Modul 2 - Introducere în limbajul C. Elemente de bază ale limbajului

- **Limbaje de programare**
- **Limbajul C – istoric şi caracteristici**
- **Dezvoltarea de programe C**
- **Structura programelor C**
- **Elemente de bază ale limbajului C**
  - **Tipuri de date şi constante**
  - **Variabile şi operatori**
  - **Expresii**

Înainte de prezentarea sintaxei şi a elementelor de bază ale limbajului C, vom descrie, pe scurt, istoricul şi caracteristicile limbajului C. Limbajul C este un limbaj de bază pentru programatorii profesionişti. Are caracteristicile limbajelor de nivel înalt, fiind însă mult mai flexibil, aspect care poate fi exploatat de programatorii experimentați.

Limbaje de programare

Noțiunea de limbaj de programare este definită ca fiind ansamblul format de un vocabular şi un set de reguli gramaticale, necesar instruirii unui computer pentru a realiza anumite activități. Altfel spus, limbajul de programare este o notație sistematică prin care se descrie un proces de calcul. El dă posibilitatea programatorului să specifice în mod exact și amănunțit acțiunile pe care trebuie să le execute calculatorul, în ce ordine și cu ce date. Specificarea constă practic în întocmirea/scrierea programelor necesare ("programare").
Orice limbaj (natural sau artificial) presupune definirea unui set de reguli sintactice și semantice.
Sintaxa reprezintă un set de reguli ce guvernează alcătuirea propozițiilor dintr-un limbaj. În cazul limbajelor de programare echivalentul propoziției este programul. Semantica este un set de reguli ce determină „înțelesul” sau semnificația propozițiilor într-un limbaj.

Putem defini două mari categorii de limbaje de programare:

1. **Limbaje de nivel coborât, dependente de calculator. Aici avem:**
   - **Limbajul maşină**
   - **Limbajul de asamblare**

Limbajul maşină este limbajul pe care computerul îl înțelege în mod direct; în acest limbaj programele se scriu în cod binar ca succesiuni de 0 şi 1, fiecare instrucțiune din limbajul maşină fiind o combinație de 4 biți (exemple: LOAD-0000, ADD-0001). Pentru aceasta programatorul

trebuie să cunoască detaliat structura hardware a computerului, trebuie să gestioneze fără greşeală alocarea adreselor de memorie pentru un program. Pot apărea multe erori datorate concepției programului, sintaxei, suprapunerii adreselor de memorie, etc.
În plus față de limbajul maşină, la nivelul limbajului de asamblare, apar mnemonice pentru operațiuni şi simboluri pentru adrese.
Limbajul de asamblare presupune existența unui program numit assembler (asamblor) care să traducă programele în limbaj maşină. Asamblorul înlocuieşte codarea mnemonică (cum ar fi ADD) cu coduri binare corespunzătoare limbajului maşină şi alocă adrese de memorie pentru toate variabilele simbolice utilizate (A, B, C) şi se asigură că aceste adrese sunt distincte. Astfel prin uşurarea procesului de programare s-a introdus un nou nivel de procesare pentru computer. Astăzi limbajele de asamblare sunt încă utilizate pentru unele programe critice deoarece aceste limbaje conferă programatorului un control foarte precis a ceea ce se întâmplă în computer.

Limbaje de nivel înalt, independente de structura calculatorului. Câteva exemple în ordinea apariției lor:

- **Fortran (FORmula TRANslation) – 1955, IBM, pentru probleme tehnico-ştiințifice**
- **Cobol – 1959, pentru probleme economice**
- **Programarea structurată apare prin anii 70 (Pascal, C, etc)**
- **Programare orientată pe obiecte prin anii 80 (C++, Java, etc)**

Bazele programării structurate au fost puse de Dijkstra şi Hoare. Este o modalitate de programare în care abordarea este top-down: descompunerea problemei complexe în subprobleme mai simple, numite module. Tot aici apare şi teorema de structură a lui Bohm şi Jacopini, pe care am folosit-o deja în elaborarea algoritmilor. Această teoremă spune că orice algoritm poate fi compus din numai trei structuri de calcul:

1. **structura secvențială - secvența;**
2. **structura alternativă - decizia;**
3. **structura repetitivă - ciclul.**

Ideea programării orientate pe obiecte este aceea de a crea programele sub forma unei colecții de obiecte, unități individuale de cod, care interacționează unele cu altele, în loc de simple liste de instrucțiuni sau de apeluri de proceduri aşa cum apare în programarea structurată.
Obiectele POO sunt de obicei reprezentări ale obiectelor din viața reală, astfel încât programele realizate prin tehnica POO sunt mai ușor de înțeles, de depanat și de extins decât programele procedurale. Aceasta este adevărată mai ales în cazul proiectelor software complexe și de dimensiuni mari, care se gestionează făcând apel la ingineria programării.

Dupa modul de “traducere” limbajele de programare pot fi:

- Limbaje compilate: C, C++, Pascal, Java;
- Limbaje interpretate: PHP, Javascript, Prolog, Matlab

La limbajele compilate translatorul se numeşte compilator, iar mecanismul folosirii unui astfel de limbaj este următorul: programul sursă este tradus integral în limbajul maşină, iar rezultatul este un fişier executabil, viteza de execuție a acestuia fiind ridicată, întrucât programul este deja transpus în întregime în cod maşină.
La limbajele interpretate translatorul poartă denumirea de interpretor şi funcționează în felul următor: preia prima comandă din codul sursă, o traduce în limbajul maşină şi o execută, apoi a doua comandă şi tot aşa.
Multe limbaje moderne combină compilarea cu interpretarea: codul sursă este *compilat* într-un limbaj binar numit *bytecode*, care la rulare este *interpretat* de către o mașină virtuală. De remarcat faptul că unele interpretoare de limbaje pot folosi compilatoare așa-numite *just-in-time*, care transformă codul în limbaj mașină chiar înaintea executării.

Limbajul C - Scurt istoric

Limbajul C a fost proiectat şi implementat de Dennis Ritchie între anii 1969 şi 1973 la AT&T Bells Laboratories, pentru programe de sistem (care până atunci erau dezvoltate doar în limbaje de asamblare). A fost numit "C" deoarece este un succesor al limbajului B, limbaj creat de Ken Thompson.
Originile limbajului C sunt strâns legate de cele ale UNIX-ului, care inițial fusese implementat în limbajul de asamblare PDP-7 tot de Ritchie şi Thompson. Deoarece limbajul B nu putea folosi eficient unele din facilitățile PDP-11, pe care vroiau să porteze UNIX-ul, au avut nevoie de un limbaj simplu și portabil pentru scrierea nucleului sistemului de operare UNIX.

- În 1973, sistemul de operare UNIX este aproape în totalitate rescris în C, fiind astfel unul din primele sisteme de operare implementate în alt limbaj decât cel de asamblare.
- Cartea de referință care defineşte un standard minim este scrisă de Brian W. Kernighan şi Dennis Ritchie, "The C Programming Language" şi a apărut în 1978 (Prentice Hall).
- Între 1983-1989 a fost dezvoltat un standard international -- ANSI C (ANSI - American National Standards Institute).
- În 1990, ANSI C standard a fost adoptat de International Organization for Standardization (ISO) ca ISO/IEC 9899:1990, care mai este numit şi C90. Acest standard este suportat de compilatoarele curente de C. Majoritatea codului C scris astăzi are la bază acest standard. Orice program scris doar în ANSI C Standard va rula corect pe orice platformă ce are instalată o variantă de C.
- În anii următori au fost dezvoltate medii de programare C performante sub UNIX şi DOS, care au contribuit la utilizarea masivă a limbajului.
- Necesitatea grupării structurilor de date cu operațiile care prelucrează respectivele date a dus la apariția noțiunilor de o*biect* sau *clasă*, în 1980, Bjarne Stroustrup elaborează "*C with Classes*".
- Acest limbaj a dus la îmbunătățirea C-ului prin adăugarea unor noi facilități, printre care şi lucrul cu clase. În vara 1983, C-with-classes a pătruns şi în lumea academică şi a instituțiilor de cercetare. Astfel, acest limbaj a putut să evolueze datorită experienței acumulate de către utilizatorii săi. Denumirea finală a acestui limbaj a fost *C++*.
- Succesul extraordinar pe care l-a avut limbajul C++ a fost asigurat de faptul că a extins cel mai popular limbaj al momentului, C.

- Programele scrise în C funcționează şi în C++, şi ele pot fi transformate în C++ cu eforturi minime.
- Cea mai recentă etapă în evoluția acestui limbaj o constituie limbajul *JAVA* (*1995)* realizat de firma *SUN*, firmă cumpărată în 2010 de către *Oracle*.

În concluzie, sintaxa limbajului C a stat la baza multor limbaje create ulterior și încă populare azi: C++, Java, JavaScript, C#.

Caracteristicile limbajului C

Caracteristicile limbajului C, care i-au determinat popularitatea, sunt prezentate pe scurt mai jos şi vor fi analizate pe parcursul cursului:

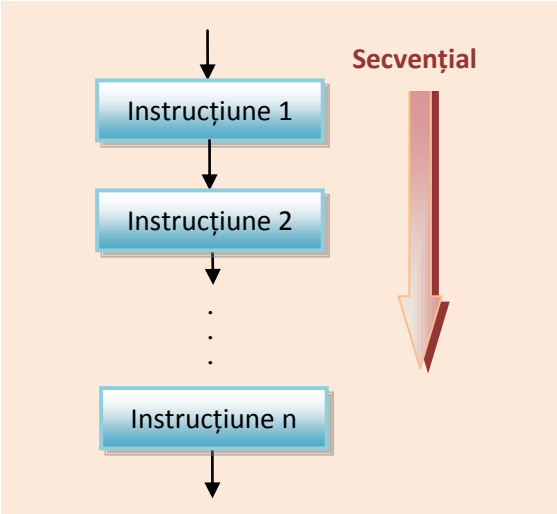
- **limbaj de nivel mediu, portabil, structurat, flexibil**
- **produce programe eficiente (lungimea codului scăzută, viteză de execuție mare)**
- **set bogat de operatori**
- **multiple facilități de reprezentare şi prelucrare a datelor**
- **utilizare extensivă a apelurilor de funcții şi a pointerilor**
- **verificare mai scăzută a tipurilor - *loose typing* - spre deosebire de PASCAL**
- **permite programarea la nivel scăzut - *low-level* - , apropiat de hardware**

Este utilizat în multiple aplicații, în care nu predomină caracterul numeric:

- programe de sistem
- proiectare asistată de calculator
- grafică
- prelucrare de imagini
- aplicații de inteligență artificială.

Procesul dezvoltării unui program C

Un program este descrierea precisă şi concisă a unui algoritm într-un anumit limbaj de programare. Putem spune că un program este o secvență de instrucțiuni care se execută una după cealaltă – secvențial:



Un program are un caracter general şi de aceea are nevoie de date inițiale care particularizează programul pentru o situație concretă. Rezultatele produse de un program pe baza datelor inițiale sunt de obicei afişate pe ecran. Datele de intrare se introduc manual de la tastatură sau se citesc din fişiere disc.
Operațiile uzuale din limbajele de programare sunt operații de prelucrare (calcule, comparații etc) şi operații de intrare-ieşire (de citire-scriere). Aceste operații sunt exprimate prin instrucțiuni ale limbajului sau prin apelarea unor funcții standard predefinite (de bibliotecă). Desfăşurarea în timp a instrucțiunilor de prelucrare şi de intrare-ieşire este controlată prin instrucțiuni repetitive (de ciclare) şi de decizie (de comparație).
Fiecare limbaj de programare are reguli gramaticale precise, a căror respectare este verificată de programul compilator (translator).

Pentru a dezvolta un program C putem folosi un mediu integrat de dezvoltare (IDE - Interactive Development Environment) precum CodeBlocks, DevCpp, MS Visual Studio, Eclipse sau Netbeans, sau o putem face în mod linie de comandă.

Etapele necesare pentru dezvoltarea unui program C sunt următoarele:
Etapa 1: Codificarea algoritmului într-un limbaj de programare - Edit
Această codificare se va realiza într-un program sursă.
Se salvează fişierul sursă, de exemplu cu numele "Hello.c". Un fişier C++ trebuie salvat cu extensia ".cpp", iar un fişier C cu extensia ".c". Trebuie ales un nume care să reflecte scopul programului.
Această etapă se poate realiza utilizând comenzile respective ale IDE-ului – de exemplu N*ew*, S*ave* - sau un editor de texte – *Notepad*.

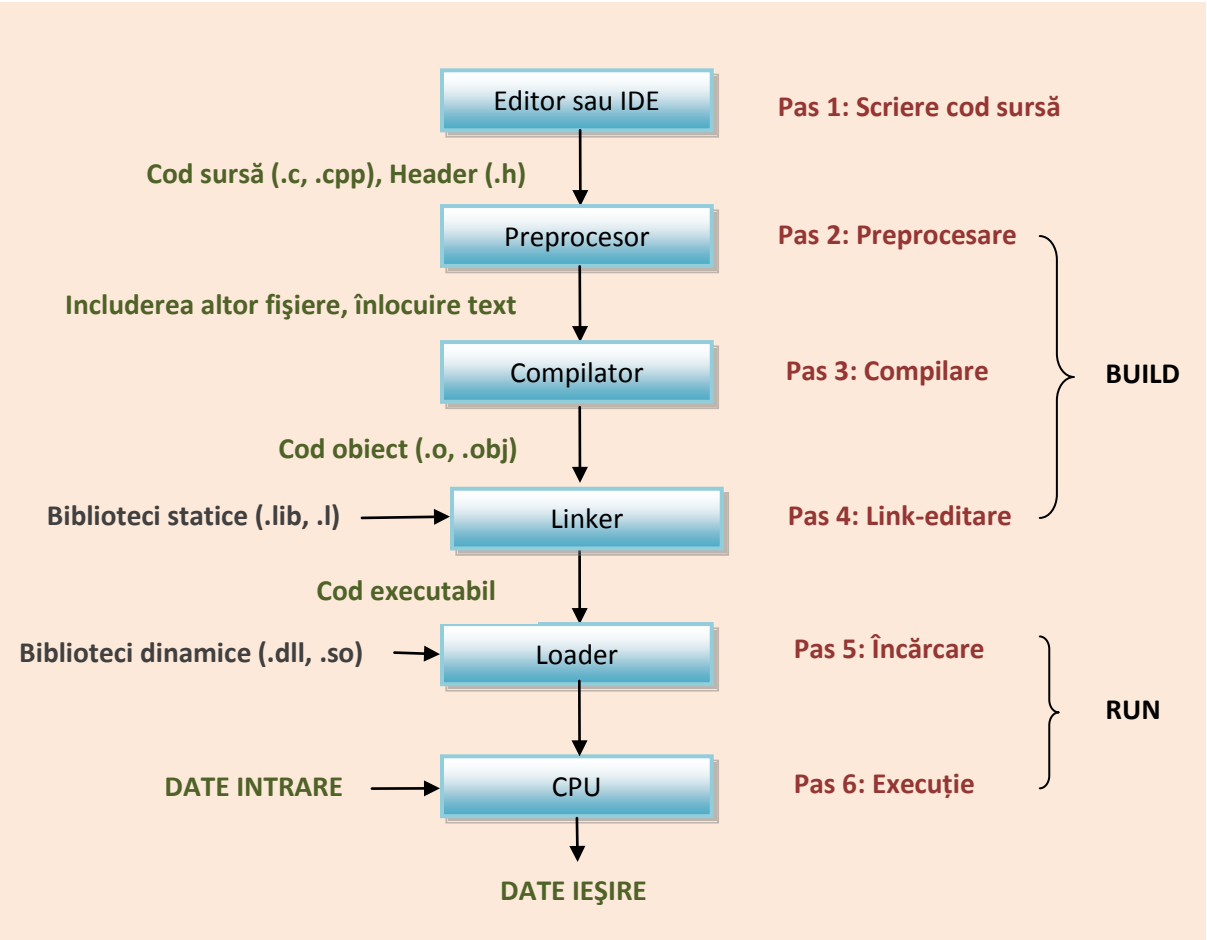
Etapa 2: Compilarea şi link-editarea - Build
În acest pas, prin compilare şi link-editare, programul sursă este tradus integral în limbajul maşină, iar rezultatul este un fişier executabil.
Această etapă se poate realiza utilizând comenzile respective ale IDE-ului – de exemplu *Compile*, *Build* - sau linie de comandă, cu compilatorul GNU GCC:

În Windows (CMD shell) - generează fişierul executabil *Hello.exe*	**> gcc Hello.c -o Hello.exe**
În Unix or Mac (Bash shell) - generează fişierul executabil *Hello*	**$ gcc Hello.c -o Hello**

Etapa 3: Execuția programului - Run
Această etapă se poate realiza utilizând comenzile respective ale IDE-ului – *Run* - sau linie de comandă. De exemplu, lansarea în execuție în mod linie de comandă se poate face astfel:

În Windows (CMD shell) - Rulare "Hello.exe" (.exe este opțional)	**> Hello**
În Unix or Mac (Bash shell) - Rulare "Hello"	**$ ./Hello**

Detaliarea acestor etape este reflectată în figura următoare: în stânga avem tipul de fişiere, la mijloc cine realizează acel pas, pentru ca în partea dreaptă a figurii să apară paşii efectivi:



Tabelul conține o descriere sintetică a paşilor detaliați:

Pas	Descriere
1	Crearea fişierului sursă (cu extensia *.c* sau *.cpp* în cazul limbajului C/C++) folosind un editor de texte sau un IDE. Rezultă un fişier sursă. Poate fi creat şi un fişier antet (*header*), cu extensia *.h*
2	Preprocesarea codului sursă în concordanță cu directivele de preprocesare *(#include*, *#define* în C). Aceste directive indică operații (includerea unui alt fişier, înlocuire de text etc) care se vor realiza ÎNAINTE de compilare.
3	Compilarea codului sursă preprocesat. Rezultă un fişier obiect (*.obj*, *.o*).
4	Legarea (linked-itarea) codului obiect rezultat cu alte fişiere obiect şi biblioteci pentru a furniza fişierul executabil (.exe).
5	Încărcarea codului executabil în memoria calculatorului (RAM).

6	Rularea codului executabil.

Pentru mai multe detalii vom include aici tutoriale de utilizare a unor medii integrate de dezvoltare C:

Tutorial CodeBlocks
Tutorial NetBeans

Structura unui program C

Un program C este compus dintr-o ierarhie de funcții, orice program trebuind să conțină cel puțin funcția *main*, prima care se execută la lansarea programului C. Funcțiile pot face parte dintr-un singur fişier sursă sau din mai multe fişiere sursă. Un fişier sursă C este un fişier text care conține o succesiune de funcții şi, eventual, declarații de variabile.

Structura unui program C este, în principiu, următoarea:

- **directive preprocesor**
- **definiții de tipuri**
- **prototipuri de funcții - tipul funcției şi tipurile parametrilor funcției**
- **definiții de variabile externe**
- **definiții de funcții**

O funcție C are un antet şi un bloc de instrucțiuni (prin care se execută anumite acțiuni) încadrat de acolade. În interiorul unei funcții există de obicei declarații de variabile precum şi alte blocuri de instrucțiuni.

Primul program C

Urmează un exemplu de program C minimal, ce afişează un text – *Hello World!* – pe ecran. Exemplul conține o funcție *main* cu o singură instrucțiune (apelul funcției *printf*) şi nu conține declarații şi este scris în trei variante diferite pentru funcția *main*:

Varianta	Cod

1	#include<stdio.h>
	void main(void)
	{
	printf(“Hello World!”);
	}

2 – fără void în antetul main-ului	#include<stdio.h>
	void main()
	{
	printf(“Hello World!”);
	}

3 - fără warning la compilare în Code::Blocks	#include<stdio.h>
	int main()
	{
	printf("Hello World!");
	return 0;
	}

Execuția programului începe cu prima linie din *main*. Cuvântul din fața funcției *main* reprezintă tipul funcției (*void* arată că această funcție nu transmite nici un rezultat prin numele său, *int* arată că trimite un rezultat de tip întreg, prin instrucțiunea *return*). Parantezele care urmează cuvântului *main* arată că *main* este numele unei funcții (şi nu este numele unei variabile), dar o funcție fără parametri (acel void care poate lipsi – varianta 2).
Acoladele sunt necesare pentru a delimita corpul unei funcții, corp care este un bloc de instrucțiuni şi declarații. Funcția *printf* realizează afişarea pe ecran a textului *Hello World!*.
Directiva *#include* este o directivă de preprocesare, permite includerea unor funcții de bibliotecă, necesară pentru folosirea funcției *printf*.
Alte observații care pot fi făcute:

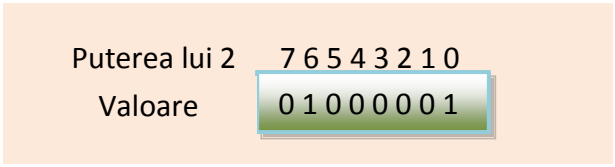
- cuvintele cheie sunt scrise cu litere mici
- instrucțiunile se termină cu ';'
- şirurile de caractere sunt incluse între ghilimele
- limbajul C este **case sensitive** – face diferență între literele mici şi literele mari (*a* este diferit de *A*)
- \n folosit în funcția *printf* poziționează cursorul la începutul liniei următoare

Elemente de bază ale limbajului C

Elementele de bază ale limbajului C sunt următoarele:

- **Alfabetul şi atomii lexicali**
- **Identificatorii**
- **Cuvintele cheie**
- **Tipurile de date**
- **Constantele şi variabilele**
- **Comentariile**
- **Operatorii şi expresiile**

Alfabetul limbajului
Caracterele se codifică conform *codului ASCII* (American Standard Code for Information Interchange), codificarea realizîndu-se pe 8 biți (un octet). Sunt 256 (codificate de la 0 la 255) de caractere în codul ASCII, alfabetul cuprinzând simboluri grafice şi simboluri fără corespondent grafic.
De exemplu, caracterul *A* cu codul ASCII 65 este codificat pe 8 biți – 1 octet astfel:



Pentru a înțelege mai bine ce înseamnă mod de codificare următoarele noțiuni auxiliare sunt descrise în anexa **Tutorial despre reprezentarea datelor**.

- Octet
- reprezentare în baza 2, 8, 10, 16

Limbajul C este case-sensitive (se face diferență între litere mici şi litere mari).
O altă observație este aceea că spațiul are codul mai mic decat simbolurile grafice (32), iar cifrele (în ordine crescătoare), literele mari şi literele mici (în ordine alfabetică) ocupă câte trei zone compacte - cu intervale între ele.
Pentru mai multe detalii legate de codul ASCII vă rugăm să urmați unul din următoarele link-uri:
Tabel Coduri ASCII
Tabel Coduri ASCII 2

Atomii lexicali pot fi:

- **identificatori**
- **constante (explicite) - numerice, caracter, şir**
- **operatori**
- **semne de punctuație.**

Un atom lexical trebuie scris integral pe o linie şi nu se poate extinde pe mai multe linii. În cadrul unui atom lexical nu se pot folosi spații albe (excepție fac spațiile dintr-o constantă şir), putem însă folosi caracterul '_'.
Respectarea acestei reguli poate fi mai dificilă în cazul unor şiruri constante lungi, dar există posibilitatea prelungirii unui şir constant de pe o linie pe alta folosind caracterul '\'.

Atomii lexicali sunt separați prin separatori, care pot fi:

- **spațiul**
- **caracterul de tabulare orizontală *\t***
- **terminatorul de linie *\n***
- **comentariul**

Un program este adresat unui calculator pentru a i se cere efectuarea unor operații, dar programul trebuie citit şi înțeles şi de către oameni; de aceea se folosesc **comentarii** care explică de ce se fac anumite acțiuni.
Inițial în limbajul C a fost un singur tip de comentariu, comentariul multilinie, care începea cu secvența /* şi se termina cu secvența */. Ulterior s-au adoptat şi comentariile din C++, care încep cu secvența // şi se termină la sfârşitul liniei care conține acest comentariu.

Identificatorii pot fi :

- **nume utilizator - nume de variabile, constante simbolice, funcții, tipuri, structuri, uniuni - este bine sa fie alese cât mai sugestiv pentru scopul utilizării;**
- **cuvinte cheie ale limbajului C - pot fi folosite doar cu înțelesul cu care au fost definite**
- **cuvinte rezervate - înțelesul poate fi modificat, de evitat acest lucru;**

Limbajul C impune următoarele reguli asupra identificatorilor:

- Un identificator este o secvență de caractere, de o lungime maximă ce depinde de compilator (în general are maxim 255 de caractere). Secvența poate conține litere mici şi mari(a-z, A-Z), cifre (0-9) şi simbolul '_' (underscore, liniuța de subliniere).
- Primul caracter trebuie să fie o literă sau '_'. Pentru că multe nume rezervate de compilator, invizibile programatorului, încep cu '_', este indicat a nu utiliza '_' pentru începutul numelor utilizator.
- Un identificator nu poate fi un cuvânt cheie (int, double, if, else, for).
- Deoarece limbajul C este case-sensitive identificatorii sunt şi ei la rândul lor case-sensitive. Astfel, *suma* nu este *Suma* şi nici *SUMA*.
- Identificatorii sunt atomi lexicali, în cadrul lor nu e permisă utilizarea spațiilor albe şi a altor caractere speciale (ca +, -, *, /, @, &, virgulă, etc.) putem însă folosi caracterul '_'.

Exemple de identificatori: *suma, _produs, x2, X5a, PI, functia_gauss* etc.

Recomandări:

- Este important să alegem un nume care este *self-descriptive*, care reflectă foarte bine înțelesul identificatorului respectiv, de exemplu, vom spune nrStudenti sau numarDeStudenti.
- Încercați să nu folosiți identificatori care nu spun nimic (lipsiți de un sens clar): a, b, c, d, i, j, k, i1, j99.
- Evitați numele de un singur caracter care sunt mai uşor de folosit dar de cele mai multe ori lipsite de vreun înțeles. Acest lucru este însă utilizat dacă sunt nume commune cum ar fi x, y, z pentru coordinate sau i, j pentru indici.
- Nu folosiți nume foarte lungi, sunt greu de utilizat, încercați să optimizați lungimea numelui cu înțelesul acestuia.
- Folosiți singularul şi pluralul pentru a diferenția. De exemplu, putem folosi numele linie pentru a ne referi la numărul unei singure linii şi numele linii pentru a ne referi la mai multe linii (de exemplu un vector de linii).

Cuvintele cheie se folosesc în declarații şi instrucțiuni şi nu pot fi folosite ca nume de variabile sau de funcții (sunt cuvinte rezervate ale limbajului). Exemple de cuvinte cheie:

int	**extern**	**double**
char	**register**	**float**

unsigned	**typedef**	**static**
do	**else**	**for**
while	**struct**	**goto**
switch	**union**	**return**
case	**sizeof**	**default**
short	**break**	**if**
long	**auto**	**continue**
Standardul ANSI C a mai adăugat:		
signed	**const**	**void**
enum	**volatile**	

Numele de funcții standard (*scanf*, *printf*, *sqrt* etc.) nu sunt cuvinte cheie, dar nu se recomandă utilizarea lor în alte scopuri, deoarece aceasta ar produce schimbarea sensului inițial, atribuit în toate versiunile limbajului.

Tipuri de date

În C există tipuri de date fundamentale şi tipuri de date derivate. Sunt prezentate mai jos principale tipuri
de date şi numărul de octeți pe care acestea le ocupă pe un sistem Unix de 32 de biți.

Tipurile de date fundamentale sunt:

- caracter (char – 1 octet)
- întreg (int – 4 octeți)
- virgulă mobilă (float – 4 octeți)
- virgulă mobilă dublă precizie (double – 8 octeți)
- nedefinit (void)

Tipurile de date derivate sunt:

- tipuri structurate (tablouri, structuri)
- tipul pointer (4 octeți)

Pentru a utiliza eficient memoria şi a satisface necesitățile unei multitudini de aplicații există în C mai multe tipuri de întregi şi respectiv de reali, ce diferă prin memoria alocată şi deci prin numărul de cifre ce pot fi memorate şi prin domeniul de valori.
Tipurile întregi sunt: *char , short , int , long , long long*. Implicit toate numerele întregi sunt numere cu semn (*signed*), dar prin folosirea cuvântului cheie *unsigned* la declararea lor se poate cere interpretarea ca numere fără semn. Utilizarea cuvintelor *long* sau *short* face ca tipul respectiv să aibă un domeniu mai mare, respectiv mai mic de valori.

Tipuri în virgulă mobilă (reale): Există două tipuri, *float* şi *double*, pentru numere reale reprezentate în virgulă mobilă cu simplă şi respectiv dublă precizie. Unele implementări suportă şi *long double* (numai cu semn). Trebuie semnalat faptul că nu toate numerele reale pot fi reprezentate, întrucât memorarea valorilor reale, fiind realizată pe un număr anume de biți, nu poate reține decât o parte dintre cifrele semnificative. Deci numai anumite valori reale au reprezentarea exactă în calculator, restul confundându-se cu reprezentarea cea mai apropiată.

Caractere: Caracterele (exemplu 'a', 'Z', '0', '9') sunt codificate ASCII sub formă de valori întregi, şi păstrate în tipul char. De exemplu, caracterul '0' este 48 (decimal, adică în baza 10) sau 30H (hexadecimal – baza 16); caracterul 'A' este 65 (decimal) sau 41H (hexadecimal); caracterul 'a' este 97 (decimal) sau 61H (hexadecimal). Se observă că tipul char poate fi interpretat ca şi character în codul ASCII sau ca un întreg fără semn în domeniul 0 – 255.
Standardul C din 1999 prevede şi tipul boolean _Bool (sau bool) pe un octet.
Reprezentarea internă şi numărul de octeți necesari pentru fiecare tip nu sunt reglementate de standardul limbajului C, dar limitele fiecărui tip pentru o anumită implementare a limbajului pot fi aflate din fişierul antet "limits.h" (care conține şi nume simbolice pentru aceste limite - INT_MAX şi INT_MIN) sau utilizând operatorul *sizeof*.
De obicei tipul int ocupă 4 octeți, iar valoarea maximă este de cca. 10 cifre zecimale pentru tipul int. Depăşirile la operații cu întregi de orice lungime nu sunt semnalate deşi rezultatele sunt incorecte în caz de depăşire.
Reprezentarea numerelor reale în diferite versiuni ale limbajului C este mai uniformă deoarece urmează un standard IEEE de reprezentare în virgulă mobilă. Pentru tipul *float* domeniul de valori este între 10E-38 şi 10E+38 iar precizia este de 6 cifre zecimale exacte. Pentru tipul *double* domeniul de valori este între 10E-308 şi 10E+308 iar precizia este de 15 cifre zecimale.

Tabelul următor prezintă dimensiunea tipică, valorile minimă şi respective maximă pentru tipurile primitive (în cazul general). Încă o dată, dimensiunea este dependentă de implementarea C folosită.

Categorie	Tip	Descriere	Octeți	Valoare minimă	Valoare Maximă
Numere Întregi	**int** **signed int**	Întreg cu semn (cel puțin 16 biți)	4 (2)	-2147483648	2147483647 (=2^{31}-1)
	unsigned int	Întreg fără semn (cel puțin 16 biți)	4 (2)	0	4294967295 (=2^{32}-1)
	char	Caracter (poate fi cu semn sau fără semn, depinde de implementare)	1		
	signed char	Caracter sau întreg mic cu semn (garantează că e cu semn)	1	-128	127 (=2^{7}-1)

	unsigned char	Caracter or sau întreg mic fără semn (garantează că e fără semn)	1	0	255 ($=2^8-1$)
	short **short int** **signed short** **signed short int**	Întreg scurt cu semn (cel puțin 16 biți)	2	-32768	32767 ($=2^{15}-1$)
	unsigned short **unsigned short int**	Întreg scurt fără semn (cel puțin 16 biți)	2	0	65535 ($=2^{16}-1$)
	long **long int** **signed long** **signed long int**	Întreg lung cu semn (cel puțin 32 biți)	4 (8)	-2147483648	2147483647 ($=2^{31}-1$)
	unsigned long **unsigned long int**	Întreg lung fără semn (cel puțin 32 biți)	4 (8)	0	4294967295 ($=2^{32}-1$)
	long long **long long int** **signed long long** **signed long long int**	Întreg foarte lung cu semn (cel puțin 64 biți) (de la standardul C99)	8	-2^{63}	($=2^{63}-1$)
	unsigned long long **unsigned long long int**	Întreg foarte lung fără semn (cel puțin 64 biți) (de la standardul C99)	8	0	$2^{64}-1$
Numere Reale	**float**	Număr în virgulă mobilă, ≈7 cifre(IEEE 754 format virgulă mobilă simplă precizie)	4	3.4e-38	3.4e38
	double	Număr în virgulă mobilă dublă precizie, ≈15 cifre(IEEE 754 format virgulă mobilă dublă precizie)	8	1.7e-308	1.7e308
	long double	Număr lung în virgulă mobilă dublă precizie, ≈19 cifre(IEEE 754 format virgulă mobilă cvadruplă	12 (8)	3.4E-4932	1.1E4932

		precizie)			
Numere booleene	**bool**	Valoare booleană care poate fi fie true fie false (de la standardul C99)	1	false (0)	true (1 sau diferit de zero)

Valori corespunzătoare tipurilor fundamentale

Aceste valori constante, ca 123, -456, 3.14, 'a', "Hello", pot fi atribuite direct unei variabile sau pot fi folosite ca parte componentă a unei expresii.

Valorile întregi se reprezintă implicit în baza 10 şi sunt de tipul signed care este acoperitor pentru reprezentarea lor.

- Pentru reprezentarea constantelor fără semn se foloseşte sufixul u sau U
- Constantă întreagă poate fi precedată de semnul plus (+) sau minus.
- Se poate folosi prefixul '0' (zero) pentru a arăta că acea valoare este reprezentată în octal, prefixul '0x' pentru o valoare în hexadecimal şi prefixul '0b' pentru o valoare binară (în unele compilatoare).
- Un întreg long este identificat prin folosirea sufixului 'L' sau 'l'. Un long long int este identificat prin folosirea sufixului 'LL'. Putem folosi sufixul 'U' sau 'u' pentru unsigned int, 'UL' pentru unsigned long, şi 'ULL' pentru unsigned long long int.
- Pentru valori constante de tip short nu e nevoie de sufix.

Exemple:

```
-10000            // int
65000             // long
32780             // long
32780u            // unsigned int
1234;             // Decimal
01234;            // Octal 1234, Decimal 2322
0x1abc;           // hexadecimal 1ABC, decimal 15274
0b10001001;       // binar (doar în unele compilatoare)
12345678L;        // Sufix 'L' pentru long
123UL;            // int 123 auto-cast la long 123L
987654321LL;      // sufix 'LL' pentru long long int
```

Valorile reale

Sunt implicit de tipul double; sufixul f sau F aplicat unei constante, o face de tipul float, iar l sau L de tipul long double.
Un număr cu parte fracționară, ca 55.66 sau -33.442, este tratat implicit ca double.
O constanta reală se poate reprezenta şi sub forma ştiințifică.

Exemplu:

```
1.2e3       // 1.2*10^3
-5.5E-6     // -5.5*10^-6
```

unde E denotă exponentul puterii lui 10. Mantisa, partea de dinaintea lui E poate fi precedată de semnul plus (+) sau minus (-).

- mantisa - *parte_intreagă.parte_zecimala* (oricare din cele două părți poate lipsi, dar nu ambele)
- exponent - e*val_exponent* sau E*val_exponent*, unde *val_exponent* este un număr întreg.

Valoarea constantei este produsul dintre mantisa si 10 la puterea data de exponent.

In tabelul de mai jos apar cateva exemple de constante reale:

Constante de tip float	Constante de tip double	Constante de tip long double
1.f	1.	1.L
.241f	.241	.241l
-12.5e5f	-12.5e5	-12.5e5l
98.E-12f	98.E-12	98.E-12L

Valori caracter

Valorile de tip caracter se reprezintă pe un octet şi au ca valoare codul ASCII al caracterului respectiv.

În reprezentarea lor se foloseşte caracterul apostrof : ‘A’ (cod ASCII 65), ‘b’, ‘+’.

Pentru caractere speciale se foloseşte caracterul \.

Exemple:

```
'\''  - pentru apostrof
'\\'  - pentru backslash
Este greşit:   '''   sau   '\'
```

Tot aici intră şi secvențele escape ce se pot scrie sub forma unei secvențe ce începe cu ‘\’,

Constantele caracter pot fi tratate în operațiile aritmetice ca un întreg cu semn pe 8 biți. Cu alte cuvinte, char şi signed int pe 8 biți sunt interschimbabile. De asemenea, putem atribui un întreg în domeniul [-128, 127] unei variabile de tip char şi [0, 255] unui unsigned char.

Caracterele non-tipăribile şi caracterele de control pot fi reprezentate printr-o secvență *escape*, care începe cu un back-slash (\) urmat de o literă (‘\n’ = new line , ‘\t’ =tab , ‘\b’ = backspace etc), sau de codul numeric al caracterului în octal sau în hexazecimal (\012 = \0x0a = 10 este codul pentru caracterul de trecere la linie nouă ‘\n’).

Cele mai utilizate secvențe escape sunt:

Secvența escape	Descriere	Hexa (Decimal)
\n	Linie nouă (Line feed)	0AH (10D)

\r	Carriage-return	0DH (13D)
\t	Tab	09H (9D)
\"	Ghilimele	22H (34D)
\'	Apostrof	27H (39D)
\\	Back-slash	5CH (92D)

Valori şir de caractere

Valorile şir de caractere sunt compuse din zero sau mai multe caractere precizate între ghilimele.

Exemple:

```
"Hello, world!"
"The sum is "
""
```

Fiecare caracter din şir poate fi un *simbol grafic*, o *secvență escape* sau un *cod ASCII* (în octal sau hexazecimal). Spațiul ocupat este un număr de octeți cu unu mai mare decât numărul caracterelor din şir, ultimul octet fiind rezervat pentru *terminatorul de şir:* caracterul cu codul ASCII 0, adica '\0'. Dacă se doreşte ca şi caracterul ghilimele să faca parte din şir, el trebuie precedat de \.

Exemple:

```
"CURS"   -   "\x43URS"
// scrieri echivalente ale unui şir ce ocupă 5 octeți

"1a24\t"   -  "\x31\x61\x32\x34\11"
//scrieri echivalente ale unui sir ce ocupă 6 octeți

 "'\""
//şir ce conține caracterele '" şi terminatorul - ocupă 3 octeți
```

Tipul void nu are constante (valori) şi este utilizat atunci când funcțiile nu întorc valori sau când funcțiile nu au parametri:

```
void f ( int a)
{
      if (a) a = a / 2;
}

sau când funcțiile nu au parametri:

void f (void);
```

```
// echivalent cu void f();

int f (void);
//echivalent cu int f();
```

Constante

Constantele pot fi: predefinite sau definite de utilizator.
Literele mari se folosesc în numele unor constante simbolice predefinite: EOF, M_PI, INT_MAX, INT_MIN. Aceste constante simbolice sunt definite în fişiere header (de tip "h") : EOF în "stdio.h", M_PI în "math.h".

Definirea unei constante simbolice se face utilizând directiva *#define* astfel:

#define identificator [text]

Exemple:

```
#define begin {    // unde apare begin acesta va fi înlocuit cu {
#define end }      // unde apare end acesta va fi înlocuit cu }
#define N 100      // unde apare N acesta va fi înlocuit cu 100
```

Tipul constantelor C rezultă din forma lor de scriere, în concordanță cu tipurile de date fundamentale.
În C++ se preferă altă definire pentru constante simbolice, utilizând cuvântul cheie *const*. Pentru a face diferențierea dintre variabile şi constante este de preferat ca definirea constantelor să se facă pentru scrierea acestora cu majuscule.
Exemple:

```
const double PI = 3.1415;
const int LIM = 10000;
```

Variabile

Variabila este o entitate folosită pentru memorarea unei valori de un anumit tip, tip asociat variabilei. O variabilă se caracterizează prin nume, tip, valoare şi adresă:

- Orice variabilă are un **nume** (*identificator*), exemplu: *raza*, *area*, *varsta*, *nr*. Numele identifică în mod unic fiecare variabilă permițând utilizarea acesteia, cu alte cuvinte, numele unei variabile este unic (nu pot exista mai multe variabile cu acelaşi nume în acelaşi domeniu de definiție)
- Orice variabilă are asociat un **tip** de date. Tipul poate fi orice tip fundamental sau derivate, precum şi tipuri definite de utilizator.
- O variabilă poate stoca o **valoare** de un anumit tip. De menționat că în majoritatea limbajelor de programare, o variabilă asociată cu un anumit tip poate stoca doar valori aparținând acelui tip. De exemplu, o variabilă de tipul int poate stoca valoarea 123, dar nu şirul "Hello".
- Oricărei variabile i se alocă (rezervă) un spațiu de memorie corespunzător tipului variabilei. Acest spațiu este identificat printr-o **adresă de memorie.**

Pentru a folosi o variabilă trebuie ca mai întâi să îi declarăm numele şi tipul, folosind una din următoarele sintaxe:

***tip nume_variabila*;**
// Declară o variabilă de un anumit tip

***tip nume_variabila1*, *nume_variabila2*,...;**
// Declarație multiplă pentru mai multe variabile de acelaşi tip

***tip var-name* = *valoare_inițiala*;**
// Declară o variabilă de un anumit tip şi îi atribuie o valoare inițială

***tip nume_variabila1* = *valoare_inițiala1*, *nume_variabila2* = *valoare_inițiala2*, ... ;**
// Declară mai multe variabile unele putând fi inițializate, nu e obligatoriu să fie toate!

Sintetizând, definirea variabilelor se face astfel:

Tip lista_declaratori;

- ***lista_declaratori*** cuprinde unul sau mai multi declaratori, despărțiți prin virgulă
- ***declarator*** poate fi:
  - ***nume_variabila*** sau
  - ***nume_variabila = expresie_de_initializare***

În *expresie_de_initializare* pot apare doar constante sau variabile inițializate!

Exemple:

```
int sum;                // Declară a variabilă sum de tipul int
int nr1, nr2;           // Declară două variabile nr1 şi nr2 de tip int
double media;           // Declară a variabilă media de tipul double
int height = 20;        /* Declară a variabilă de tipul int şi îi atribuie o
valoare inițială */
char c1;                // Declară o variabilă c1 de tipul char
char car1 = 'a', car2 = car1 + 1;      // car2 se initializeaza cu 'b'
float real = 1.74, coef;  /*Declară două variabile de tip float prima din
ele este şi inițializată*/
```

În definirea ***tip lista_declaratori;*** tip poate fi precedat sau urmat de cuvântul cheie ***const***, caz în care variabilele astfel definite sunt de fapt **constante** ce trebuie să fie inițializate şi care nu-şi mai pot modifica apoi valoarea:

```
const int coef1 = -2, coef2 = 14;
coef1 = 5; /* modificarea valorii variabilei declarate const e gresita,
apare eroare la compilare!! */
```

Odată ce o variabilă a fost definită îi putem atribui sau reatribui o valoare folosind *operatorul de atribuire* "=".

Exemple:

```
int number;
nr = 99;        //atribuie valoarea întreagă 99 variabilei nr
nr = 88;        //îi reatribuie valoarea 88
nr = nr + 1;    //evaluează nr+1 şi atribuie rezultatul lui nr
int sum = 0;
int nr;         //ERROR: O variabilă cu numele nr e deja definită
sum = 55.66;    // WARNING: Variabila sum este de tip int
sum = "Hello";   /* ERROR: Variabila sum este de tip int. Nu i se poate
atribui o valoare şir */
```

Observații:

- O variabilă poate fi declarată o singură dată.
- În fişiere cu extensia *cpp* putem declara o variabilă oriunde în program, atâta timp cât este declarată înainte de utilizare.
- Tipul unei variabile nu poate fi schimbat pe parcursul programului.
- Numele unei variabile este un substantiv, sau o combinație de mai multe cuvinte. Prin convenție, primul cuvânt este scris cu literă mică, în timp ce celelalte cuvinte pot fi scrise cu prima literă mare, fără spații între cuvinte. Exemple: *dimFont*, *nrCamera*, *xMax*, *yMin*, *xStangaSus* sau *acestaEsteUnNumeFoarteLungDeVariabila*.

Comentariile

Comentariile sunt utilizate pentru documentarea şi explicarea logicii şi codului programului. Comentariile nu sunt instrucțiuni de programare şi sunt ignorate de compilator, dar sunt foarte importante pentru furnizarea documentației si explicațiilor necesare pentru înțelegerea programului nostru de către alte persoane sau chiar şi de noi înşine peste o săptămână.

Există două tipuri de comentarii:

Comentarii Multi-linie:

încep cu /* şi se termină cu */, şi se pot întinde pe mai multe linii

***Comentarii în linie*:**

încep cu // şi țin până la sfârşitul liniei curente.

Exemple:

```
/* Acesta este
            un comentariu în C */

// acesta este un alt comentariu C (C++)
```

În timpul dezvoltării unui program, în loc să ştergem o parte din instrucțiuni pentru totdeauna, putem să le comentăm astfel încât, să le putem recupera mai târziu dacă vom avea nevoie.

Operatori, operanzi, expresii

Limbajul C se caracterizează printr-un set foarte larg de operatori.

Operatorii sunt simboluri utilizate pentru precizarea operațiilor care trebuie executate asupra operanzilor.

Operanzii pot fi*:* constante, variabile, nume de funcții, expresii.

Expresiile sunt entități construite cu ajutorul operanzilor şi operatorilor, respectând *sintaxa (regulile de scriere)* şi *semantica (sensul, înțelesul)* limbajului. Cea mai simplă expresie este cea formată dintr-un singur operand.

Cu alte cuvinte, putem spune că o expresie este o combinație de *operatori* ('+', '-','*', '/', etc.) şi *operanzi* (variabile sau valori constante), care poate fi evaluată ca având o valoare fixă, de un anumit tip.

Expresiile sunt de mai multe tipuri, ca şi variabilele: 3*x+7, x<y etc.

La evaluarea expresiilor se ține cont de precedență şi asociativitatea operatorilor!

Exemple

```
1 + 2 * 3            // rezultă int 7

int sum, number;
sum + number;         // evaluată la o valoare de tip int

double princ, rata;
princ * (1 + rata);  // evaluată la o valoare de tip double
```

Operatorii

Clasificarea operatorilor se poate face:

- **după numărul operanzilor prelucrați:**
  - **unari**
  - **binari**
  - **ternari - cel condițional;**
- **după ordinea de succedare a operatorilor şi operanzilor:**
  - **prefixati**
  - **infixati**
  - **postfixati**
- **după tipul operanzilor şi al prelucrării:**
  - **aritmetici**
  - **relaționali**
  - **logici**
  - **la nivel de bit.**

Operatorii se împart în *clase de precedență*, fiecare clasă având o *regulă de asociativitate*, care indică ordinea aplicării operatorilor consecutivi de aceeaşi precedență (prioritate). Regula de asociativitate de la stânga la dreapta înseamnă că de exemplu următoarea expresie 1+2+3-4 este evaluată astfel ((1+2)+3)-4.

Tabelul operatorilor C indică atât regula de asociativitate, implicit de la stânga la dreapta (s-a figurat doar unde este dreapta-stânga), cât şi clasele de precedență, de la cea mai mare la cea mai mică precedență:

Operator	Semnificație	Utilizare	Asociativitate
( ) **[ ]** **.** **->** **--** **++**	paranteze indexare selecție selecție indirectă postdecrementare postincrementare	(e) t[i] s.c p->c a-- a++	
- **+** **--** **++** **!** **~** ***** **&** **sizeof** **( )**	schimbare semn plus unar (fără efect) predecrementare preincrementare negație logică complementare (negare bit cu bit) adresare indirectă preluare adresă determinare dimensiune (în octeți) conversie de tip (cast)	-v +v --a ++a !i ~i *p &a sizeof(x) (d) e	dreapta - stânga ←
***** **/** **%**	înmulțire împărțire rest împărțire (modulo)	v1 * v2 v1 / v2 v1 % v2	
+ **-**	adunare scădere	v1 + v2 v1 - v2	
<< **>>**	deplasare stânga deplasare dreapta	i1 << i2 i1 >> i2	
< **<=** **>** **>=**	mai mic mai mic sau egal mai mare mai mare sau egal	v1 < v2 v1 <= v2 v1 > v2 v1 >= v2	
== **!=**	egal diferit	v1 == v2 v1 != v2	
&	şi pe biți	i1 & i2	

^	sau exclusiv pe biți	i1 ^ i2	
|	sau pe biți	i1 | i2	
&&	şi logic (conjuncție)	i1 && i2	
||	sau logic (disjuncție)	i1 || i2	
? :	operator condițional (ternar)	expr ? v1 : v2	dreapta - stânga ←
= ***= /= %=** **+= -=** **&= ^= |=** **<<= >>=**	atribuire variante ale operatorului de atribuire	a = v a *= v	dreapta - stânga ←
,	secvențiere	e1, e2	

Legendă:

a - variabila întreagă sau reală
c - câmp
d - nume tip
e - expresie
f - nume de funcție
x - nume tip sau expresie
i - întreg
v - valoare întreagă sau reală
p - pointer
s - structură sau uniune
t - tablou

Operatorii aritmetici

C suportă următorii operatori aritmetici pentru numere de tip: *short, int, long, long long, char* (tratat ca 8-bit signed integer), *unsigned short, unsigned int, unsigned long, unsigned long long, unsigned char, float, double* şi *long double*.
expr1 şi *expr2* sunt două expresii de tipurile enumerate mai sus.

Operator	Semnificație	Utilizare	Exemple
-	schimbare semn	*-expr1*	-6 -3.5
+	plus unar (fără efect)	*+expr1*	+2 +2.5
*****	înmulțire	*expr1 * expr2*	2 * 3 → 6; 3.3 * 1.0 → 3.3
/	împărțire	*expr1 / expr2*	1 / 2 → 0; 1.0 / 2.0 → 0.5
%	rest împărțire (modulo)	*expr1 % expr2*	5 % 2 → 1; -5 % 2 → -1
+	adunare	*expr1 + expr2*	1 + 2 → 3; 1.1 + 2.2 → 3.3

-	scădere	*expr1* - *expr2*	1 - 2 → -1; 1.1 - 2.2 → -1.1

Este important de reținut că int / int produce un int, cu rezultat trunchiat, şi anume partea întreagă a împărțirii: 1/2 → 0 (în loc de 0.5), iar operatorul modulo (%) este aplicabil doar pentru numere întregi.

În programare, următoarea expresie aritmetică:

$$\frac{1+2a}{3}+\frac{4(b+c)(5-d-e)}{f}-6\left(\frac{7}{g}+h\right)$$

Trebuie scrisă astfel: (1+2*a)/3 + (4*(b+c)*(5-d-e))/f - 6*(7/g+h). Simbolul pentru înmulțire '*' nu se poate omite (cum este în matematică).

Ca şi matematică, înmulțirea şi împărțirea au precedență mai mare decât adunarea şi scăderea. Parantezele sunt însă cele care au cea mai mare precedență.

Depăşirile la calculele cu reali sunt semnalate, dar nu şi cele de la calculele cu întregi (valoarea rezultată este trunchiată). Se semnalează, de asemenea, eroarea la împărțirea cu 0.

Operatori relaționali şi logici

Deseori, e nevoie să comparăm două valori înainte să decidem ce acțiune să realizăm. De exemplu, dacă nota este mai mare decât 50 afişează "Admis". Orice operație de comparație implică doi operanzi (x <= 100)

În C există şase operatori de comparație (mai sunt numiți şi operatori relaționali):

Operator	Semnificație	Utilizare	Exemple (x=5, y=8)
==	egal cu	*expr1* == *expr2*	(x == y) → false
!=	diferit	*expr1* != *expr2*	(x != y) → true
>	mai mare	*expr1* > *expr2*	(x > y) → false
>=	mai mare egal	*expr1* >= *expr2*	(x >= 5) → true
<	mai mic	*expr1* < *expr2*	(y < 8) → false
<=	mai mic egal	*expr1* >= *expr2*	(y <= 8) → true

Aceste operații de comparație returnează valoarea 0 pentru fals şi o valoare diferită de zero pentru adevărat.

Convenție C:

Valoarea 0 este interpretată ca *fals* şi orice valoare diferită de zero ca *adevărat*.

În C există patru operatori logici:

Operator	Semnificație	Utilizare	Exemple (expr1=0, expr2=1)
&&	şi logic	*expr1* && *expr2*	**0**
||	sau logic	*expr1* || *expr2*	Diferit de 0
!	negație logică	!*expr1*	Diferit de 0
^	sau exclusiv logic	*expr1* ^ *expr2*	1

Tabelele de adevăr sunt următoarele:

AND (&&)	true	false
true	true	false
false	false	false

NOT (!)	true	false
	false	true

OR (||)	true	false
true	true	true
false	true	false

XOR (^)	true	false
true	false	true
false	true	false

Este incorect să scriem 1 < x < 100 , în loc de aceasta spargem în două operații de comparație x>1, x < 100, pe care le unim cu un operator logic ŞI: (x > 1) && (x < 100).

Exemple:

```
// Returnează true dacă x este între 0 şi 100 (inclusiv)
(x >= 0) && (x <= 100)         // greşit 0 <= x <= 100

// Returnează true dacă x nu este între 0 şi 100 (inclusiv)
(x < 0) || (x > 100)   //sau
!((x >= 0) && (x <= 100))

/* Returnează true dacă year este bisect. Un an este bisect dacă este
divizibil cu 4 dar nu cu 100, sau este divisibil cu 400.*/
((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0)
```

Operatorii de atribuire

Pe lângă operatorul de atribuire '=' descris mai sus, limbajul C mai pune la dispoziție aşa numiții operatori de atribuire compuşi:

Operator	Semnificație	Utilizare	Exemple
=	atribuire	var = expr	x=5

+=	*var = var + expr*	var + = expr	x+=5 (echivalent cu x=x+5)
-=	*var = var – expr*	var - = expr	x-=5 (echivalent cu x=x-5)
***=**	*var = var * expr*	var * = expr	x*=5 (echivalent cu x=x*5)
/=	*var = var / expr*	var / = expr	x/=5 (echivalent cu x=x/5)
%=	*var = var % expr*	var %= expr	x%=5 (echivalent cu x=x%5)

Limbajul C mai introduce şi cei doi operatori aritmetici pentru incrementare '++' şi decrementare '--' cu 1:

Operator	Semnificație	Exemple	Rezultat

--var	predecrementare	y = --x;	echivalent cu
			x=x-1;
			y=x;

var--	post decrementare	y=x--;	echivalent cu
			y=x;
			x=x-1;

++var	preincrementare	y=++x;	echivalent cu x=x+1; y=x;

var++	post incrementare	y=x++;	echivalent cu
			y=x;
			x=x+1;

Exemple:

```
int i, j, k;
// variabilele sunt initializate cu aceeasi valoare, 0
i = j = k = 0;

float lungime, latime, inaltime, baza, volum;
// calculeaza baza si volumul unui paralelipiped
volum = inaltime * ( baza = lungime * latime );
```

Operatorul sizeof

Rezultatul aplicării acestui operator unar este un întreg reprezentând *numărul de octeți* necesari pentru stocarea unei valori de tipul operandului sau pentru stocarea rezultatului

expresiei dacă operandul este o expresie. Operatorul are efect la compilare, pentru că atunci se stabileşte tipul operanzilor.

Sintaxa:

sizeof (tip)
sizeof (expresie)

Exemple:

```
sizeof('a')        // 1
sizeof(int)        // 2
sizeof(2.5 + 3)    // 4??
```

Conversii de tip. Operatorul de conversie explicita (cast)
Conversia de tip ia un operand de un anumit tip şi returnează o valoare echivalentă de un alt tip.

Există două tipuri de conversii de tip:

1. **Implicit, realizat automat de compilator**
2. **Explicit, utilizând operatorul unar de conversie de tip în forma (tip_nou) *operand***

Conversii de tip implicite
În limbajul C, dacă atribuim o valoare double value unei variabile întregi, compilatorul realizează o conversie de tip implicit, returnând o valoare întreagă. Partea fracțională se va pierde. Unele compilatoare generează o avertizare (warning) sau o eroare "possible loss in precision"; altele nu.
La evaluarea expresiilor pot apare *conversii implicite*:

- dacă o expresie are doar operanzi întregi, ei se convertesc la int
- dacă o expresie are doar operanzi reali sau intregi, ei se convertesc la double.
- dacă o expresie are operanzi de tipuri diferite, compilatorul promovează valoarea tipului mai *mic* la tipul mai *mare*. Operația se realizează apoi în domeniul tipului mai mare.De exemplu, int / double → double / double → double. Deci, 1/2 → 0, 1.0/2.0 → 0.5, 1.0/2 → 0.5, 1/2.0 → 0.5.
- în expresia variabila=expresie se evaluează prima dată expresia, fără a ține cont de tipul variabilei; dacă tipul rezultatului obținut este diferit de cel al variabilei, se realizează conversia implicită la tipul variabilei astfel:

De exemplu,

Tip	Exemplu	Operație
int	2 + 3	int 2 + int 3 → int 5
double	2.2 + 3.3	double 2.2 + double 3.3 → double 5.5
mixt	2 + 3.3	int 2 + double 3.3 → double 2.0 + double 3.3 → double 5.3

int	1 / 2	int 1 / int 2 → int 0
double	1.0 / 2.0	double 1.0 / double 2.0 → double 0.5
mixt	1 / 2.0	int 1 / double 2.0 → double 1.0 + double 2.0 → double 0.5

Exemple conversii implicite:

```
int i; char c = 'c'; long l; float f;
i = 2.9;           // 2.9 e convertit implicit la int, deci i va fi 2
f = 'A';           // f va fi 65.0 ( codul ASCII )
i = 30000 + c;     // expresia se calculeaza in domeniul int, i va fi 30099
i = 30000 + 10000;// calcul in dom. int, depasire nesemnalata, i e 25536
l = 30000 + 10000;       // -25536 va fi convertit la long
l=30000u+10000;          // rezultat corect
l=30000l+10000;          // rezultat corect
```

Conversii de tip explicite

Operatorul de conversie explicită (*cast*) se utilizează atunci când se doreşte ca valoarea unui operand (expresie) să fie de un alt tip decât cel implicit. Operatorul este unar, are prioritate ridicată si are sintaxa: (tip) expresie

Exemple conversii explicite

```
float r;
r = 5 / 2;         // impartirea se face in domeniul int, deci r va fi 2.0

r = (float) 5 / 2; /* r va fi 2.5, pentru ca primul operand este de tip
float calculul se face în domeniul real */

int x = (int) r;        //x va fi 2
```

C++ supportă şi conversii de tip de genul *new-type*(*operand*):

```
medie = double(sum) / 100;      // echivalent cu (double)sum / 100
```

Operatorul condițional

Operatorul condițional ? : este singurul *operator ternar*. Are prioritatea mai ridicată doar decât a operatorilor de atribuire şi a celui secvential, iar asociativitatea este de la dreapta spre stanga. El se foloseşte în situațiile în care există două variante de obținere a unui rezultat, dintre care se alege una singură, funcție de îndeplinirea sau neîndeplinirea unei condiții. Cei trei operanzi sunt expresii, prima reprezentand condiția testată.

```
expr0 ? expr1 : expr2
```

Dacă valoarea expr0 este adevarată (!=0), se evaluează expr1, altfel expr2, rezultatul expresiei evaluate fiind rezultatul final al expresiei condiționale.

Exemple:

```
/* Expresia de mai jos determină valoarea maximă dintre a si b, pe care o
```

```
memorează în max:*/
max = a > b ? a : b;

/* Funcție de ora memorată în variabila hour, funcția puts va tipări mesajul
corespunzător.
Evaluare dreapta -> stânga a expresiilor condiționale multiple */
puts( hour < 0 || hour > 24 ? "Ora invalida" : hour < 12 ? "Buna dimineata!"
: hour < 18 ? "Buna ziua!" : hour < 22 ? "Buna seara" : "Noapte buna!");
```

Operatorul secvențial

Operatorul secvențial , (virgulă) este cel cu prioritatea cea mai scăzută. Se foloseşte atunci când sintaxa limbajului impune prezența unei singure expresii, iar prelucrarea presupune evaluarea a două sau mai multe expresii; acestea se evaluează de la stânga la dreapta, iar rezultatul întregii expresii este cel al ultimei expresii (exprn):

```
expr1, expr2, ..., exprn
```

Exemplu:

```
/* Expresia de mai jos memorează în max valoarea maximă dintre a si b,
realizând şi ordonarea descrescătoare a acestora (le interschimbă dacă
a<b). A se observa că interschimbarea presupune utilizarea unei variabile
auxiliare). Operatorul secvențial e necesar pentru a avea o singură
expresie după : */

    int a, b, aux, max;
    max = a >= b ? a : (aux = b, b = a, a = aux);
```