Programarea Calculatoarelor

Curs 5

Iulian Năstac

Recapitulare din cursul precedent

Clasificarea limbajelor de programare

- **Limbaje de nivel înalt** (Ada, Pascal, Fortran, etc.)
  - limbaje de programare cu un nivel înalt de abstracție care nu depinde de detaliile unui anumit calculator

- **Limbaje de nivel mediu** (C, C++, FORTH, etc.)

- **Limbaje de nivel scăzut** (Macro-assembler; Limbaje de asamblare)
  - limbaje de programare ce depind puternic de detaliile unui anumit calculator

Recap.

Limbajul de programare C

- 1966 Martin Richards (University of Cambridge) a dezvoltat **BCPL** (Basic Combined Programming Language)

- 1969 **Ken Thomson** ajutat și de Dennis Ritchie – **B programming language**

- 1969-1973 **Dennis Ritchie** – **C programming language**

Recap.
Caracteristici ale limbajului C

1. Portabilitatea
2. Tipurile de date
3. Controlul erorilor
4. Lucrează la nivelul limbajului assembler
5. Număr restrâns de cuvinte cheie (keywords)
6. Intră în categoria limbajelor structurate
7. Considerat un limbaj al programatorului

Recap.
Structura unui program scris în C

- Declarații globale:
  - Includerea de fișiere header
  - declararea de constante și variabile globale
  - declararea de funcții locale

- Funcția **main()**

- Alte funcții

Recap

Preprocesare în C

- Un program sursă C poate fi prelucrat înainte de a fi supus compilării. Această prelucrare poartă numele de preprocesare.

- Preprocesorul asigură:
  - includeri de fişiere cu texte sursă;
  - definiţii şi apeluri de macrouri;
  - compilare condiţionată.

Constante în limbajul C

- Constantele reprezintă valori care nu se modifică pe parcursul evoluţiei unui program. Acestea reprezintă valori de anumite tipuri, care sunt determinate de caracterele ce intră în compunerea constantelor.

- Ex:
  #define PI 3.14159

- Declararea unei constante permite schimbarea rapidă și ușoară a unei valori care este folosită pe tot parcursul programului, pur și simplu prin schimbarea declarației inițiale.

Clasificarea constantelor

- ***Întregi*** - generate pe 2 octeţi
- ***Lungi explicite*** - generate pe 4 octeţi
- ***Flotante (reale)*** - de tip *float*
  - de tip *double*
- ***Caracter*** (Ex.: ‘x’ – constantă caracter)
- ***Şir de caractere*** (“mesaj” – constantă şir de caractere)
- ***Constante simbolice*** - introduse de directiva *#define* (ex: #define MAX 100)

Tipuri de date de bază

- În limbajul *C* există 5 tipuri de date de bază:
  - *char* – caracter
  - *int* – întreg
  - *float* – virgulă mobilă
  - *double* – virgulă mobilă cu dublă precizie
  - *void* – fără nici o valoare

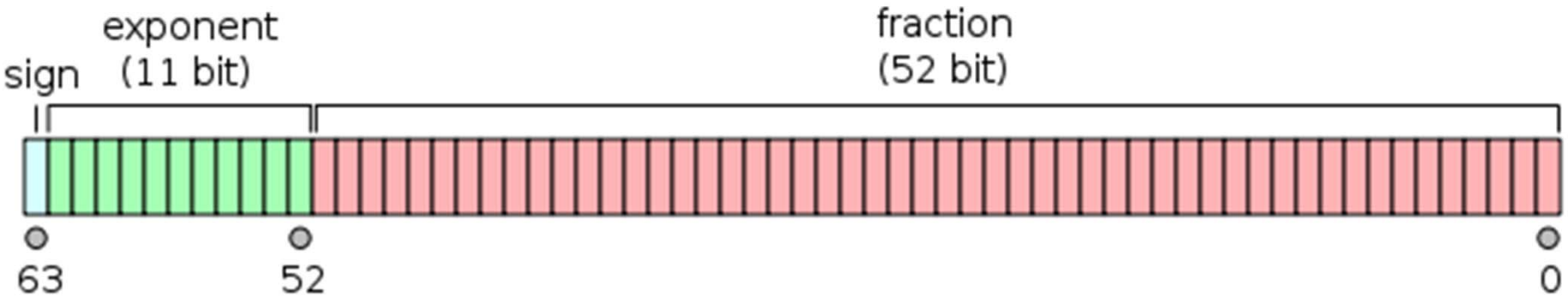
- *specificatorii de conversie*:
  - signed,
  - unsigned
  - short
  - long

Tip	Explicații
char	Cea mai mică unitate adresabila (8 biți), care poate conține setul de caractere de bază. Este un tip întreg. Poate fi cu sau fără semn.
signed char	La fel ca char, dar cu semn obligatoriu
unsigned char	La fel ca char, dar sigur fără semn.
short **short int** **signed short** **signed short int**	*Întreg scurt*. Lungime de cel puțin 16 biți
unsigned short **unsigned short int**	La fel ca short, dar fără semn
int **signed int**	Tipul întreg de bază. Lungime de cel puțin 16 biți (dar deseori dublu față de short)
unsigned **unsigned int**	La fel ca int, dar fără semn

Tip	Explicații
long **long int** **signed long** **signed long int**	*Întreg lung*. Lungime de cel puțin 32 biți
unsigned long **unsigned long int**	La fel ca ***long***, dar fără semn.
long long **long long int** **signed long long** **signed long long int**	*long long* este un întreg cu semn de mare dimensiune. Lungime de cel puțin 64 biți (specificat odată cu versiunea standard C99).
unsigned long long **unsigned long long int**	La fel ca long long, dar fără semn. (specificat odată cu versiunea standard C99).

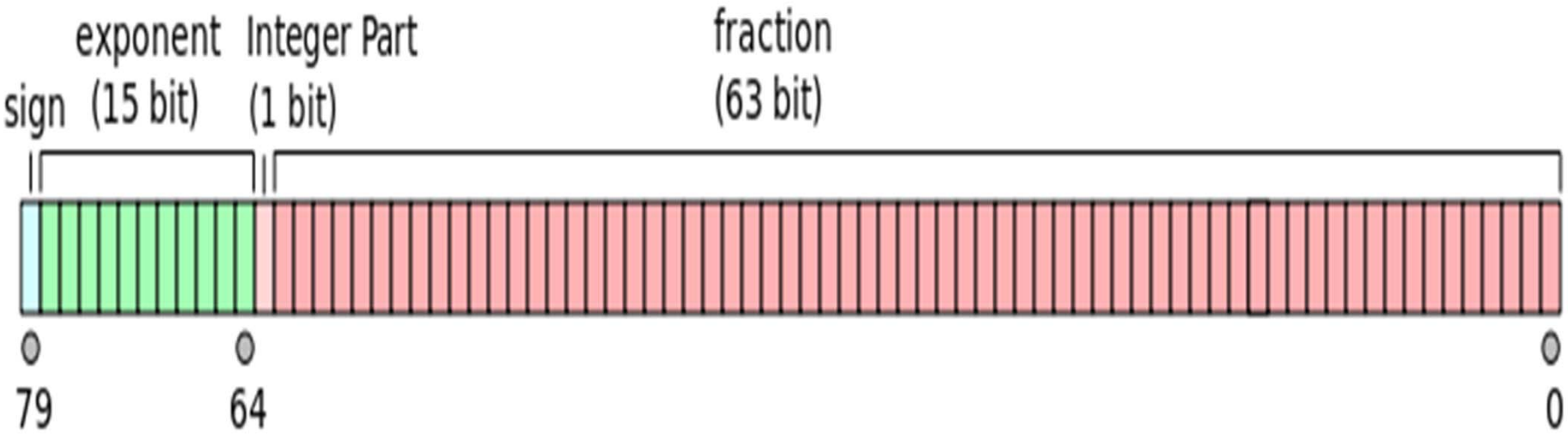
Tip	Explicații
float	Format în virgulă mobilă cu precizie simplă. Ocupă 4 octeți în memorie (32 biți).
double	Format în virgulă mobilă cu precizie dublă. Ocupă 8 octeți în memorie (64 biți).
long double	Format în virgulă mobilă cu precizie extinsă. Extended precision floating-point type. Ocupă 80-biți (IEEE 754 quadruple-precision floating-point format)

double



În general, tipul double are 15-16 cifre zecimale pentru precizie, în timp ce float are doar 7.

long double



Formatul virgulă mobilă de 80 de biți a fost disponibil pe scară largă după 1984.

Tip	Variantă	Dimensiune aproximativă în biţi	Domeniu de valori
char	*char*	8	-127 ÷ 127
	unsigned char	8	0 ÷ 255
	signed char	8	-127 ÷ 127
int	*int* (signed int, short int, signed short int)	16	-32767 ÷ 32767
	unsigned int (unsigned short int)	16	0 ÷ 65535
	long int (signed long int)	32	-2.147.483.647 ÷ 2.147.483.647
	unsigned long int	32	0 ÷ 4.294.967.295
float	*float*	32	$\sim +/-3.4\times10^{-38} \div +/-3.4\times10^{38}$ şase zecimale exacte
double	*double*	64	$\sim +/-1.7\times10^{-308} \div +/-1.7\times10^{308}$ şase zecimale exacte
	long double	80 ÷ 128	$\sim +/-3.4\times10^{-4932} \div +/-3.4\times10^{4932}$

Observații:

- Dimensiunea tipului întreg variază în funcție de implementarea compilatorului utilizat.
- Standardul prevede doar relații de mărime între tipurile de date și dimensiunile minime pentru fiecare tip de date.
- **long long** este mai cuprinzător decât **long**, care este mai cuprinzător decât **int**, care este mai cuprinzător decât **short**.

Observații:

- **char** este întotdeauna cel mai mic tip de date standard.
- Dimensiunea minimă pentru **char** este de 8 biți, dimensiunea minimă pentru **short** și **int** este de 16 biți, pentru **long** este de 32 biți, iar **long long** trebuie să conțină cel puțin 64 biți.
- Există o multitudine de conversii posibile în C.

Conversii în C

- ***Implicite*** - dacă într-o expresie sunt mai multe tipuri de date, se face conversia la tipul cel mai cuprinzător.

- ***prin asignare*** - valoarea membrului drept este convertită la valoarea membrului stâng care este şi tipul rezultatului.

- ***conversii logice*** (la folosirea operatorilor logici).

Conversii explicite
(conversia de tip **cast)**

```
float a = 1.3;
double b = 2.5;
long c = 3.4;
int x;
…
x = (int)a + (int)b + (int)c;
```

Notă:

- Rezultatul este x == 9
- dacă s-ar fi folosit conversia implicită (de ex. “x = a + b + c"), rezultatul ar fi fost egal cu 7

Variabile

- **Variabilele** sunt pur și simplu denumirile unor valori ce pot suferi modificări pe parcursul unui program.

- Clase de variabile:
  - Variabile locale
  - Variabile globale

O clasificare mai specializată defineşte următoarele tipuri de variabile:

- **Variabile automatice**
  - O variabilă automată este o variabilă definită în interiorul unui bloc funcțional
  - sunt alocate și dezalocate în mod automat atunci când fluxul de program intră și iese din blocul unde a fost definită variabila
- **Variabilele externe**
  - variabilă este definită în afara oricărui bloc
- **Variabilele locale statice**
  - variabile care au fost alocate static - a căror existență/valabilitate se extinde pe întreaga durata a programului
- **Variabilele registru**
  - alocare în regiștri este procesul de atribuire a unui număr de variabile de program direct în regiștrii procesorului

Variabile automatice

- Termenul ***variabilă locală*** este de obicei sinonim cu ***variabilă automată***, deoarece acestea sunt același lucru în mai multe limbaje de programare.

- Cele mai multe **variabile locale** sunt **variabile locale automate**, dar există și o categorie specială de **variabile locale statice** în C.

- Se declară implicit în context sau cu identificatorul *"**auto**"*.

- Aceste variabile sunt locale fiecărui bloc şi se distrug la părăsirea blocului.

Variabilele externe

- Ca o alternativă la variabilele automatice, este posibil să se definească variabilele care sunt **externe**, adică definite în afara oricărei funcții (variabile care pot fi accesate însă de orice funcție a programului).

- O variabilă externă poate fi, de asemenea, redeclarată în interiorul unei funcții. În acest caz, cuvântul cheie **extern** trebuie să fie utilizat, deoarece în caz contrar compilatorul o va considera variabilă locală. Această redeclarație va fi vizibilă doar în interiorul funcției.

- O variabilă de tip **extern** pot fi accesată de către toate funcțiile din toate modulele unui program. Este o variabilă globală.

Variabile locale statice

- Se declară cu ajutorul cuvântului cheie ***static*** şi nu sunt recunoscute decât în cadrul fişierului în care au fost definite.

- Nu se alocă pe stivă.

- Sunt de două tipuri:
  - *interne* (în interiorul unei funcţii şi nu se distrug la părăsirea funcţiei).
  - *externe* (în tot programul).

Variabile registru

- Se declară cu *ajutorul cuvântului cheie* ***register***
  Sunt asemănătoare variabilelor auto.

- Indiferent de încadrarea la una din clasele anterior enunţate, variabilele sunt definite împreună cu *tipul* lor: ***register tip nume_variabilă;***

- Registrul trebuie să fie suficient de mare pentru a găzdui o variabilă de aceast tip (se pretează pentru **char** și **int**).

Iniţializarea variabilelor

- O variabilă poate fi iniţializată astfel:

clasă tip nume = expresie;

unde *clasă* poate lipsi (la variabile globale sau auto), sau poate fi *extern*, *auto*, *register*, *static*.

Observaţii:

- Se fac automat conversii dacă tipul expresiei de iniţializare nu coincide cu tipul variabilei pe care o iniţializează.
- Expresiile utilizate pentru iniţializare trebuie să fie constante.
- Variabilele globale şi statice neiniţializate au implicit valoarea 0. Restul au valori nedefinite (imprevizibile), până în momentul când li se atribuie o valoare printr-o instrucţiune de atribuire. De aceea este indicat ca variabilele să se iniţializeze imediat la declarare.

Exemplu:

```
# define MAX 100
...
int funct(int m)  /*Parametrii unei funcţii nu se iniţializează! */
{  int z=5;
   int a=z+m;
   float c=a/MAX;
   ...
}
```

Specificatori de format

- Un **specificator de format** constă dintr-o pereche formată din **%** (procent) și un caracter ce indică tipul variabilei utilizate.
- Indică unei funcţii de intrare/ieşire tipul unei variabile folosite.
- Folosirea incorectă a specificatorilor de format poate duce la erori în compilarea programului (sau conversii nedorite în execuţia programului care favorizează alterarea datelor).

specificator de format	**Tipul asociat de variabilă**
%c	caracter
%s	Şir de caractere
%d	întreg
%u	întreg fără semn
%p	Pointer (adresă)
%f	Real (format simplă precizie)
%lf	Real (format dublă precizie)
%Lf	Real (format extins)

Notă:

%5d – întreg pe 5 spaţii cu aliniere de la dreapta

%-5d – întreg pe 5 spaţii cu aliniere de la stânga

Secvenţe de evitare
(Escape sau *backslash \ character)*

- Denumite uneori *constante backslash caracter*, secvenţele de evitare pot reprezenta salturi făcute de cursorul programului sau o afişare de anumite caractere.
- De obicei, încadrarea constantelor de tip caracter între ghilimele simple lucrează pentru majoritatea caracterelor afişabile.
- Există unele caractere (linie nouă, tab, etc.) imposibil de introdus de la tastatură.
- Din acest motiv *C* include constante speciale numite *backslash caracter*.

Notă:

\ "acționează" înăuntrul unui șir se caractere

Escape (backslash \) character	**Efectul într-un șir de caractere**
\n	linie nouă
\t	caracterul tab (tabulare orizontală)
\b	backspace
\v	tabulare verticală
****	Caracterul backslash
\/	caracterul slash

Codurile ASCII

- **American Standard Code for Information Interchange** (ASCII) stabilește modul de codare (în format **7-bit binary integers**) pentru 128 caractere utilizate în alfabetul Englez plus semne de punctuație.

Notă:

- Codurile ASCII sunt de obicei generate în zona tampon a tastaturii.
- În urma apăsării unei taste sau unei combinaţii de taste se generează nişte cifre de cod (hexazecimal) care sunt preluate de S.O. pentru a fi transformate în simboluri (litere, cifre, semne de punctuaţie) sau diferite comenzi (TAB, Enter, etc.).

ASCII codează 128 caractere:

- 33 are charactere neprintabile (multe fiind acum învechite)

- 95 caractere printabile, incluzând și spațiul gol (care este considerat invizibil grafic)

USASCII code chart

Bits b_7 b_6 b_5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4	b_3	b_2	b_1	Row ↓ Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Exemple	Cod hexa
ESC	1B
1 2 : 9	31 32 : 39
0	30
A B :	41 42 :
a b :	61 62 :

Ca observaţie, în unele cazuri se va genera un cod dublu, primul caracter fiind nul (este vorba de taste speciale pentru care se face o dublă citire de caracter).

Rutine standard de intrare şi ieşire

- Rutinele standard de intrare şi ieşire sunt funcţii care suplinesc lipsa instrucţiunilor de intrare şi ieşire.

- Acestea se regăsesc în diferite biblioteci de headere (în special *stdio.h*, *conio.h și stdlib.h*).

A. **Principalele rutine de ieșire**

1. Rutina *printf*

- int printf (const char * format, ...);

- Are ca rezultat afişarea unui mesaj pe ecran.
- Ca observaţie, *şirul format* trebuie să fie încadrat între ghilimele. Acest șir poate include specificatori de format (%), valorile variabilelor de după ghilimele fiind introduse în locul acestor specificatori.

- Ex.:

  printf(“\n Rezultatul este %d \n”, x);

2. **Rutina *puts***

- int puts (const char * sir_caractere);

- Funcţia *puts* scrie pe ecran un şir urmat de salt la linie nouă (\n).

- Ex.:

printf(“Mesaj\n”); ⇔ *puts(“Mesaj”);*

3. **Rutina *putchar***

- int putchar (int caracter);

- Macroul *putchar* afişează un caracter al codului ASCII. Se returnează codul caracterului afişat sau -1 la eroare.

- Ex.:

```
#include <stdio.h>
int main()
{ char c;
for (c = 'A' ; c <= 'Z' ; c++) putchar(c);
return 0;
}
```

Observaţii:

Apelul *putchar(10);* ⇔ *putchar('\n');* are ca efect trecerea cursorului în coloana 1 de pe linia următoare.

4. **Rutina *putch***

- int putch (int caracter);

- Spre deosebire de *putchar* care se ocupă numai de caractere ale codului ASCII (nu şi de caractere corespunzătoare tastelor speciale), funcţia *putch* afişează în plus şi caracterele speciale.

- Ex.: *putch(parametru);*

Observaţie:

- Valoarea parametrului se interpretează ca fiind codul ASCII al caracterului care se afişează.

- Dacă valoare expresiei se află în:
  - intervalul [32, 126] – se afişează un caracter imprimabil al codului ASCII;
  - afara intervalului [32, 126] – se afişează diferite imagini care pot fi folosite în diverse scopuri, cum ar fi trasarea de chenare.

B. **Principalele rutine de intrare**

1. **Rutina *scanf***

- **int scanf (const char * format, ...);**

- Funcţia *scanf* citeşte un şir de mesaje de la tastatură pe care le pune la anumite adrese indicate de parametrii funcţiei.

- Citirea se face cu ecou.

Exemple:

scanf("%d %d", &x, &y);

Spaţiul liber indică faptul că între cele două valori pot exista oricâte spaţii goale: blank, tab, linie nouă, etc.).

scanf("%d, %d", &x, &y);

Virgula arată că cele două valori vor fi despărţite la citire prin virgulă.

Observaţii:

Citirea caracterelor prin intermediul zonelor tampon are avantajul că permite corectarea erorilor de stare. Astfel operatorul de la terminal poate să modifice o secvenţă de caractere tastate, înainte de a acţiona tasta Enter.

2. **Rutina *gets***

- **char * gets (char * sir_caractere);**

- Funcţia *gets* citeşte un întreg şir de caractere tastat (inclusiv blank-uri) până când se apasă tasta Enter. Citirea se face cu ecou.
- Se pot citi numai caractere ale codului ASCII. Funcţia *gets* returnează adresa de început a zonei de memorie în care s-au păstrat caracterele.

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


int main()
{
char a[20];
gets(a);
puts(a);

getch();
}
```

3. **Rutina *getchar***

- **int getchar (void);**

- Funcţia *getchar* permite citirea fără ecou a unui caracter de la terminalul standard. Citirea se face fără ecou.
- Se pot citi numai caractere ale codului ASCII, nu şi caractere corespunzătoare tastelor speciale.

- Ex.:

```
variabila = getchar();
```

```
#include <stdio.h>

int main()

{ putchar(getchar());
   putchar('\n');
   getch();
}
```

4. Funcția **getch**

int getch (void);

- getch() acceptă doar caractere singulare de la tastatură.

- *getch()* - lucrează fără ecou.

5. Funcția **getche**

int getche (void);

- getche() acceptă caractere singulare de la tastatură.

- *getch()* - lucrează cu ecou.

Sinteza rutinelor de bază intrare – ieşire

IEŞIRI	INTRĂRI
printf	scanf
puts	gets
putchar	getchar
putch	getch getche

Expresii

- O expresie într-un limbaj de programare este o combinație de valori explicite, constante, variabile, operatori și funcții care sunt interpretate în conformitate cu normele specifice de prioritate și de asociere pentru un anumit limbaj de programare. Programul calculează expresia și produce o nouă valoare.
- Acest proces asupra unei expresii se numește evaluare.
- Valoarea unei expresii poate fi de diferite tipuri, cum ar fi numerică, șir de caractere, sau logică.

- Expresiile pot fi
  - primare
  - listă de expresii

Ordinea evaluării

- În programare, ordinea de evaluare a operațiilor (ce stabilește prioritatea lor) este o regulă folosită pentru a clarifica care operație ar trebui să fie efectuate mai întâi într-o expresie dată.

- Operatorii din C au o ordine strictă ce stabilește prioritățile.

Precedenţa maximă	**( ) [ ] → .**
	! ~ ++ -- * &
	*** / %**
	+ -
	<< >>
	< <= > >=
	== !=
	^
	|
	&&
	||
	? :
Precedenţa minimă	**= += -= *= /=**

1	**() [] -> . ::**	grupare, matrici, acces
2	**! ~ - + * & sizeof *type cast* ++x - -x**	Operatori unari, sizeof și casts
3	*** / %**	Multiplicare, împărțire, modulo
4	**+ -**	Adunare și scădere
5	**<< >>**	Deplasare stânga și dreapta
6	**< <= > >=**	Comparație inegalitate
7	**== !=**	Comparație egalitate
8	**&**	ȘI logic (pe biți)
9	**^**	SAU exclusiv (pe biți)
10	**|**	SAU (pe biți)
11	**&&**	ȘI logic
12	**||**	SAU logic
13	**? : = += -= *= /= %= &= |= ^= <<= >>=**	Expresia condițională și operatorii de atribuire
14	**,**	Operatorul virgulă

Conversia automată în expresii

- Compilatorul face conversia tuturor elementelor asupra cărora se operează în tipul celui mai mare, acţiune numită *promovarea tipului*.

- ***regula conversiilor implicite în C***
  - aceasta acţionează când un operator binar se aplică la doi operanzi.

Paşii regulii *conversiilor implicite:*

- Mai întâi se convertesc operanzii de tip char şi enum la tipul int;

- Dacă operatorul curent se aplică la operanzi de acelaşi tip atunci rezultatul va fi de acelaşi tip. Dacă rezultatul reprezintă o valoare în afara limitelor tipului respectiv atunci rezultatul este eronat (au loc depăşiri).

- Dacă operatorul binar se aplică la operanzi diferiţi atunci se face o conversie înainte de execuţia operatorului conform paşilor de mai jos:
  - Dacă un operand este long double rezultă că şi celălalt se converteşte spre long double şi rezultatul este de tip long double.
  - Altfel, dacă unul dintre operanzi este de tip double şi celălalt se converteşte spre double şi rezultatul este de tip double .
  - Altfel, dacă unul dintre operanzi este de tip float rezultă că celălalt este tot float, iar rezultatul este float.
  - Altfel, dacă unul dinre operanzi este unsigned long rezultă că şi celălalt se converteşte la unsigned long, iar rezultatul este de tip unsigned long.
  - Altfel, dacă unul dintre operanzi este de tip long rezultă că şi celălalt se converteşte la tipul long, iar rezultatul operaţiei este tot de tip long.
  - Altfel, dacă unul dintre operanzi este de tip unsigned şi celălalt int rezultă că se converteşte ultimul la unsigned, iar rezultatul este unsigned.

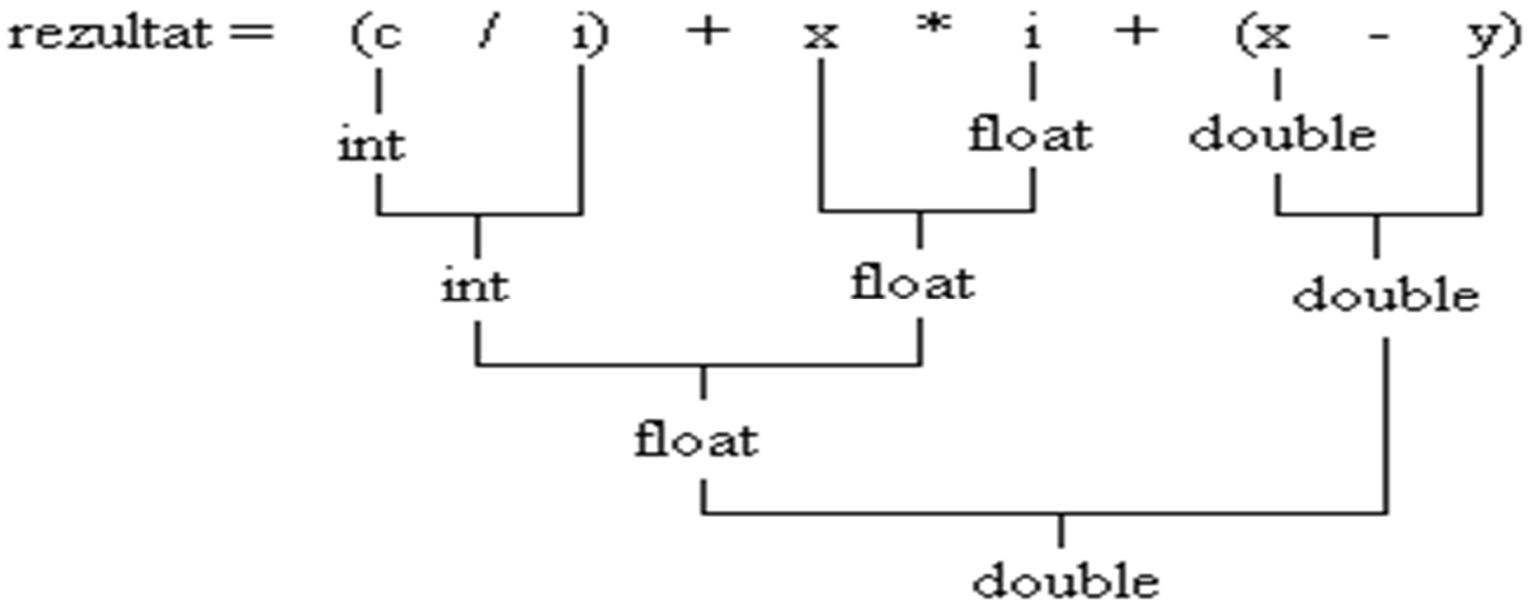
Observaţie:

- Pentru operatorii de atribuire (cum ar fi „=”) regula de conversie este: membrul drept se converteşte la tipul membrului stâng.

Exemplu:

- …

  float x;
  double y;
  char c;
  int i;
  . . .



Operatori în C

- Limbajele de programare utilizează un set de operatori cu o sintaxă proprie.

- Limbajul de programare C conține un număr fix de operatori predefiniți.

1	**() [] -> . ::**	grupare, matrici, acces
2	**! ~ - + * & sizeof *type cast* ++x - -x**	Operatori unari, sizeof și casts
3	*** / %**	Multiplicare, împărțire, modulo
4	**+ -**	Adunare și scădere
5	**<< >>**	Deplasare stânga și dreapta
6	**< <= > >=**	Comparație inegalitate
7	**== !=**	Comparație egalitate
8	**&**	ȘI logic (pe biți)
9	**^**	SAU exclusiv (pe biți)
10	**|**	SAU (pe biți)
11	**&&**	ȘI logic
12	**||**	SAU logic
13	**? : = += - = *= /= %= &= |= ^= <<= >>=**	Expresia condițională și operatorii de atribuire
14	**,**	Operatorul virgulă

1. **Operatorii paranteză**

- Sunt întâlniţi în două situaţii distincte:

  ( ) – generali pentru operaţii şi funcţii;

  [ ] – pentru matrici.