

 Universitatea "Politehnica" din București
 Facultatea de Electronică, Telecomunicații și
 Tehnologia Informației



Programarea Calculatoarelor (limbajul C)

Curs 4 – Operatori și Modul de Evaluare al Expresiilor

Prof. Bogdan IONESCU

2016-2017

Cuprins

- 4.1. Operatori și modul de evaluare al expresiilor
- 4.2. Instrucțiuni de scriere și citire

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017

1/44

4.1. Operatori și modul de evaluare al expresiilor

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017

2/44

Operatorii folosiți în limbajul C

> Am vorbit până acum de variabile și constante, precum și de tipuri de date, dar cum ne folosim de valorile acestora pentru efectuarea anumitor calcule ?

→ pe baza **operatorilor**

> În limbajul C, spre deosebire de alte limbaje de programare în care operatorii sunt definiți folosind cuvinte cheie, operatorii sunt definiți folosind simboluri ce nu fac parte din alfabet, dar care sunt disponibile pe orice tip de tastatură.

→ acest lucru face ca limbajul C să fie mai compact, precum și mai accesibil pentru toate limbile.

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017

3/44

Operatorii folosiți în limbajul C (continuare)

> Limbajul C propune următorii operatori:

- A. operatorul de atribuire,
- B. operatori aritmetici,
- C. operatori de atribuire mixtă,
- D. operatori de incrementare și decrementare,
- E. operatori relaționali,
- F. operatori logici,
- G. operatorul virgulă,
- H. operatori de lucru cu biți,
- I. operatorul de forțare a tipului,
- J. operatorul sizeof(),

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017

4/44

Operatorii folosiți în limbajul C (continuare)

A. Operatorul de atribuire

> În limbajul C atribuirea unei valori unei variabile se face folosind "=".

Exemplu: $a = 3;$ \Leftrightarrow a ia valoarea 3
 $a = b;$ \Leftrightarrow a ia valoarea lui b

B. Operatorii aritmetici

- **operatori clasici:** "+" (adunare), "-" (scădere), "*" (înmulțire) și "/" (împărțire),
- **operatorul modulo:** "%" (restul împărțirii întregi, doar pentru numere întregi)

Exemplu: $5 \% 3 = 2$ pentru numere întregi
 $3 \% 1 = 0$

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017

5/44

Operatorii folosiți în limbajul C (continuare)

B. Operatorii aritmetici (continuare)

- împărțirea întregă: “/” (același operator de împărțire)
Cum se specifică că este vorba de o împărțire întregă ?

Exemplu: $13 / 3 = 4$
 $13.0 / 3 = 4.33$

dacă numerele sunt întregi →
împărțire întregă,
dacă un număr este real →
împărțire reală.

Exemple:

```
...  
float x;  
x=23/3; ???  
...
```

x=7.00

```
...  
float x=3.5,y;  
y=x/5; ???  
...
```

y=0.7

Operatorii folosiți în limbajul C (continuare)

C. Operatorii de atribuire mixtă

- > Există o modalitate de scriere prescurtată a operațiilor de atribuire pentru a facilita utilizarea lor directă în alte expresii.

Formă generală:

+ , - , * , / , %

<variabilă> <operator> = <expresie>

Rezultat:

<variabilă> = <variabilă> <operator> <expresie>

Exemple:

i+=3;	→ i=i+3;
x*=y+2;	→ x=x*(y+2);
a%=14;	→ a=a%14;

Operatorii folosiți în limbajul C (continuare)

D. Operatorii de incrementare și decrementare

- > Aceștia sunt: “++” și respectiv “--”.
↳ y++; este echivalentul scrierii y=y+1;
y--; este echivalentul scrierii y=y-1;

> Motivație: sunt mai rapizi decât executarea clasică a operației de atribuire.

> Pot fi utilizați atât ca prefix cât și ca sufix, eficientizând astfel scrierea operațiilor (++x sau x++, --y sau y--).

Exemplu:

```
...  
x=((++z) - (w--)) % 100;  
...
```



```
z=z+1;  
x=(z-w) % 100;  
w=w-1;
```

Operatorii folosiți în limbajul C (continuare)

E. Operatorii relaționali

> În limbajul C se folosesc următorii operatori relaționali:

- operatori clasici: “>” (mai mare), “<” (mai mic), “<=” (mai mic sau egal) și “>=” (mai mare sau egal),
- operatorul egal: “==” (în sens logic)

Exemplu: dacă (x==y) → adevărat dacă x are valoarea y,
→ fals dacă x diferit de y.

Atenție: dacă (x=y) → x ia valoarea lui y
→ se evaluează această valoare
→ dacă este diferită de 0 atunci
adevărat, altfel fals.

Operatorii folosiți în limbajul C (continuare)

E. Operatorii relaționali (continuare)

- operatorul diferit: “!=”

F. Operatorii logici

și logic - “&&”

0 && 0 = 0
1 && 0 = 0
0 && 1 = 0
1 && 1 = 1

(10>3) && (15<4) = 0 (Fals)

(3==3) && (5>4) = 1 (Adevărat)

Operatorii folosiți în limbajul C (continuare)

F. Operatorii logici (continuare)

sau logic - “||”

0 0 = 0
1 0 = 1
0 1 = 1
1 1 = 1

(10>3) || (15<4) = 1 (Adevărat)

(3==4) || (5<4) = 0 (Fals)

negația - “!”

!0 = 1
!1 = 0

!(10>3) = 0 (Fals)

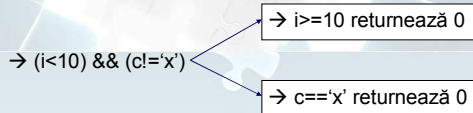
!((3==3) && (5>4)) = 0 (Fals)

Operatorii folosiți în limbajul C (continuare)



F. Operatorii logici (continuare)

P *Enunț:* i este o variabilă întreagă și c este o variabilă alfanumerică (caracter). Să se scrie condiția logică care returnează valoarea 1 dacă i mai mic ca 10 și c diferit de 'x', și respectiv valoarea 0 dacă i mai mare sau egal cu 10 sau c are valoarea 'x':



Operatorii folosiți în limbajul C (continuare)

G. Operatorul virgulă

> Operatorul „,” este folosit pentru a separa două sau mai multe expresii ce sunt incluse într-o formulare în care de regulă se așteaptă doar o expresie.

Formă generală:

(<expresie1>, ..., <expresieN>, <expresie>);

> Se evaluează toate expresiile, dar în cazul în care se folosește valoarea operatorului atunci doar ultima expresie este returnată.

Exemplu:

a = (b=3, i=i+1, b+2);

→ b←3, i←i+1, a←b+2=5

Operatorii folosiți în limbajul C (continuare)

H. Operatorii de lucru cu biți

> Aceștia modifică variabilele pe baza modificării valorilor binare ce corespund acestora.

Operator	Echivalent	Descriere
&	ȘI	operația logică ȘI pe biți
	SAU	operația logică SAU pe biți
^	SAU excl.	operația logică SAU excl. pe biți
~	negație	operația de inversare biți
<<	SHL	operația de shiftare pe biți
>>	SHR	operația de shiftare pe biți

Operatorii folosiți în limbajul C (continuare)

I. Operatorul de forțare a tipului (casting)

> Acesta permite conversia temporară a unei date de un anumit tip, într-un tip de bază ce este specificat de utilizator.

Formă generală:

(<tip_nou> <expresie>

sau C++

<tip_nou>(<expresie>)

ex.: x=float(y)/4;

Exemple:

```
int i;  
float f=3.14;  
i=(int) f;
```

i=3

```
float x;  
int y=9;  
x=(float) y/4;
```

x=2.25

Operatorii folosiți în limbajul C (continuare)

J. Operatorul sizeof()

> Acest operator acceptă un singur parametru și anume, fie un tip de dată fie o variabilă, și returnează dimensiunea în bytes a acestora.

> Este necesar la *alocarea dinamică* a memoriei pentru a specifica dimensiunea datelor.

Exemple:

```
int a;  
a=sizeof(char);
```

a=1

```
int i;  
double a;  
i=sizeof(a);
```

i=8

Precedența operațiilor

> În limbajul C toți operatorii au o prioritate, iar operatorii cu prioritate mai mare sunt evaluați înaintea celor cu prioritate mai mică.

> Operatorii cu aceeași prioritate sunt evaluați de la stânga la dreapta.

Exemplu: a - b - c este evaluat astfel: (a - b) - c

Exemplu: a=5+7%2 este evaluat așa? 5+(7%2)

corect

sau așa?

~~(5+7)%2~~

Precedența operațiilor (continuare)

> Prioritatea operatorilor în C (de la prioritatea maximă la cea minimă):

Level	Operator	Description	Grouping
1	::	scope	Left-to-right
2	() [] . -> ++ -- dynamic_cast static_cast reinterpret_cast const_cast typeid postfix		Left-to-right
3	* & -- ++ ! sizeof new delete	unary (prefix)	Right-to-left
4	* &	indirection and reference (pointers)	Right-to-left
5	++ --	unary sign operator	Right-to-left
6	(type)	type casting	Right-to-left
7	* * *	pointer-to-member	Left-to-right
8	* / %	multiplicative	Left-to-right
9	+ -	additive	Left-to-right
10	<< >>	shift	Left-to-right
11	< > <= >=	relational	Left-to-right
12	== !=	equality	Left-to-right
13	&	bitwise AND	Left-to-right
14	^	bitwise XOR	Left-to-right
15		bitwise OR	Left-to-right
16	&&	logical AND	Left-to-right
17		logical OR	Left-to-right
18	?:	conditional	Right-to-left
19	= * /= % = &= += -= >>= <<= <<= >>= =	assignment	Right-to-left
20	,	comma	Left-to-right

sursă www.cplusplus.com

Precedența operațiilor (continuare)

> Exemplu:

`a < 10 && 2 * b < c` ???

> Să vedem care sunt prioritățile:

nivelul 6 - * (înmulțirea)

nivelul 9 - <, > (relații)

nivelul 14 - && (și logic)

`(a < 10) && ((2 * b) < c)`

> În general ca să fim siguri de execuția corectă a operațiilor este recomandat să folosim parantezele.

Evaluarea expresiilor

> Din evaluarea unei expresii în C rezultă o *valoare*. Tipul și implicit valoarea rezultatului unei expresii se stabilesc pe baza unor **reguli de conversie**.

> Vorbim de reguli de conversie în cazul în care în expresie apar constante, variabile și funcții de tipuri diferite.

> Regulile de conversie de evaluare a unei expresii **ce nu implică operatorul de atribuire** sunt următoarele:

R1. toate datele de tip **char** și **short int** sunt convertite la **int**.
Toate datele de tip **float** sunt convertite la **double**.

Evaluarea expresiilor (continuare)

> Regulile de conversie de evaluare a unei expresii **ce nu implică operatorul de atribuire** sunt (continuare):

R2. pentru toate perechile de operanzi, dacă unul dintre operanzi este de tip **long double**, celălalt operand este convertit la **long double**.

R3. dacă unul dintre operanzi este de tip **double**, celălalt operand este convertit la **double**.

R4. dacă unul dintre operanzi este **long**, celălalt operand este convertit la **long**.

R5. dacă unul dintre operanzi este **unsigned**, celălalt operand este convertit la **unsigned**.

Evaluarea expresiilor (continuare)

> Pentru operația de atribuire, regula de conversie de tip este:

RA. valoarea din partea dreaptă a operatorului de atribuire ("=") este convertită la tipul variabilei din partea stângă. Dacă această conversie nu este posibilă, compilatorul va semnală o eroare.

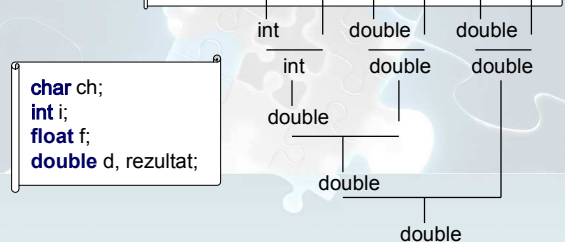
Concluzie: întotdeauna are loc evaluarea expresiei din partea dreaptă aplicându-se regulile de conversie R1-R5 și după aceea se aplică regula RA pentru operația de atribuire.

R1 → R2 → R3 → R4 → R5 → RA

Evaluarea expresiilor (continuare)

> Exemplu:

`rezultat = (ch / l) + (f * d) - (f + l);`



```

char ch;
int l;
float f;
double d, rezultat;
    
```

4.2. Instrucțiuni de scriere și citire

Instrucțiuni de scriere și citire a datelor

> În această secțiune vom discuta despre instrucțiunile clasice de scriere și citire a datelor în limbajul C.

→ acestea se găsesc în biblioteca `<stdio.h>`

> Vom vorbi despre:

- instrucțiunea **printf** (scriere date),
- instrucțiunea **scanf** (citire date).

Instrucțiuni de scriere a datelor

Instrucțiunea printf

> Instrucțiunea **printf** scrie pe dispozitivul standard de ieșire (*stdout*, de regulă ecran), o secvență de date formatată.

Prototipul funcției:

```
int printf (const char *format, a, b, 15, ... );
```

format (șir de caractere) specifică modul de afișare (*formatare*) cât și tipul datelor afișate

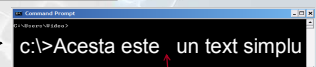
după „,” se specifică variabilele sau constantele anunțate în *format*.

Instrucțiuni de scriere a datelor (continuare)

Instrucțiunea printf (continuare)

Exemplu simplu:

```
printf("Acesta este un text simplu");
```



> Modul de formatare (șirul de caractere) este specificat între “ și “ (precum un string),

> În cazul în care nu *inserăm* variabile sau constante nu se mai folosește enumerarea de după “,”.

> Șirul de caractere furnizat este afișat **întocmai** (respectând spațiile existente).

Instrucțiuni de scriere a datelor (continuare)

Instrucțiunea printf (continuare)

> Șirul de caractere specificat poate conține atât text cât și **parametri de afișare** ce vor fi interpretați și substituiți la afișare cu argumentele specificate după “,” (variabile, valori, ...).

> Modul de specificare al parametrilor de afișare este următorul:

```
%[indicatori][lățime][.precizie][lungime]specificator
```

[] înseamnă opțional

• **caracterul special %**: specifică faptul că vrem să afișăm o variabilă sau constantă. Este înlocuit la afișare cu valoarea argumentului corespunzător de după “,”

Instrucțiuni de scriere a datelor (continuare)

Instrucțiunea printf (continuare)

```
%[indicatori][lățime][.precizie][lungime]specificator
```

• **specificator**: (obligatoriu) specifică tipul și modul de afișare a valorii argumentului.

> Cei mai frecvenți folosiți specificatori sunt:

c – tipul caracter (char)

d sau **i** – tipul întreg (decimal: signed int)

e sau **E** – tipul real în scrierea științifică

f – tipul real (float)

s – tipul șir de caractere (string)

% – se afișează caracterul rezervat ‘%’

$3.9265\mathbf{e}-2 = 3.9265 \times 10^{-2}$

Instrucțiuni de scriere a datelor (continuare)

Instrucțiunea printf (continuare)

`%[indicatori][lățime][.precizie][lungime]specificator`

- **lățime:** (opțional) specifică numărul minim de caractere pe care va fi afișată valoarea argumentului. Surplusul de caractere alocate nefolosite este bordat cu spații.
- **indicatori:** (opționali) specifică să zicem "detalii" de afișare:
 - "-" - aliniere a textului la stânga pe lățimea specificată (implicit este la dreapta)
 - "+" - forțează afișarea semnului chiar dacă numărul este pozitiv,
 - "0" - bordează numărul cu zerouri la stânga în loc de spații dacă se specifică lățimea de afișare.

Instrucțiuni de scriere a datelor (continuare)

Instrucțiunea printf (continuare)

`%[indicatori][lățime][.precizie][lungime]specificator`

- **precizie:** (opțional)
 - **pentru valori întregi:** specifică numărul minim de cifre (digiți) cu care va fi afișată valoarea (nu trunchiază),
 - **pentru valori reale:** specifică numărul de cifre (digiți) de după virgulă (rotunjește),
 - **pentru șiruri de caractere:** specifică numărul maxim de caractere ce vor fi afișate (trunchiază),
- **lungime:** (opțional) specifică mai precis tipul datei:
 - h** - short int sau unsigned short int,
 - l** - long int sau unsigned long int.

Instrucțiuni de scriere a datelor (continuare)

Instrucțiunea printf (continuare)

Exemple:

```
printf("Caractere: %c %c", 'a', 65);  
c:\>Caractere: a A  
  
printf("Valori intregi: %i %ld", 1977, 650000);  
c:\>Valori intregi: 1977 650000  
  
printf("Precedat de spatii: %10d", 1977);  
c:\>Precedat de spatii:      1977
```

Instrucțiuni de scriere a datelor (continuare)

Instrucțiunea printf (continuare)

Exemple (continuare):

```
printf("Precedat de zerouri: %010d", 1977);  
c:\>Precedat de zerouri: 0000001977  
  
printf("Numere reale: %4.2f %+0e %E ", 3.1416, 3.1416, 3.1416);  
c:\>Numere reale: 3.14 +3e+000 3.141600E+000  
  
printf(" %s", "Un sir de caractere");  
c:\> Un sir de caractere
```

Instrucțiuni de scriere a datelor (continuare)

Instrucțiunea printf (continuare)

> Există și o serie de comenzi speciale de formatare:

\n - "new line": la întâlnirea acestei comenzi cursorul trece pe linie nouă (echivalent tastă Enter),

\t - "tab": la întâlnirea acestei comenzi cursorul avansează un anumit număr de caractere la dreapta (echivalent tastă Tab),

Exemplu:

```
int i=10;  
float x=3.14;  
printf("i=%d \n x=%f", i, x);  
i=10  
x=3.14
```

Instrucțiuni de scriere a datelor (continuare)

Instrucțiunea printf (continuare)

P **Enunț:** folosind instrucțiunea printf să se afișeze pe ecran valorile reale 3.12, 34.5, 400.0, 56.789, cu 3 zecimale, sub forma unui tablou (se folosește indentarea)

Soluție:

```
printf("Solutia este\n");  
printf("%8.3f \t %8.3f \n", 3.12, 34.5);  
printf("%8.3f \t %8.3f", 400.0, 56.789);  
3.120    34.500  
400.000  56.789
```

Instrucțiuni de citire a datelor

- Instrucțiunea **scanf**

> Instrucțiunea **scanf** permite citirea datelor de pe dispozitivul standard de intrare (*stdin*, de regulă tastatură), și le stochează în locațiile de memorie specificate de utilizator (pointeri).

Prototipul funcției:

```
int scanf (const char *format, &a, &b, ... );
```

format (șir de caractere) specifică tipul datelor citite.

după „,” se specifică unde vor fi stocate datele (adrese de memorie)

Instrucțiuni de citire a datelor (continuare)

- Instrucțiunea **scanf** (continuare)

Exemplu simplu:

```
scanf ("%d", &a);
```

- se așteaptă introducerea de la tastatură a unui număr întreg ce va fi stocat în variabila **a**.

> Modul de formatare a datelor (șirul de caractere **format**) este specificat între “ și “ (precum un string),

> După “,” urmează specificarea locației de memorie unde va fi stocată valoarea introdusă.

în memorie:

```
00110010 10110111 00000000
```

adresă 10 adresă 11 adresă 12

desemnată de **a** (valoare)

desemnată de **&a** (adresă)

Instrucțiuni de citire a datelor (continuare)

- Instrucțiunea **scanf** (continuare)

> Șirul de caractere **format**, ce specifică tipul datelor citite, poate conține următoarele informații:

- **spații goale** (“whitespace”): funcția va ignora toate spațiile goale indiferent de cantitate (aceasta include spațiile, tab și linie nouă),
- **alte caractere diferite de spațiu și %** : orice alt caracter ce nu este spațiu sau alt caracter ce face parte din specificarea formatului datelor are ca efect:

- citirea de la tastatură a unui caracter, și compararea acestuia cu caracterul în cauză,
- dacă ==, nu este luat în calcul și funcția continuă cu următorul caracter din șirul **format**, dacă != exit.

Instrucțiuni de citire a datelor (continuare)

- Instrucțiunea **scanf** (continuare)

> Șirul de caractere **format**, ce specifică tipul datelor citite, poate conține următoarele informații (continuare):

- **specificatori de format**: aceștia specifică tipul și modul în care vor fi introduse datele:

[] înseamnă opțional

```
%[*][lățime][modificatori]tipdate
```

- **caracterul special %**: specifică faptul că vrem să citim o anumită valoare de la tastatură. Aceasta este stocată la adresa specificată după “,”.

→ numărul de valori = numărul de adrese.

Instrucțiuni de citire a datelor (continuare)

- Instrucțiunea **scanf** (continuare)

```
%[*][lățime][modificatori]tipdate
```

- **caracterul asterix ***: (opțional) specifică faptul că datele primite de la dispozitivul de intrare vor fi ignorate (nu vor fi stocate).
- **lățime**: (opțional) specifică numărul maxim de caractere (alfanumerice) ce vor fi citite de la tastatură.
- **tipdate**: (obligatoriu) specifică tipul datelor ce vor fi citite. Se folosește aceeași convenție ca la instrucțiunea **printf**:

c – tip caracter (char), citește un caracter, dacă [lățime] diferit de 1, funcția citește N=lățime caractere și le stochează în locații succesive ale vectorului specificat ca argument.

Instrucțiuni de citire a datelor (continuare)

- Instrucțiunea **scanf** (continuare)

```
%[*][lățime][modificatori]tipdate
```

- **tipdate**: (continuare) :

- d** – tip întreg cu semn (decimal integer),
- u** – tip întreg fără semn (unsigned decimal integer),
- f, e, E** – tip real în forma standard sau științifică (conversie implicită la float),
- s** – tip șir de caractere (char *).

- **modificatori**: (opțional) specifică mai precis tipul datei:

- h** - short int sau unsigned short int,
- l** - long int sau unsigned long int sau double pentru **f,e,E**.

Instrucțiuni de citire a datelor (continuare)

- Instrucțiunea **scanf** (continuare)

Exemple:

```
int x,y;
scanf("a%d %d", &x, &y);
printf("x=%d, y=%d", x, y);
```

c:\> (tastăm 'b' enter) → ??? stop

c:\> (tastăm 'a' enter '2' enter '4' enter) → ???

```
int x=20;
scanf("%*d", &x);
printf("x=%d", x);
```

x=2, y=4

c:\> (tastăm "33" enter) → ??? x=20

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 42/44

Instrucțiuni de citire a datelor (continuare)

- Instrucțiunea **scanf** (continuare)

Exemple:

```
int x;
scanf("%2d", &x);
printf("x=%d", x);
```

c:\> (tastăm "12345" enter) → ??? x=12

```
int a;
float y;
scanf("%d %f", &a, &y);
printf("a=%d, y=%f", a,y);
```

a=10, y=0.130000

c:\> (tastăm "10" enter "13e-2" enter) → ???

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 43/44

Sfârșitul Cursului 4

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 44/44