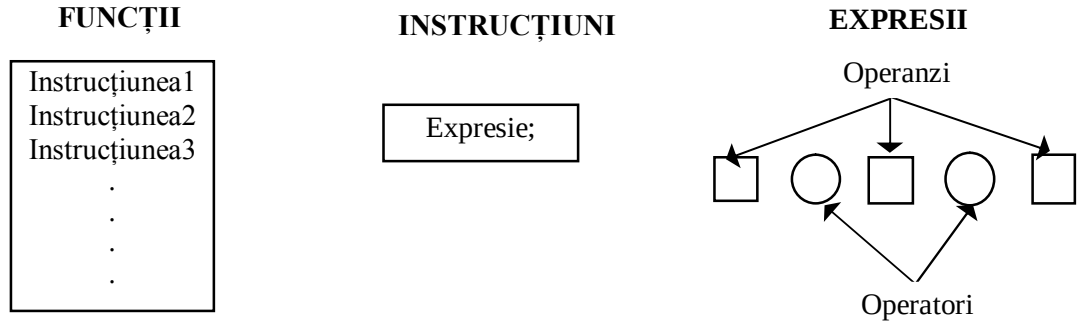

Figura 2.2. Funcţie, instrucţiune, expresie

Pentru a înţelege mai bine noţiunile prezentate, să considerăm un exemplu foarte simplu. Programul următor afişează pe ecran un mesaj (mesajul *Primul meu program*). Informaţia de prelucrat (de intrare) este însuşi mesajul (o constantă şir), iar prelucrarea ei constă în afişarea pe ecran.

Exemplu:

```
#include <iostream.h>     // linia 1
int main()          // linia 2 - antetul funcţiei main
{                           /* linia 3 - începutul corpului funcţiei, a unei intrucţiuni
compuse */
cout<<"Primul meu program in limbajul C++\n";     // linia 5
return 0 ;
}                                   // linia6-sfârşitul corpului funcţiei
```

Prima linie este o *directivă preprocesor* (indicată de simbolul #) care determină includerea în fişierul sursă a fişierului header cu numele `iostream.h`. Acest header permite realizarea afişării pe monitor.

Programul conţine o singură funcţie, *funcţia principală*, numită `main`, al cărui *antet* (linia 2) indică:

- tipul valorii returnate de funcţie (`int`, ceea ce înseamnă că funcţia nu returnează nici o valoare)
- numele funcţiei (`main`)
- lista argumentelor primite de funcţie, încadrată de cele 2 paranteze rotunde.

Funcţiile comunică între ele prin argumente. Aceste argumente reprezintă datele de intrare ale funcţiei. În cazul nostru, nu avem nici un argument în acea listă.

Ceea ce urmează după simbolul `//`, până la sfârşitul liniei, este un *comentariu*, care va fi ignorat de către compilator. Comentariul poate conţine un text explicativ; informaţii lămuritoare la anumite aspecte ale problemei sau observaţii. Dacă vrem să folosim un comentariu care cuprinde mai multe linii, vom delimita începutul acestuia indicat prin simbolulurile `/*`, iar sfârşitul - prin `*/` (vezi liniile 3, 4). Introducerea comentariilor în programele sursă uşurează înţelegerea acestora. În general, se recomandă introducerea unor comentarii după antetul unei funcţiei, pentru a preciza prelucrările efectuate în funcţie, anumite limite impuse datelor de intrare, etc.

Începutul şi sfârşitul corpului funcţiei *main* sunt indicate de cele două acoalade `{` (linia3) şi `}`(linia 6). Corpul funcţiei (linia 5) este format dintr-o singură instrucţiune, care implementează o operaţie de scriere. Cuvantul `cout` este un cuvânt predefinit al limbajului C++ - **c**onsole **out**put - care desemnează dispozitivul logic de iesire; simbolul `<<` este operatorul de transfer a informaţiei. Folosite astfel, se deschide un canal de comunicaţie a datelor către dispozitivul de ieşire, în cazul acesta, monitorul. După operator se specifică informaţiile care vor fi afişate (în acest exemplu, un şir de caractere constant). Faptul că este un şir constant de caractere este indicat de ghilimelele care îl încadrează. Pe ecran va fi afişat fiecare caracter din acest şir, cu excepţia grupului `\n`. Deşi grupul este format din două caractere, acesta va fi interpretat ca un singur caracter - numit *caracter escape* - care determină poziţionarea cursorului la începutul următoarei linii. O secvenţă escape (cum este `\n`) furnizează un mecanism general şi extensibil pentru reprezentarea caracterelor invizibile sau greu de obţinut. La sfârşitul instrucţiunii care implementează operaţia de scriere, apare `;` .

2.3. PREPROCESORUL

În faza de compilare a fişierului sursă este invocat întâi preprocesorul. Acesta tratează directivele speciale - numite *directive preprocesor* - pe care le găseşte în fişierul sursă. Directivele preprocesor sunt identificate prin simbolul `#`, care trebuie să fie primul caracter, diferit de spaţiu, dintr-o linie. Directivele preprocesor sunt utilizate la includerea fişierelor header, la definirea numelor constantelor simbolice, la definirea macro-urilor, sau la realizarea altor funcţii (de exemplu, compilarea condiţionată), aşa cum ilustrează exemplele următoare:

- Includerea fişierelor header în codul sursă:

Exemplul1:

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

Când procesorul întâlneşte această linie, datorită simbolului #, o recunoaşte ca fiind o directivă preprocesor, localizează fişierul header indicat (parantezele unghiulare < > indică faptul că este vorba de un *fişier header sistem*).

Exemplul 2:

```
#include "headerul_meu.h"
```

Numele fişierului header inclus între ghilimele, indică faptul că `headerul_meu.h` este un fişier header creat de utilizator. Preprocesorul va căuta să localizeze acest fişier în directorul curent de lucru al utilizatorului. În cazul în care fişierul header nu se află în directorul curent, se va indica şi calea către acesta.

Exemplul 3:

```
#include "c:\\bc\\head\\headerul_meu.h"
```

În acest exemplu, pentru interpretarea corectă a caracterului backslash \, a fost necesară "dublarea" acestuia, din motive pe care le vom prezenta în paragraful 2.5.2.4.

- Asignarea(atribuirea) de nume simbolice constantelor:

Exemplu:

```
#define TRUE      1
#define FALSE     0
```

Tratarea acestor directive preprocesor are ca efect asignarea (atribuirea) valorii întregi 1 numelui (constantei simbolice) `TRUE`, şi a valorii 0 numelui simbolic `FALSE`. Ca urmare, înaintea compilării propriu-zise, în programul sursă, aparițiile numelor `TRUE` şi `FALSE` vor fi înlocuite cu valorile 1, respectiv 0.

- Macrodefiniții:

Directiva `#define` este folosită şi în macrodefiniții. Macrodefinițiile permit folosirea unor nume simbolice pentru expresiile indicate în directivă.

Exemplu:

```
#define NEGATIV(x)      -(x)
```

Între numele macrodefiniției şi paranteza stângă (*`NEGATIV(`*`…)`) nu sunt permise spații albe. La întalnirea în programul sursă a macrodefiniției `NEGATIV`, preprocesorul subtituie argumentul acesteia cu expresia (negativarea argumentului). Macrodefiniția din exemplu poate fi folosită în programul sursă astfel: `NEGATIV(a+b)`. Când preprocesorul întâlneşte numele expresiei, subtituie literalii din paranteză, `a+b`, cu argumentul din macrodefiniție, x, obținându-se `-(a+b)`.

Dacă macrodefiniția ar fi fost de forma:

```
#define NEGATIV(x)      -x
```

`NEGATIV(a+b)` ar fi fost tratată ca `-a+b`.

2.4. ELEMENTE DE BAZĂ ALE LIMBAJULUI

2.4.1. VOCABULARUL

În scrierea programelor în limbajul C/C++ pot fi folosite doar anumite simboluri care alcătuiesc *alfabetul limbajului.* Acesta cuprinde:

- Literele mari sau mici de la A la Z (a-z);
- Caracterul subliniere (_ underscore), folosit, de obicei, ca element de legătura între cuvintele compuse;
- Cifrele zecimale (0-9);
- Simboluri speciale:
- Caractere:
- operatori (Exemple: `+`, `*`, `!=`);
- delimitatori (Exemple: blank (spațiu), tab `\t`, newline `\n`, cu rolul de a separa cuvintele);
- Grupuri (perechi de caractere).

Grupurile de caractere, numire adesea *separatori*, pot fi:

- `( )` - Încadrează lista de argumente ale unei funcții sau sunt folosite în expresii pentru schimbarea ordinii de efectuare a operațiilor (în ultimul caz, fiind operator);
- `{ }` - Încadrează instrucțiunile compuse;
- `//` - Indică începutul unui comentariu care se poate întinde până la sfârşitul liniei;
- `/* */` - Indică începutul şi sfârşitul unui comentariu care poate cuprinde mai multe linii;
- `" "` - Încadrează o constantă şir (un şir de caractere);
- `' '` - Încadrează o constantă caracter (un caracter imprimabil sau o secvență escape).

2.4.2. UNITĂȚILE LEXICALE

Unitățile lexicale (cuvintele) limbajului C/C++ reprezintă grupuri de caractere cu o semnificație de sine stătătoare. Acestea sunt:

- Identificatori;
- Cuvinte cheie ale limbajului;

Identificatorii reprezintă numele unor date (constante sau variabile), sau ale unor funcții. Identificatorul este format dintr-un şir de litere, cifre sau caracterul de subliniere (underscore), trebuie să înceapă cu o literă sau cu caracterul de subliniere şi să fie sugestivi.

Exemple: `viteză`, `greutate_netă`, `Viteza`, `Viteza1`, `GreutateNetă`

Identificatorii pot conține litere mici sau mari, dar limbajul C++ este senzitiv la majuscule şi minuscule (case-sensitive). Astfel, identificatorii `viteza` şi `Viteza` sunt diferiți.

Nu pot fi folosiți ca identificatori cuvintele cheie. Identificatorii pot fi standard (ca de exemplu numele unor funcții predefinite: `scanf`, `clear`, etc.) sau aleşi de utilizator.

Cuvintele cheie sunt cuvinte ale limbajului, împrumutate din limba engleză, cărora programatorul nu le poate da o altă utilizare. Cuvintele cheie se scriu cu litere mici şi pot reprezenta:

- Tipuri de date (Exemple: `int, char, double`);
- Clase de memorare (Exemple: `extern, static, register`);
- Instrucțiuni (Exemple: `if, for, while`);
- Operatori (Exemplu: `sizeof`).

Sensul cuvintelor cheie va fi explicat pe masură ce vor fi prezentate construcțiile în care acestea apar.

2.5. DATE ÎN LIMBAJUL C/C++

Aşa cum s-a văzut în capitolul 1, un program realizează o prelucrare de informaţie. Termenul de prelucrare trebuie să fie considerat într-un sens foarte general (de exemplu, în programul prezentat în paragraful 2.2., prelucrarea se referea la un text şi consta în afişarea lui). În program datele apar fie sub forma unor ***constante*** (valori cunoscute anticipat, care nu se modifică), fie sub forma de ***variabile***. Constantele şi variabilele sunt obiectele informaţionale de bază manipulate într-un program.

Fiecare categorie de date este caracterizată de atributele:

- Nume;
- Valoare;
- Tip;
- Clasa de memorare.

De primele trei tipuri de atribute ne vom ocupa în continuare, urmând ca de atributul clasă de memorare să ne ocupăm în paragraful 6.8.

Numele unei date

Numele unei date este un identificator şi, ca urmare, trebuie să respecte regulile specifice identificatorilor. Deasemenea, numărul de caractere care intră în compunerea unui identificator este nelimitat, însă, implicit, numai primele 32 de caractere sunt luate în considerare. Aceasta înseamnă că doi identificatori care au primele 32 de caractere identice, diferenţiindu-se prin caracterul 33, vor fi consideraţi identici.

2.5.1. TIPURI DE DATE

Tipul unei date constă într-o *mulţime de valori* pentru care s-a adoptat un anumit mod de reprezentare în memoria calculatorului şi o *mulţime de operatori* care pot fi aplicaţi acestor valori. Tipul unei date determină *lungimea zonei de memorie* ocupată de acea dată. În general, lungimea zonei de memorare este dependentă de calculatorul pe care s-a implementat compilatorul. Tabelul 2.1. prezintă lungimea zonei de memorie ocupată de fiecare tip de dată pentru compilatoarele sub MS-DOS şi UNIX/LINUX.

Tipurile de bază sunt:

- `char` un singur octet (1 byte=8 biţi), capabil să conţină codul unui caracter din setul local de caractere;
- `int` număr întreg, reflectă în mod tipic mărimea naturală din calculatorul utilizat;
- `float` număr real, în virgulă mobilă, simplă precizie;
- `double` număr real, în virgulă mobilă, dublă precizie.

În completare există un număr de *calificatori*, care se pot aplica tipurilor de bază `char`, `int`, `float` sau `double`: `short`, `long`, `signed` şi `unsigned`. Astfel, se obţin ***tipurile derivate de date***. `Short` şi `long` se referă la mărimea diferită a întregilor, iar datele de tip `unsigned int` sunt întotdeauna pozitive. S-a intenţionat ca `short` şi `long` să furnizeze diferite lungimi de întregi, `int` reflectând mărimea cea mai "naturală" pentru un anumit calculator. Fiecare compilator este liber să interpreteze `short` şi `long` în mod adecvat propriului hardware; în nici un caz, însă, `short` nu este mai lung decât `long`. Toţi aceşti calificatori pot aplicaţi tipului `int`. Calificatorii `signed` (cel implicit) şi `unsigned` se aplică tipului `char`. Calificatorul `long` se aplică tipului `double`. Dacă într-o declaraţie se omite tipul de bază, implicit, acesta va fi int.

Tabelul 2.1.

Tip	**Lungimea zonei de memorie ocupate (în biţi)**		**Descriere**
	MS-DOS	UNIX LINUX	
`char`	8	8	Valoarea unui singur caracter; poate fi întâlnit în expresii cu extensie de semn
`unsigned char`	8	8	Aceeaşi ca la char, fără extensie de semn
`signed char`	8	8	Aceeaşi ca la char, cu extensie de semn obligatorie
`int`	16	32	Valoare întreagă
`long (long int)`	32	64	Valoare întreagă cu precizie mare
`long long int`	32	64	Valoare întreagă cu precizie mare
`short int`	16	32	Valoare întreagă cu precizie mică
`unsigned int`	16	32	Valoare întreagă, fără semn
`unsigned long int`	32	64	Valoare întreagă, fără semn
`float`	32	32	Valoare numerică cu zecimale, simplă precizie (6)
`double`	64	64	Valoare numerică cu zecimale, dublă precizie (10)
`long double`	80	128	Valoare numerică cu zecimale, dublă precizie

Să considerăm, de exmplu, tipul `int`, folosit pentru date întregi (pozitive sau negative). Evident că mulţimea valorilor pentru acest tip va fi, de fapt, o *submulţime finită* de numere întregi. Dacă pentru memorarea unei date de tip int se folosesc 2 octeţi de memorie, atunci valoarea maximă pentru aceasta va fi $\frac{1}{2}\times 2^{16}$ - 1, deci 2^{15} - 1 (`32767`), iar valoarea minimă va fi -$\frac{1}{2}\times 2^{16}$, deci -2^{15} (`-32768`). Încercarea de a calcula o expresie de tip int a cărei valoare se situează în afara acestui domeniu va conduce la o eroare de execuţie.

Mulţimea valorilor pentru o dată de tip `unsigned int` (întreg fără semn) va fi formată din numerele întregi situate în intervalul [0, 2^{16} - 1].

În header-ul **`<values.h>`** sunt definite constantele simbolice (cum ar fi: `MAXINT`, `MAXSHORT`, `MAXLONG`, `MINDOUBLE`, `MINFLOAT`, etc.) care au ca valoare limitele inferioară şi superioară ale intervalului de valori pentru tipurile de date enumerate. (de exemplu `MAXINT` reprezintă valoarea întregului maxim care se poate memora, etc.)

Fără a detalia foarte mult modul de reprezentare a datelor reale (de tip `float` sau `double`), vom sublinia faptul că, pentru acestea, este importantă şi *precizia de reprezentare*. Deoarece calculatorul poate reprezenta doar o submulţime finită de valori reale, în anumite cazuri, pot apare erori importante.

Numerele reale pot fi scrise sub forma: $N = \text{mantisa} \times \text{baza}^{\text{exponent}}$

unde:`baza` reprezintă baza sistemului de numeraţie; `mantisa` (`coeficientul`) este un număr fracţionar normalizat (în faţa virgulei se află 0, iar prima cifră de după virgulă este diferită de zero); `exponentul` este un număr întreg. Deoarece forma internă de reprezentare este binară, baza=2. În memorie vor fi reprezentate doar mantisa şi exponentul. Numărul de cifre de după virgulă determină *precizia* de exprimare a numărului. Ce alte cuvinte, pe un calculator cu o precizie de 6 cifre semnificative, două valori reale care diferă la a 7-a cifră zecimală, vor avea aceeaşi reprezentare. Pentru datele de tip float, precizia de reprezentare este 6; pentru cele de tip double, precizia este 14, iar pentru cele de tip long double, precizia este 20.

Lungimea zonei de memorie ocupate de o dată de un anumit tip (pe câţi octeţi este memorată data) poate fi aflată cu ajutorul operatorului `sizeof`.

Exemplu:

```
cout<<"Un int este memorat pe "<<sizeof(int)<<"octeti.\n";
```

Instrucţiunea are ca efect afişarea pe monitor a mesajului: *Un int este memorat pe 2 octeţi.*

2.5.2. CONSTANTE

O constantă este un **literal** (o formă externă de reprezentare) *numeric*, *caracter* sau *şir de caractere*. Numele şi valoarea unei constante sunt identice. Valoarea unei constante nu poate fi schimbată în timpul execuţiei programului în care a fost utilizată. Tipul şi valoarea ei sunt determinate în mod automat, de către compilator, pe baza caracterelor care compun literalul.

2.5.2.1. Constante întregi

Constantele întregi sunt literali numerici (compuşi din cifre), fără punct zecimal.

- Constante *întregi în baza 10, 8 sau 16*
- Constante *întregi în baza 10*

Exemple:

```
45
-78     // constante întregi decimale (în baza 10), tip int
```

- Constante *întregi octale*

Dacă în faţa numărului apare cifra zero (0), acest lucru indică faptul că acea constantă este de tipul int, in baza opt (constantă octală).

Exemple:

```
056
077     // constante întregi octale, tip int
```

- Constante *întregi hexagesimale*

Dacă în faţa numărului apar caracterele zero (0) şi x (sau X), acest lucru indică faptul că acea constantă este de tipul int, în baza 16 (constantă hexagesimală). Amintim că în baza 16 cifrele sunt: 0-9, A (sau a) cu valoare 10, B (sau b) cu valoare 11, C (sau c) cu valoare 12, D (sau d) cu valoare 13, E (sau e) cu valoare 14, F (sau f) cu valoare 15.

Exemple:

```
0x45
0x3A
0Xbc    // constante întregi hexagesimale, tip int
```

- Constante *întregi, de tipuri derivate*
- Dacă secvenţa de cifre este urmată de `L` sau `l`, tipul constantei este **long int**.

Exemple:

```
145677L
897655l         // tip decimal long int
```

- Dacă secvenţa de cifre este urmată de `U` sau `u`, tipul constantei este **unsigned int**.

Exemple:

```
65555u
```

- Dacă secvenţa de cifre este urmată de U (u) şi `L` (`l`), tipul constantei este **unsigned long int**.

Exemple: `7899UL` //tip decimal **unsigned long int**

2.5.2.2. Constante numerice, reale

- Dacă o constantă numerică conţine punctul zecimal, ea este de tipul **double**.

Exemplu:

```
3.1459                  //tip double
```

- Dacă numărul este urmat de F sau f, constante este de tip **float**.
- Dacă numărul este urmat de L sau l, este de tip **long double.**

Exemplu:

```
0.45f           //tip float
9.788L          //tip long double
```

- Constante reale în format ştiinţific

Numărul poate fi urmat de caracterul `e` sau `E` şi de un număr întreg, cu sau fără semn. În acest caz, constanta este în *notaţie ştiinţifică*. În această formă externă de reprezentare, numărul din faţa literei E reprezintă *mantisa*, iar numărul întreg care urmează caracterului E reprezintă *exponentul*. In forma externă de reprezentare, baza de numeraţie este 10, deci valoarea constantei va fi dată de $\text{mantisa} \times 10^{\text{exponent}}$.

Exemplu:

`1.5e-2` //tip double, în notație științifică, valoare 1.5×10^{-2}

Exercițiu: Să se scrie următorul program şi să se urmărească rezultatele execuției acestuia.

```
#include <iostream.h>
#include <values.h>
#define PI  3.14359
int main()
{
cout<<"Tipul int memorat pe: "<<sizeof(int)<<" octeti\n";
cout<<"Tipul int memorat pe: "<<sizeof(23)<<" octeti\n"; //23-const. zecimala int
cout<<"Int maxim="<<MAXINT<<'\n';
//const. simbolice MAXINT, MAXLONG, etc. - definite in <values.h>
cout<<"Const. octala 077 are val decimala:"<<077<<'\n;
cout<<"Const. hexagesimala d3 are val decimala:"<<0xd3<<'\n';
cout<<"Tipul unsigned int memorat pe:"<<sizeof(unsigned int)<<" octeti\n";
cout<<"Tipul unsigned int memorat pe: "<<sizeof(23U)<<" octeti\n";
cout<<"Tipul unsigned int memorat pe: "<<sizeof(23u)<<" octeti\n";
cout<<"Tipul long int memorat pe: "<<sizeof(long int)<<" octeti\n";
cout<<"Tipul long int memorat pe: "<<sizeof(23L)<<" octeti\n";
cout<<"Tipul long int memorat pe: "<<sizeof(23l)<<" octeti\n";
//23L sau 23l-const. decimala long int
cout<<"Long int maxim="<<MAXLONG<<'\n';
cout<<"Tipul unsigned long memorat pe:";
cout<<sizeof(unsigned long int)<<" octeti\n";
cout<<"Tipul unsigned long memorat pe: "<<sizeof(23UL)<<" octeti\n";
cout<<"Tipul unsigned long memorat pe: "<<sizeof(23ul)<<" octeti\n";
//23UL sau 23ul-const. decimala unsigned long int
cout<<"Tipul long long int memorat pe: ";
cout<<sizeof(long long int)<<" octeti\n";
cout<<"Tipul long long int memorat pe: "<<sizeof(d)<<" octeti\n";
cout<<"Tipul short int memorat pe: "<<sizeof(short int)<<" octeti\n";
cout<<"Short int maxim="<<MAXSHORT<<'\n';
cout<<"Tipul float memorat pe: "<<sizeof(float)<<" octeti\n";
cout<<"Tipul float memorat pe: "<<sizeof(23.7f)<<" octeti\n";
//23.7f-const. decimala float
cout<<"Float maxim="<<MAXFLOAT<<'\n';
cout<<"Float minim="<<MINFLOAT<<'\n';
cout<<"Tipul double memorat pe: "<<sizeof(double)<<" octeti\n";
cout<<"Tipul double memorat pe: "<<sizeof(23.7)<<" octeti\n";
//23.7-const. decimala double
cout<<"Const. decim. doubla in notatie stiintifica:"<<23.7e-5<<'\n';
cout<<"Const. PI este:"<<PI<<'\n';
cout<<"Constanta PI este memorata pe:"<<sizeof(PI)<<"octeti\n":
cout<<"Double maxim="<<MAXDOUBLE<<'\n'<<"Double minim="<<MINDOUBLE<<'\n';
cout<<"Tipul long double memorat pe: "<<sizeof(long double)<<" octeti\n";
cout<<"Tipul long double memorat pe: "<<sizeof(23.7L)<<" octeti\n";
//23.7L-const. decimala long double
cout<<"Cifra A din HEXA are val.:"<<0xA<<"\n";
cout<<"Cifra B din HEXA are val.:"<<0XB<<"\n";
cout<<"Cifra C din HEXA are val.:"<<0xc<<"\n";
cout<<" Cifra D din HEXA are val.:"<<0xD<<"\n";
cout<<" Cifra E din HEXA are val.:"<<0XE<<"\n";
cout<<" Cifra F din HEXA are val.:"<<0xf<<"\n";
cout<<"Val. const. hexa 0x7ac1e este: "<<0x7ac1e<<'\n';
cout<<"Val. const. octale 171 este: "<<0171<<'\n';
cout<<"O const. octala se memoreaza pe "<<sizeof(011)<<" octeti\n";
cout<<"O const.oct.long se mem pe ";cout<<sizeof(011L)<<" octeti\n";
return 0 ;
}
```

2.5.2.3. Constante caracter

Constantele caracter sunt încadrate între apostrof.

Exemplu:

```
'a'          //tip char
```

O constantă caracter are ca valoare ***codul ASCII*** al caracterului pe care îl reprezintă.

Acest set de caractere are următoarele *proprietăți*:

- Fiecărui caracter îi corespunde o valoare întreagă distinctă (ordinală);
- Valorile ordinale ale literelor mari sunt ordonate şi consecutive ('A' are codul ASCII 65, 'B' - codul 66, 'C' - codul 67, etc.);
- Valorile ordinale ale literelor mici sunt ordonate şi consecutive ('a' are codul ASCII 97, 'b' - codul 98, 'c' - codul 99, etc.);
- Valorile ordinale ale cifrelor sunt ordonate şi consecutive ('0' are codul ASCII 48, '1' - codul 49, '2' - codul 50, etc.).

- Constante caracter *corespunzătoare caracterelor imprimabile*

O constantă caracter corespunzătoare unui caracter imprimabil se reprezintă prin caracterul respectiv inclus între apostroafe.

Exemplu:

Constantă caracter	Valoare
'A'	65
'a'	97
'0'	48
'*'	42

Excepții de la regula de mai sus le constituie *caracterele imprimabile apostrof* (') şi backslash (\).

Caracterul *backslash* se reprezintă: `'\\'`. Caracterul *apostrof* se reprezintă: `'\''`.

- Constante caracter *corespunzătoare caracterelor neimprimabile*

Pentru caracterele neimprimabile, se folosesc ***secvențe escape***. O secvență escape furnizează un mecanism general şi extensibil pentru reprezentarea caracterelor invizibile sau greu de obținut. În tabelul 2.2. sunt prezentate câteva caractere escape utilizate frecvent.

Tabelul 2.2.

Constantă caracter	Valoare (Cod ASCII)	Denumirea caracterului	Utilizare
'\n'	10	LF	rând nou (Line Feed)
'\t'	9	HT	tabulator orizontal
'\r'	13	CR	poziționează cursorul în coloana 1 din rândul curent
'\f'	12	FF	salt de pagină la imprimantă (Form Feed)
'\a'	7	BEL	activare sunet

O constantă caracter pentru o secvență escape poate apare însă, şi sub o formă în care se indică codul ASCII, în octal, al caracterului dorit:

`'\ddd'` unde d este o cifră octală.

Exemple:

`'\11'` (pentru `'\t'`)

reprezintă constanta caracter backspace, cu codul 9 în baza 10, deci codul 11 în baza 8.

`'\15'` (pentru `'\r'`)

reprezintă constanta caracter CR, cu codul 13 în baza 10, deci codul 11 în baza 8.

Exercițiu: Să se scrie următorul program şi să se urmărească rezultatele execuției acestuia.

```
#include <iostream.h>
int main()
{
cout<<"Un caracter este memorat pe "<<sizeof(char)<<" octet\n";
cout<<"Caracterul escape \\n este memorat pe ";
cout<<sizeof('\n')<<" octet\n";
```

```
cout<<"Caracterul escape '\\n\' este memorat pe "<<sizeof('\n');
cout<<" octet\n";
cout<<"Caracterul '9' este memorat pe "<<sizeof('9')<<" octet\n";
cout<<'B';cout<<' ';cout<<'c';cout<<'\t';
cout<<'\t';cout<<'9';cout<<'\b';cout<<'\a';
cout<<'L';cout<<'\v';cout<<'L';
cout<<'\'';cout<<'\t';cout<<'\"';cout<<'\\';cout<<'\n';
cout<<'\a';cout<<'\7';
return 0 ;
}
```

2.5.2.4. Constante şir de caractere

Constanta şir este o succesiune de zero sau mai multe caractere, încadrate de ghilimele. În componența unui şir de caractere, poate intra orice caracter, deci şi caracterele escape. Lungimea unui şir este practic nelimitată. Dacă se doreşte continuarea unui şir pe rândul următor, se foloseşte caracterul backslash.

Caracterele componente ale unui şir sunt memorate într-o zonă continuă de memorie (la adrese succesive). Pentru fiecare caracter se memorează codul ASCII al acestuia. După ultimul caracter al şirului, compilatorul plasează automat *caracterul NULL* (`\0`), caracter care reprezintă *marcatorul sfârşitului de şir*. Numărul de octeți pe care este memorat un şir va fi, deci, mai mare cu 1 decât numărul de caractere din şir.

Exemple:

```
"Acesta este un şir de caractere"     //constantă şir memorată pe 32 octeți
"Şir de caractere continuat\"
      pe rândul următor!"                    //constantă şir memorată pe 45 octeți
      "Şir \t cu secvențe escape\n"          //constantă şir memorată pe 26 octeți
      '\n'                                   //constantă caracter memorată pe un octet
      "\n"  //constanta şir memorată pe 2 octeți (codul caracterului escape şi terminatorul de şir)
      "a\a4"              /*Şir memorat pe 4 octeți:
                                Pe primul octet: codul ASCII al caracterului a
                                Pe al doilea octet: codul ASCII al caracterului escape \a
                                Pe al treilea octet: codul ASCII al caracterului 4
                                Pe al patrulea octet: terminatorul de şir NULL, cod ASCII 0 */
      "\\ASCII\\"         /*Şir memorat pe 8 octeți:
Pe primul octet: codul ASCII al caracterului backslah
Pe al doilea octet: codul ASCII al caracterului A
Pe al treilea octet: codul ASCII al caracterului S
Pe al patrulea octet: codul ASCII al caracterului S
Pe al 6-lea octet: codul ASCII al caracterului I
Pe al 7-lea octet: codul ASCII al caracterului I
Pe al 8-lea octet: codul ASCII al caracterului backslah
Pe al 9-ea octet: terminatorul de şir NULL, de cod ASCII 0 */
      "1\175a"            /*Şir memorat pe 4 octeți:
Primul octet: Codul ASCII al caracterul 1
Al 2-lea octet: codul ASCII 125 (175 in octal) al caracterului }
Al 3-lea octet: codul ASCII al caracterului a
Al 4-lea octet: codul ASCII 0 pentru terminatorul şirului */
```

Exercițiu: Să se scrie următorul program şi să se urmărească rezultatele execuției acestuia.

```
#include <iostream.h>
int main()
{
cout<<"Şirul \"Ab9d\" este memorat pe:"<<sizeof("Ab9d")<<" octeți\n";
cout<<"Şirul \"Abcd\\t\" este memorat pe:"<<sizeof("Abcd\t")<<" octeți\n";
cout<<"Şirul \"\n\" este memorat pe "<<sizeof("\n")<<" octeți\n";
cout<<"Şirul \"\\n\" este memorat pe "<<sizeof("\n")<<" octeți\n";
cout<<"Şirul \"ABCDE\" se memorează pe "<<sizeof("ABCDE")<<" octeți\n";
return 0 ;
}
```

2.5.3. VARIABILE

Spre deosebire de constante, variabilele sunt date (obiecte informaționale) ale căror valori se pot modifica în timpul execuției programului. Şi variabilele sunt caracterizate de atributele *nume, tip, valoare* şi *clasă de memorare*. Variabilele sunt *nume simbolice* utilizate pentru memorarea valorilor introduse pentru datele de intrare sau a rezultatelor. Dacă la o constantă ne puteam referi folosind caracterele componente, la o variabilă ne vom referi prin numele ei. Numele unei variabile ne permite accesul la valoarea ei, sau schimbarea valorii sale, dacă este necesar acest lucru. Numele unei variabile este un identificator ales de programator. Ca urmare, trebuie respectate regulile enumerate în secțiunea identificatori.

Dacă o dată nu are legături cu alte date (de exemplu, relația de ordine), vom spune că este o dată *izolată*. O dată izolată este o *variabilă simplă*. Dacă datele se grupează într-un anumit mod (în tablouri - vectori, matrici - sau structuri), variabilele sunt *compuse* (*structurate*).

În cazul constantelor, în funcție de componența literalului, compilatorul stabilea, automat, tipul constantei. În cazul variabilelor este necesară specificarea tipului fiecăreia, la declararea acesteia. Toate variabilele care vor fi folosite în program, trebuie declarate înainte de utilizare.

2.5.3.1. Declararea variabilelor

Modul general de declarare a variabilelor este:

```
tip_variabile    listă_nume_variabile;
```

Se specifică tipul variabilei(lor) şi o listă formată din unul sau mai mulți identificatori ai variabilelor de tipul respectiv. Într-un program în limbajul **C++**, declarațiile de variabile pot apare în orice loc în programul sursă. La declararea variabilelor, se rezervă în memorie un număr de octeți corespunzător tipului variabilei, urmând ca ulterior, în acea zonă de memorie, să fie depusă (memorată, înregistrată) o anumită valoare.

Exemple:

```
int i, j;/*declararea var. simple i, j, de tip int. Se rezervă pentru i şi j câte 16 biți (2octeți)*/
char c;            /* declararea variabilei simple c, de tip char. Se rezervă un octet. */
float lungime;     /* declararea variabilei simple lungime; se rezervă 4 octeți */
```

2.5.3.2. Inițializarea variabilelor în declarații

În momentul declarării unei variabile, acesteia i se poate da (asigna, atribui) o anumită valoare. În acest caz, în memorie se rezervă numărul de locații corespunzător tipului variabilei respective, iar valoarea va fi depusă (memorată) în acele locații.

Forma unei declarații de variabile cu atribuire este:

```
tip_variabilă  nume_variabilă=expresie;
```

Se evaluează expresia, iar rezultatul acesteia este asignat variabilei specificate.

Exemple:

```
char backslash='\\';     //declararea şi inițializarea variabilei simple backslash
int a=7*9+2;       /* declararea variabilei simple a, de tip int şi inițializarea ei cu valoarea 65*/
float radiani, pi=3.14;/*declararea variabilei radiani;declararea şi inițializarea var. pi*/
short int z=3;                  //declararea şi inițializarea variabilei simple z
char d='\011';
char LinieNoua='\n';
double x=9.8, y=0;
```

Compilatorul C++ furnizează mecanisme care permit programatorului să influențeze codul generat la compilare, prin aşa-numiții ***calificatori***.

Aceştia sunt:

- `const`;
- `volatile`.

Calificatorul **const** asociat unei variabile, nu va permite modificarea ulterioară a valorii acesteia, prin program (printr-o atribuire). Calificatorul **volatile** (cel implicit) are efect invers calificatorului const. Dacă după calificator nu este specificat tipul datei, acesta este considerat tipul implicit, adică **int**.

Exemple:

```
const float b=8.8;
      volatile char terminator;terminator='@';terminator='*';      //permis
b=4/5;                              //nepermisa modificarea valorii variabilei b
const w;     volatile g;            //w, g de tip int, implicit
```

2.5.3.3. Operații de intrare/ieşire

Limbajele C/C++ nu posedă instrucțiuni de intrare/ieşire, deci de citire/scriere (ca limbajul PASCAL, de exemplu). În limbajul C aceste operații se realizează cu ajutorul unor funcții (de exemplu, printf şi scanf), iar în limbajul C++ prin supraîncărcarea operatorilor (definirea unor noi proprietăți ale unor operatori existenți, fără ca proprietățile anterioare să dispară), mai precis a operatorilor **>>** şi **<<** . Vom folosi în continuare abordarea limbajului C++, fiind, în momentul de față, mai simplă. În limbajul C++ sunt predefinite următoarele dispozitive logice de intrare/ieşire:

`cin` - **c**onsole **in**put - dispozitivul de intrare (tastatura);

`cout` - **c**onsole **ou**tput - dispozitivul de ieşire (monitorul).

Aşa cum se va vedea în capitolul 9, `cin` şi `cout` sunt, de fapt, obiecte (predefinite). Transferul informației se realizează cu **operatorul >>** pentru intrare şi **operatorul <<** pentru ieşire. Utilizarea dispozitivelor de intrare/ieşire cu operatorii corespunzători determină deschiderea unui canal de comunicație a datelor către dispozitivul respectiv. După operator se specifică informațiile care vor fi citite sau afişate.

Exemple:

```
cout << var;        /* afişează valoarea variabilei var pe monitor*/
cin  >> var;        /* citeşte valoarea variabilei var de la tasatatură */
```

Sunt posibile operarții multiple, de tipul:

Exemple:

```
cout << var1 << var2 << var3;
cin >> var1 >> var2 >> var3;
```

În acest caz, se efectuează succesiv, de la stânga la dreapta, scrierea, respectiv citirea valorilor variabilelor `var1`, `var2` şi `var3`.

Operatorul **>>** se numeşte ***operator extractor*** (extrage valori din fluxul datelor de intrare, conform tipului acestora), iar operatorul **<<** se numeşte ***operator insertor*** (inserează valori în fluxul datelor de ieşire, conform tipului acestora). Tipurile de date citite de la tastatură pot fi toate tipurile numerice, caracter sau şir de caractere. Tipurile de date transferate către ieşire pot fi: toate tipurile numerice, caracter sau şir de caractere. Operanzii operatorului extractor (**>>**) pot fi doar nume de variabile. Operanzii operatorului insertor (**<<**) pot fi nume de variabile (caz în care se afişează valoarea variabilei), constante sau expresii. Utilizarea dispozitivelor şi operatorilor de intrare/ieşire în C++ impune includerea fişierului **`iostream.h`**.

Exemple:

```
char c;
cout<<"Astept un caracter:";  //afişarea constantei şir de caractere, deci a mesajului
cin>>c;                       //citirea valorii variabilei c, de tip caracter
int a, b, e; double d;
cin>>a>>b>>e>>d;          //citirea valorilor variabilelor a, b, e, d de tip int, int, int, double
cout<<"a="<<a<<"Valoarea expresiei a+b este:"<<a+b<<'\n';
```

2.6. OPERATORI ŞI EXPRESII

Datele (constante sau variabile) legate prin operatori, formează **expresii** (figura 2.4). Operatorii care pot fi aplicați datelor (operanzilor) depind de tipul operanzilor, datorită faptului că tipul unei date constă într-o mulțime de valori pentru care s-a adoptat un anumit mod de reprezentare în memoria calculatorului şi *o mulțime de operatori* care pot fi aplicați acestor valori.

Operatorii pot fi:
- unari (necesită un singur operand);
- binari (necesită doi operanzi);
- ternari (trei operanzi).

O **expresie** este o combinație corectă din punct de vedere sintactic, formată din operanzi şi operatori. Expresiile, ca şi operanzii, au ***tip*** şi ***valoare***.

2.6.1. OPERATORI

- Operatorul unar ***adresă &***, aplicat identificatorului unei variabile, furnizează adresa la care este memorată aceasta. Poate fi aplicat *oricărui tip de date* şi se mai numeşte *operator de referențiere.*

Exemplu:
```
int a;
cout<<"Adresa la care este memorata variabila a este:"<<&a;
```

- Operatorul ***de atribuire*** *(de asignare)* este un operator *binar* care se aplică tuturor tipurilor de variabile. Este folosit sub formele următoare:

```
nume_variabilă=expresie;
```
sau:
```
expresie1=expresie2;
```

Se evaluează expresia din membrul drept, iar valoarea acesteia este atribuită variabilei din membrul stâng. Dacă tipurile membrilor stâng şi drept diferă, se pot realiza anumite conversii, prezentate în paragraful 2.7.

Exemplu:
```
float x; int a,b; x=9.18;
a=b=10;
int s; s=a+20*5;                  //rezultat: s=110
s=x+2;                            //rezultat s=11, deoarece s este int.
```

Aşa cum se observă în linia a 2-a din exemplul precedent, operatorul de atribuire poate fi utilizat de mai multe ori în aceeaşi expresie. Asociativitatea operatorului are loc de la dreapta la stânga. Astfel, mai întâi b=10, apoi a=b.

Exercițiu: Să se scrie următorul program şi să se urmărească rezultatele execuției acestuia.
```
#include <iostream.h>
int main()
{
float x,y=4.25;   char car='A'; int a,b,c;
cout<<"Val. lui y este:"<<y<<'\n';              //Afişare: Val. lui y este:4.25
x=y; cout<<"Val. lui x este:"<<x<<'\n';         //Afişare: Val. lui x este:4.25
a=x;cout<<"Val.lui a este:"<<a<<'\n'; //Afişare:Val. lui a este:4, deoarece a de tip int!!!
c=b=a; cout<<"b="<<b<<"\tc="<<c<<'\n';          //Afişare: b=4      c=4
cout<<"Introduceți val. lui c:"; cin>>c;        // citire val. pentru c
cout<<"Val. lui c este:"<<c<<'\n';              //Afişare: Val. lui c este:4
}
```

Operatorul poate fi aplicat tipurilor de date întregi, reale, caracter, şi chiar şiruri de caractere, aşa cum vom vedea în capitolele următoare (exemplu: char şir [10]="a5dfgthklj").

- Operatori ***aritmetici unari***:

Operator	**Semnificație**	**Exemple**
-	Minus unar	-a
++	Operator de incrementare (adună 1 la valoarea operandului)	a++ sau ++a
--	Operator de decrementare (scade 1 din valoarea operandului)	a-- sau --a

- Operatorul **-** unar schimbă semnul operandului.

Exemplu:

```
int a,b;    cout<<"a="<<-a<<'\n';   b=-a;
cout<<"b="<<b<<'\n';
```

Operatorul - unar poate fi aplicat datelor întregi, reale, caracter.

- Operatorii de incrementare şi decrementare pot fi aplicați *datelor numerice sau caracter*.

Ambii operatori pot fi folosiți în formă *prefixată*, înaintea operandului, (`++a`, respectiv `--a`) sau *postfixată*, după operand (`a++`, respectiv `a--`).

Operatorul de decrementare `--` care poate fi folosit în formă *prefixată* (--a) sau *postfixată* (a--).

Utilizarea acestor operatori în expresii, în formă prefixată sau postfixată, determină evaluarea acestora în moduri diferite, astfel:

y=++x	este echivalent cu:	x=x+1; y=x;
y=x++ x=x+1;	este echivalent cu:	y=x;
y=--x	este echivalent cu:	x=x-1; y=x;
y=x-- x=x-1;	este echivalent cu:	y=x;

***Exercițiu*:** Să se scrie următorul program şi să se urmărească rezultatele execuției acestuia.

```
#include <iostream.h>
int main()
{ int a=9; cout<<"a++="<<a++<<'\n';  //Afişare: a++=9
cout<<"a="<<a<<'\n';                 //Afişare: a=10
a=9;                                 //Revenire in situatia anterioara
cout<<"++a="<<++a<<'\n';             //Afişare: ++a=10
cout<<"a="<<a<<'\n';                 //Afişare: a=10
a=9; cout<<"a--="<<a--<<'\n';        //Afişare: a--=9
cout<<"a="<<a<<'\n';                 //Afişare: a=8
a=9;                                 //Revenire in situația anterioara
cout<<"--a="<<--a<<'\n';             //Afişare: --a=8
cout<<"a="<<a<<'\n';                 //Afişare: a=8
int z,x=3; z=x++-2;
cout<<"z="<<z<<'\n';                 //Afişare: z=1
cout<<"x="<<x<<'\n';                 //Afişare: x=4
x=3; z=++x-2; cout<<"z="<<z<<'\n';   //Afişare: z=2
cout<<"x="<<x<<'\n';                 //Afişare: x=4
return 0 ;
}
```

- Operatori ***aritmetici binari:***

Operator	**Semnificație**	**Exemple**
+	Adunarea celor doi operanzi	a+b
-	Scăderea celor doi operanzi	a-b

*****	Înmulţirea celor doi operanzi	a*b
/	Împărţirea celor doi operanzi	a/b
%	Operatorul modulo (operatorul rest) (furnizează restul împărţirii operatorului stâng la operatorul drept).	a%b

Operatorul modulo se aplică numai operanzilor întregi (de tip int sau char). Ceilalţi operatori aritmetici binari pot fi aplicaţi datelor întregi sau reale.

Dacă într-o expresie cu 2 operanzi şi un operator binar aritmetic, ambii operanzi sunt întregi, rezultatul expresiei va fi tot un număr întreg. De exemplu, la evaluarea expresiei 9/2, ambii operanzi fiind întregi, rezultatul furnizat este numărul întreg 4.

Operatorii prezentaţi respectă o serie de reguli de precedenţă (prioritate) şi asociativitate, care determină precis modul în care va fi evaluată expresia în care aceştia apar. În tabelul 2.3 sunt prezentaţi operatorii anteriori, în ordinea descrescătoare a priorităţii. Precedenţa operatorilor poate fi schimbată cu ajutorul parantezelor.

Tabelul 2.3.

Clasă de operatori	**Operatori**	**Asociativitate**
Unari	- (unar) ++ --	de la dreapta la stânga
Multiplicativi	* / %	de la stânga la dreapta
Aditivi	+ -	de la stânga la dreapta
Atribuire	=	de la dreapta la stânga

Exerciţiu: Să se scrie următorul program şi să se urmărească rezultatele execuţiei acestuia.

```
#include <iostream.h>
int main()
{
int rezult, a=20,b=2,c=25,d=4; rezult=a-b;
cout<<"a-b="<<rezult<<'\n';                          // Afişare: a-b=18
rezult=a+b; cout<<"a+b="<<rezult<<'\n';              // Afişare: a+b=22
rezult=a*b;cout<<"c*b="<<rezult<<'\n';               // Afişare: c*b=50
rezult=a/d; cout<<"a/d="<<rezult<<'\n';              // Afişare: a/d=5
rezult=c%b; cout<<"c%b="<<rezult<<'\n';              // Afişare: c%b=1
rezult=c/b*d; cout<<"c/b*d="<<rezult<<'\n';          // Afişare: c/b*d=48
rezult= -b+a; cout<<"-b+a="<<rezult<<'\n';           // Afişare: -b+a=18
rezult= -(b+a); cout<<"-(b+a)="<<rezult<<'\n';       // Afişare: -(b+a)=-22
rezult=b+c*d;cout<<"b+c*d="<<rezult<<'\n';           // Afişare: b+c*d=102
rezult=(b+c)*d;cout<<"(b+c)*d="<<rezult<<'\n';       // Afişare: (b+c)*d=108
return 0 ;
}
```

- Operatori ***aritmetici binari compuşi***

Operator	**Semnificaţie**	**Exemple**
+=	a=a+b	a+=b
-=	a=a+b	a-=b
***=**	a=a*b	a*=b
/=	a=a/b	a/=b
%=	a=a%b	a%=b

Aceşti operatori se obţin prin combinarea operatorilor aritmetici binari cu operatorul de atribuire şi sunt folosiţi sub forma următoare:

```
expresie1  operator= expresie2;
```

Rezultatul obţinut este acelaşi cu rezultatul obţinut prin:

```
expresie1 = expresie1  operator  expresie2;
```

Toţi aceşti operatorii modifică valoarea operandului stâng prin adunarea, scăderea, înmulţirea sau împărţirea acestuia prin valoarea operandului drept.

Construcţia x+=1 generează acelaşi rezultat ca expresia x=x+1.

Observaţiile referitoare la operatorii aritmetici binari sunt valabile şi pentru operatorii aritmetici binari compuşi. Operatorii aritmetici binari compuşi au aceeaşi prioritate şi asociativitate ca şi operatorul de atribuire.

Exerciţiu: Să se scrie următorul program şi să se urmărească rezultatele execuţiei acestuia.

```
#include <iostream.h>
int main()
{
int a,b; float c=9.3; a=3; b=8;
cout<<"a="<<a<<'\n';                    //Afişare a=3
a+=b; cout<<"a="<<a<<'\n';              //Afişare a=11
a-=b; cout<<"a="<<a<<'\n';              //Afişare a=-5
a*=b; cout<<"a="<<a<<'\n';              //Afişare a=24
a/=b; cout<<"a="<<a<<'\n';              //Afişare a=0
a%=b; cout<<"a="<<a<<'\n';              //Afisare a=3
return 0 ;
}
```

- Operatori ***relaţionali binari***

Operator	**Semnificaţie**	**Exemple**
==	Egal cu	a==b
!=	Diferit de	a!=b
<	Mai mic decât	a<b
<=	Mai mic sau egal	a<=b
>	Mai mare decât	a>b
>=	Mai mare sau egal	a>=b

Primii doi operatori mai sunt numiţi *operatori de egalitate*. Operatorii relaţionali servesc la compararea valorilor celor doi operanzi şi nu modifică valorile operanzilor. Rezultatul unei expresii în care apare unul din operatorii relaţionali binari este întreg şi are valoarea zero (0) dacă relaţia este falsă, sau valoarea unu (1) (sau diferită de 0 în cazul compilatoarelor sub UNIX), dacă relaţia este adevărată. Aceşti operatorii pot fi aplicaţi datelor de tip întreg, real sau char.

Regulile de precedenţă şi asociativitate ale acestor operatori sunt prezentate în tabelul 2.4.

Tabelul 2.4.

Clasă de operatori	**Operatori**	**Asociativitate**
Unari	- (unar) ++ --	de la dreapta la stânga
Multiplicativi	* / %	de la stânga la dreapta
Aditivi	+ -	de la stânga la dreapta
Atribuire	=	de la dreapta la stânga
Relaţionali	< <= > >=	de la stânga la dreapta
De egalitate	== !=	de la stânga la dreapta
Atribuire şi aritmetici binari	= *= /= %= += -=	de la dreapta la stânga

Observaţie: Deosebirea dintre operatorii **==** (relaţional, de egalitate) şi **=** (de atribuire) constă în faptul că primul nu modifică valoarea nici unuia dintre operanzii săi, pe când cel de-al doilea modifică valoarea operandului stâng (vezi exemplul următor)

Exerciţiu: Să se scrie următorul program şi să se urmărească rezultatele execuţiei acestuia.

```
#include <iostream.h>
int main()
{
int a=1, b=20, lim=100; int rezult; rezult=a<b;
cout<<"a<b="<<rezult<<'\n';
// Afişare: a<b=1 (sau o altă valoare diferită de zero pentru alte compilatoare)
rezult=a<=b;
//operatorul realţional <= are prioritate mai mare decât cel de atribuire
```

```
cout<<"a<=b="<<rezult<<'\n';
// Afisare: a<b=1 (sau o alta valoare diferită de zero pentru alte compilatoare)
rezult=a>b; cout<<"a>b="<<rezult<<'\n';          // Afişare: a<b=0
rezult=a+10>=lim;  cout<<"a+10>=lim="<<rezult<<'\n';
/* Operatorul + are prioritate mai mare decât operatorul >= .  Afişare: a+10>=lim=0 */
rezult=a+(10>=lim);      cout<<"a+(10>=lim)="<<rezult<<'\n';
/* Schimbarea prioritatii operatorilor prin folosirea parantezelor;  Afişare: a+(10>=lim)=1  */
rezult=a==b;
cout<<"a==b="<<rezult<<'\n';          // Afişare: a==b=0
cout<<"a="<<a<<'\n';                  // Afişare: a=1
cout<<"b="<<b<<'\n';                  // Afişare: b=20
rezult=a=b; cout<<"a=b="<<rezult<<'\n';     // Afişare: a=b=20
cout<<"a="<<a<<'\n';                  // Afişare: a=20
cout<<"b="<<b<<'\n';                  // Afişare: b=20
rezult=5>b>10;cout<<"b="<<b<<'\n';    // Afişare: b=20
cout<<"5>b>10="<<rezult<<'\n';        //Echivalent cu (5>b)>10 Afişare: 5>b>10=0
return 0 ;
}
```

- Operatori ***logici pe cuvânt***

Operator	**Semnificație**	**Exemple**
!	Not (**negație** logică)	!(a==b)
&&	And (conjuncție, **şi** logic)	**(**a>b) && (b>c)
||	Or **(**disjuncție, **sau** logic)	**(**a>b) || (b>c)

Aceşti operatori pot fi aplicați datelor de tip întreg, real sau caracter. Evaluarea unei expresii în care intervin operatorii logici se face conform tabelului 2.5.

Tabelul 2.5.

x	**y**	**!x**	**x&&y**	**x||y**
adevărat (1)	adevărat (1)	fals (0)	adevărat (1)	adevărat (1)
adevărat (1)	fals (0)	fals (0)	fals (0)	adevărat (1)
fals (0)	adevărat (1)	adevărat (1)	fals (0)	adevărat (1)
fals (0)	fals (0)	adevărat (1)	fals (0)	fals (0)

Expresia `!expresie` are valoarea 0 (fals) dacă expresia-operand are o valoare diferită de zero şi valoarea unu (adevărat) dacă expresia-operand are valoarea zero.

Expresia `expresie1||expresie2` are valoarea diferită de 0 (true) dacă FIE expresie1, FIE expresie2 au valori diferite de zero.

Expresia `expresie1 && expresie2` are valoarea diferită de 0 (true) dacă AMBELE expresii-operand (expresie1 şi expresie2) au valori diferite de zero.

Exercițiu: Să se scrie următorul program şi să se urmărească rezultatele execuției acestuia.

```
#include <iostream.h>
int main()
{ int a=0, b=10, c=100, d=200; int rezult; rezult=a&&b;
cout<<"a&&b="<<rezult<<'\n';                  //Afişare a&&b=0
rezult=a||b; cout<<"a||b="<<rezult<<'\n';//Afişare a||b=1 (sau valoare nenula)
rezult=!a;cout<<"!a="<<rezult<<'\n';          //Afişare !a=1 (sau valoare nenula)
rezult=!b; cout<<"!b="<<rezult<<'\n';         //Afişare !b=0
rezult=(a>b) || (b>c);cout<<"(a>b) || (b>c)="<<rezult<<'\n';
//Afişare (a>b) || (b>c) =1(sau valoare nenula)
rezult=!(c<d);cout<<"!(c<d)="<<rezult<<'\n';//Afişare !(c>d)=0
rezult=(a-b)&&1;cout<<"(a-b)&&1="<<rezult<<'\n';
//Afişare (a-b)&&1 =1(sau valoare nenula)
rezult=d||b&&a;cout<<"d||b&&a="<<rezult<<'\n';//Afişare d||b&&a =1
return 0 ;
}// În evaluarea expresiilor din exemplu, s-au aplicat prioritățile operatorilor, indicate în tabelul. 2.6.
```

Tabelul 2.6.

Clasă de operatori	**Operatori**	**Asociativitate**
Unari	! - (unar) ++ --	de la dreapta la stânga
Multiplicativi	* / %	de la stânga la dreapta
Aditivi	+ -	de la stânga la dreapta
Atribuire	=	de la dreapta la stânga
relaţionali	< <= > >=	de la stânga la dreapta
de egalitate	== !=	de la stânga la dreapta
logici	&&	de la stânga la dreapta
logici	||	de la stânga la dreapta
atribuire şi aritmetici binari	= *= /= %= += -=	de la dreapta la stânga

Exerciţiu: Să se scrie un program care citeşte un număr real şi afişează 1 dacă numărul citit aparţine unui interval ale cărui limite sunt introduse tot de la tastatură, sau 0 în caz contrar.

```
#include <iostream.h>
int main()
{
double lmin, lmax, nr;cout<<"Numar=";cin>>nr;
cout<<”Limita inferioară a intervalului:”; cin>>lmin;
cout<<”Limita superioară a intervalului:”; cin>>lmax;
cout<<(nr>=lmin && nr<=lmax);
return 0 ;
}
```

- Operatori ***logici pe bit***

Operator	**Semnificaţie**	**Exemple**
~	Negaţie (cod complementar faţă de unu)	~a
&	AND (Conjuncţie, şi logic pe bit	a & 0377
|	OR **(**Disjuncţie, sau logic pe bit)	a | 0377
^	XOR (Sau exclusiv logic pe bit)	a^b
<<	Deplasare stânga	0377 << 2
>>	Deplasare dreapta	0377 >> 2

Aceşti operatori nu se aplică numerelor reale, ci numai datelor de tip întreg sau caracter. Primul operator este unar, ceilalţi binari. Operatorii acţionează la nivel de bit, la nivelul reprezentării interne (în binar), conform tabelelului 2.7.

Tabelul 2.7.

x	**y**	**x&y**	**x | y**	**x^y**	**~x**
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1

Operatorul **~** are aceeaşi prioritate ca şi ceilalţi operatori unari. El furnizează complementul faţă de unu al unui întreg, adică va schimba fiecare bit de pe 1 în zero şi invers. Operatorii de deplasare pe bit (**<<** şi **>>**) efectuează deplasarea la stânga sau la dreapta a operandului stâng, cu numărul de biţi indicaţi de operandul drept. Astfel, x<<2 deplasează biţii din x la stânga, cu două poziţii, introducând zero pe poziţiile rămase vacante.

Exemple:

```
int a=3;      //Reprezentare internă a lui a (pe 2 octeţi):      0000000000000011
int b=5;      //Reprezentare internă a lui b (pe 2 octeţi):      0000000000000101
int rez=~a;
cout<<"~"<<a<<'='<<rez<<'\n'; //~3= -4
//Complementul faţă de unu este: 1111111111111100 (în octal: 0177777774 (!a= - 4)
rez=a & b; cout<<a<<'&'<<b<<'='<<rez<<'\n'; //3&5=1
```

```
//a&b=0000000000000001 =1
rez=a^b; cout<<a<<'^'<<b<<'='<<rez; // 3^5= 6
//a ^b = 0000000000000110
rez=a|b; cout<<a<<'|'<<b<<'='<<rez;   //3|5= 7
//a | b = 0000000000000111
rez=a<<2; cout<<a<<"<<"<<3<<'='<<rez;       //3<<2=16=2*2^3
//a<<2= 0000000001100000
rez=5>>2; cout<<b<<">>"<<2<<'='<<rez;       //5>>2=1=5/2^2
//b>>2= 0000000000000001
```

Operatorul binar ^ îşi găseşte o utilizare tipică în expresii ca: `x&^077`, care maschează ultimii 6 biţi ai lui x pe zero.

Operatorul & este adesea utilizat în expresii ca `x&0177`, unde setează toţi biţii pe zero, cu excepţia celor de ordin inferior din x.

Operatorul | este utilizat în expresii ca: `x&MASK` , unde setează pe unu biţii care în x şi masca MASK sunt setaţi pe unu.

Operatorii logici pe bit & şi | sunt diferiţi de operatorii logici && şi || (pe cuvânt).

Deplasarea la stânga a unei date cu n poziţii este echivalentă cu înmulţirea valorii acesteia cu 2^n. Deplasarea la dreapta a unei date fără semn cu n poziţii este echivalentă cu împărţirea valorii acesteia cu 2^n.

Combinând operatorii logici pe bit cu operatorul de atribuire, se obţin operatorii:
`&=, ^=, |=, <<=, >>=.`

- Operatorul ***condiţional***

Este un operator ternar (necesită 3 operanzi), utilizat în construcţii de forma:

```
expresie1?expresie2:expresie3
```

Se evaluează *expresia1*. Dacă aceasta are o valoare diferită de zero, atunci tipul şi valoarea întregii expresii vor fi aceleaşi cu tipul şi valoarea *expresiei2*. Altfel (dacă expresie1 are valoarea zero), tipul şi valoarea întregii expresii vor fi aceleaşi cu tipul şi valoarea *expresiei3*. Deci operatorul condiţional este folosit pentru a atribui întregii expresii tipul şi valoarea expresiei2 sau a expresiei3, în funcţie de o anumită condiţie. Acest lucru este echivalent cu:

Dacă *expresie1* diferită de zero
Atunci evaluează *expresie2*
Altfel evaluează *expresie3*

Exemplu:

```
int semn=(x<0)?-1:1
```

Dacă `x<0,` atunci `semn=-1,` altfel `semn=1.`

- Operatorul ***virgulă***

Este utilizat în construcţii de forma:

```
expresie1 , expresie2
```

Operatorul virgulă forţează evaluarea unei expresii de la stânga la dreapta. Tipul şi valoarea întregii expresii este dată de tipul şi valoarea expresiei2. Operatorul virgulă este folosit în instrucţiunea ***for***. Operatorul virgulă are cea mai mică prioritate.

Exemplu:

```
int x, c, y;
cout<<"Astept val. ptr. y:"; cin>>y;
x=(c=y, c<=5);      /* c va primi valoarea lui y (citită); se verifică dacă c este mai mic sau
egal cu 5. Daca nu, x=0; daca da, x=1 sau x=valoare diferită de zero)*/
x++, y--;           //întâi este incrementat x, apoi este decrementat y
```

- Operatorul ***`sizeof()`***

Este un operator unar, care are ca rezultat numărul de octeţi pe care este memorată o dată de un anumit tip. Operandul este *un tip* sau *o dată (constantă sau variabilă) de un anumit tip*.

Exemple:

```
cout<<sizeof(int);          // afişează numărul de octeți pe care este memorat un întreg (2)
cout<<sizeof("ab6*");// afişează 5, nr. de octeți pe care este memorată constanta şir "ab6*"
```

- Operatorul ***(tip)***

Este un operator unar care apare în construcții numite "cast" şi converteşte tipul operandului său la tipul specificat între paranteze.

Exemple:

```
int a; (float) a;  // converteşte operandul a (care era de tip întreg) în float
```

În afara operatorilor prezentați, există şi alții, pe care îi vom enumera în continuare. Despre aceşti operatori vom discuta în capitolele viitoare, când cunoştințele acumulate vor permite acest lucru.

- Operatorul unar ***

Este operator unar, numit şi *operator de deferențiere*. Se aplică unei expresii de tip pointer şi este folosit pentru a accesa conținutul unei zone de memorie spre care pointează operatorul. Operatorii & (adresă) şi * sunt complementari.

Exemplu: Expresia *a este înlocuită cu valoarea de la adresa conținută în variabila pointer a.

- Operatorii ***paranteză***

Parantezele rotunde **( )** se utilizează în expresii, pentru schimbarea ordinii de efectuare a operațiilor, sau la apelul funcțiilor. La apelul funcțiilor, parantezele rotunde încadrează lista parametrilor efectivi. Din acest motiv, parantezele rotunde sunt numite şi *operatori de apel de funcție.*

Exemplu:

```
double sum(double a, double b);
/*declar. funcției sum, care primeşte 2 argumente reale(double) şi returnează o valoare tip double */
int main()
{
. . .
double a=sum(89.9, 56.6);   //apelul funcției sum, cu parametri efectivi 89.9 şi 56.6
int s0=6; double s1=(s0+9)/a; //folosirea parantezelor în expresii
. . .
return 0 ;
}
```

- Operatorii ***de indexare***

Operatorii de indexare sunt parantezele pătrate **[]**. Acestea includ expresii întregi care reprezintă indici ai unui tablou.

- Operatori ***de acces la membri structurilor***

Operatorii **::**, **.**, **->**, **.*** şi **->*** permit accesul la componentele unei structuri. Ei vor fi studiați în alt capitol.

În tabelul 2.8. sunt prezentați toți operatorii, grupați pe categorii, cu prioritățile lor şi regulile de asociativitate. Operatorii dintr-o categorie au aceeaşi prioritate.

Tabelul 2.8.

	Clasă de operatori	**Operatori**	**Asociativitate**
	Primari	**() [] . -> ::**	de la stânga la dreapta
	Unari	**! ~ ++ -- sizeof (tip)** **-**(unar) ***(deferențiere) **&**(referențiere)	de la stânga la dreapta
	Multiplicativi	*** / %**	de la stânga la dreapta
	Aditivi	**+ -**	de la stânga la dreapta
	Deplasare pe bit	**<< >>**	de la stânga la dreapta

	Relaționali	`<  <=  >  >=`	de la stânga la dreapta
	De egalitate	`==  !=`	de la stânga la dreapta
		`&` (ŞI logic pe bit)	de la stânga la dreapta
		`^` (XOR pe bit)	de la stânga la dreapta
		`|` (SAU logic pe bit)	de la stânga la dreapta
		`&&`	de la stânga la dreapta
		`||`	de la stânga la dreapta
	Condițional	`?:`	de la dreapta la stânga
	De atribuire	`=  +=  -=  *=  %=` `&=  ^=  |=  <<=  >>=`	de la dreapta la stânga
	Virgulă	`,`	de la stânga la dreapta

2.6.2. EXPRESII

Prin combinarea operanzilor şi a operatorilor se obțin ***expresii***. Tipul unei expresii este dat de tipul rezultatului obținut în urma evaluării acesteia. La evaluarea unei expresii se aplică regulile de prioritate şi asociativitate a operatorilor din expresie. Ordinea de aplicare a operatorilor poate fi schimbată prin folosirea parantezelor. La alcătuirea expresiilor, este indicată evitarea expresiilor în care un operand apare de mai multe ori.

2.6.3. CONVERSII DE TIP

La evaluarea expresiilor, se realizează conversii ale tipului operanzilor. Conversiile sunt:

- Automate;
- Cerute de evaluarea expresiilor;
- Cerute de programator (prin construcțiile cast), explicite.

Conversiile automate sunt realizate de către compilator:

```
char, short int ->   int
```

Ele sunt realizate de fiecare dată când într-o expresie apar operanzi de tipul char sau short int.

Conversiile cerute de evaluarea expresiilor sunt efectuate în cazurile în care în expresii apar operanzi de tipuri diferite. Înaintea aplicării operatorilor, se realizează conversia unuia sau a ambilor operanzi:

- Dacă un operand este de tip `long int`, celălalt este convertit la acelaşi tip; tipul expresiei este long int.
- Dacă un operand este de tipul `double`, celălalt este convertit la acelaşi tip; tipul expresiei este double.
- Dacă un operand este de tipul `float`, celălalt este convertit la acelaşi tip; tipul expresiei este float.

Conversiile explicite (cerute de programator) se realizează cu ajutorul construcțiilor cast.

Exemplu:

```
int x=3;    float y;          y=(float)x/2;
```

Înainte de a se efectua împărțirea celor 2 operanzi, operandul x (întreg) este convertit în număr real simplă precizie. După atribuire, valoarea lui y va fi 1.5. Dacă nu ar fi fost folosit operatorul de conversie în expresia y=x / 2, operanzii x şi 2 fiind întregi, rezultatul împărțirii este întreg, deci y ar fi avut valoarea 1.

ÎNTREBĂRI ŞI EXERCIȚII

Chestiuni teoretice

1. Ce reprezintă datele şi care sunt atributele lor?
2. Care sunt diferențele între constante şi variabile?
3. Cine determină tipul unei constante?
4. Ce sunt identificatorii?
5. Ce sunt directivele preprocesor?
6. Ce reprezinta variabilele?
7. Ce sunt constantele?
8. Enumerați tipurile simple de variabile.
9. Câte tipuri de directive preprocesor cunoasteți? Exemple.
10. Care este modalitatea de a interzice modificarea valorii unei variabile?
11. Ce loc ocupă declararea varibilelor în cadrul unui program sursă scris în limbajul C++?
12. Ce conțin fişierele header?
13. Ce tipuri de variabile se utilizează pentru datele numerice?
14. Care sunt calificatorii folosiți alături de tipurile de bază pentru obținerea tipurilor derivate de date?
15. Ce semnifică parantezele unghiulare < > care încadrează numele unui fişier header?
16. Care este diferența între constantele 35.2e-1 şi 3.52 ? Dar între "\t" şi '\t'?
17. Ce tip are constanta 6.44 ?
18. Care este diferența între operatorii = şi = = ?
19. Ce reprezintă caracterele "escape"?
20. Constante întregi.
21. Constante caracter.
22. Ce tipuri de conversii cunoaşteți?
23. Care sunt conversiile realizate în mod automat, de către compilator?
24. Constante şir de caractere.
25. Constante reale.
26. Ce operatori ternari cunoasteți?
27. Operatorul virgulă.
28. Operatorul sizeof.
29. Operatori aritmetici binari compuşi.
30. Operatorul de referențiere.
31. Operatori relaționali binari.

Chestiuni aplicative

1. Să se scrie declarațiile pentru definirea constantelor simbolice: pi, g (accelerația gravitațională), unghi_drept, dimensiune_MAX.
2. Care va fi rezultatul afişat pe ecran în urma execuției următoarelor secvențe de instrucțiuni:

```
double a=9/2; cout<<a*5<<'\n';
double a=9.7, b=5.6; cout<<(a+6<b)<<'\n';
double a=9/4; cout<<a*6<<'\n';
double x=3;int y=++x+5;cout<<y<<'\n';
int a=7;    cout<<(!a)<<'\n';
int a=10.5; cout<<a++<<'\n'; cout<<a<<'\n';
int a=7; cout<<++a<<'\n'; cout<<a<<'\n';
int a=10; cout<<a++<<'\n'; cout<<a<<'\n';
double a=7/2; cout<<a<<'\n';
int x=3; int y=x++-2; cout<<y<<'\n';
int x=3; int y=++x+5; cout<<y<<'\n';
double a=5.6, b=7.45; cout<<(a>b)<<'\n';
```

3. Să se verifice corectitudinea următoarelor secvențe. Pentru cele incorecte, explicați sursa erorilor.

```
double a=9.7, b=5.2; int c=(a+6<b)++; cout<<c<<'\n';
double a=7/5; double c=a*5++; cout<<c<<'\n';
double a=9.7, b=5.6; int c=(a%6<b)++; cout<<c<<'\n';
double a=5.6, b=7.45; cout<<++(a+5>b)<<'\n';
double a=9.8; double b=9.7; cout<<a%b<<'\n';
cout<<&(a+8)<<'\n';
int I=8; cout<<(I+10)++<<'\n';
double a=8.7; A=(a+8)/56; cout<<A<<'\n';
int x=3/5; int y=x++; char x='J'; cout<<"y="<<y<<'\n';
char a='X'; const int b=89; b+=8; cout<<"b="<<b<<" a="<<a<<'\n';
```

4. Să se scrie un program care afişează următoarele mesaje:
   - Sirul "este dupa-amiaza" este memorat pe .... octeti.
   - O marime intreaga este memorata pe ... octeti.
   - O marime reala, in simpla precizie este memorata pe ... octeti!
   - O marime reala, in dubla precizie este memorata pe ... byti!
   - Constanta caracter 'Q' memorata pe ... octeti!
   - Sirul "a\n\n" este memorat pe ... octei!
   - Sirul "\n" este memorat pe ... biti!
   - Caracterul '\' este memorat pe .... biti.

5. Să se evalueze expresiile, ştiind că: `int i=1;int j=2;int k=-7;double x=0;double y=2.3;`
   - `-i - 5 * j  >= k + 1`
   - `3 < j < 5`
   - `i + j + k == -2 * j`
   - `x && i || j - 3`

6. Ce operație logică şi ce mască trebuie să folosiți pentru a converti codurile ASCII ale literelor mici în litere mari? Dar pentru conversia inversă?
7. O deplasare la dreapta cu 3 biți este echivalentă cu o rotație la stânga cu câți biți?
8. Să se seteze pe 1 toți biții dintr-un octet, cu excepția bitului cel mai semnificativ.
9. Să se scrie un program care citeşte o valoare întreagă. Să se afişeze un mesaj care să indice dacă numărul citit este par sau impar.
10. Să se citeasca două valori întregi. Să se calculeze şi să se afişeze restul împărțirii celor două numere.