Expresii și operatori

CURS NR. 1/2

Expresii și operatori

Expresii

- Sunt formate din **operanzi** și **operatori**
  - Arată modul de calcul al unor valori
  - Cea mai simplă expresie este formată dintr-un singur operand

Operatori

- Elemente fundamentale ale expresiilor
  - Ex. Operatori aritmetici, relaționali, etc.
- C are mulți operatori

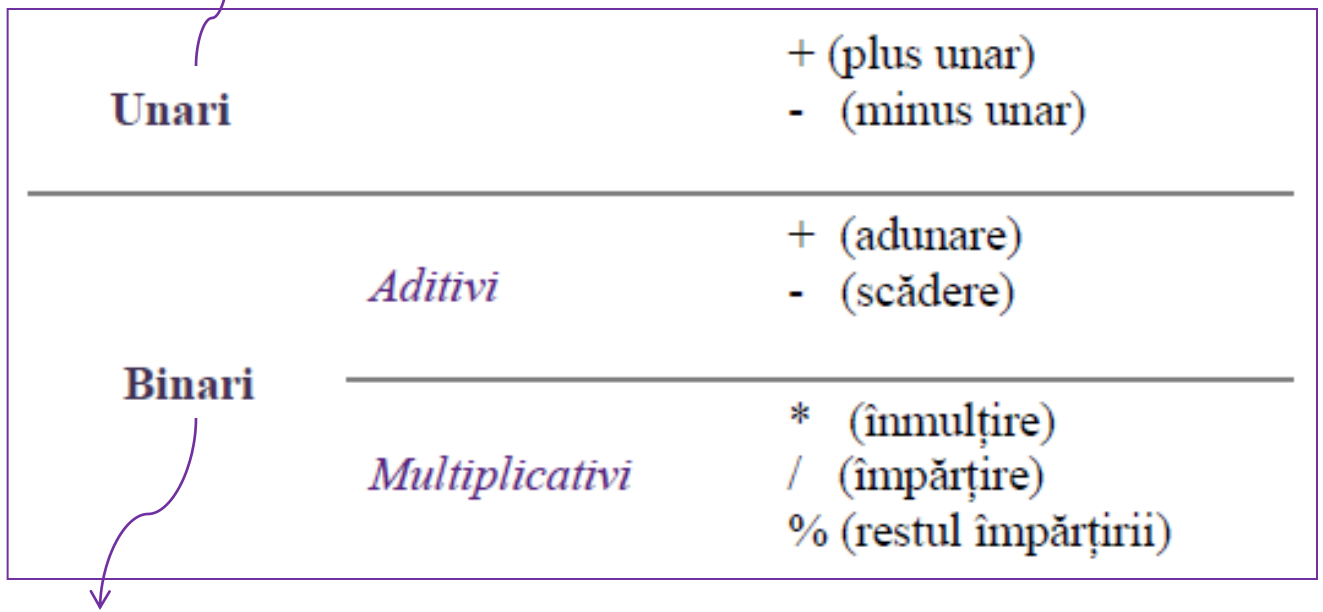
Operanzi

- Variabilă, constantă, etc.
- Apel de funcție, identificatorul unui tip de data, etc.
- Expresie între paranteze

Expresii aritmetice și operatori aritmetici

!

Se aplică asupra unui singur operand



Necesită doi operanzi

Expresii aritmetice și operatori aritmetici

!

Se poate aplica asupra operanzilor

- de tip întreg (int, char) sau
- de tip real (float sau double)

- Se pot combina aceste tipuri în aceeași expresie
  - **Excepție**: % doar între întregi

Exemple de expresii aritmetice:

```
int a, b, c = +3;      // operatorul unar +
b = -4;                // operatorul unar -
a = b - c + 1;         // operatorul binar - și +    a este -6
a = a * b / 2;         // operatorul binar * și /    a este 12
c = a % 5;             // operatorul binar %         c este 2
```

Expresii aritmetice și operatori aritmetici

!

Observații

- Operatorul **/** semnifică
  - **împărțirea întreagă**
    - dacă ambii operanzi sunt întregi (int)
  - **împărțirea cu rest**
    - dacă cel puțin unul dintre operanzi este de tip real (float, double)

Exemple

```
int a = 5, b = 2;
float x = 5.0f;
a = a / b;      // a este 2
x = x / b;      // x este 2.5
x = 5 / b;      // x este 2.0
```

Expresii aritmetice și operatori aritmetici

Observații

- Împărțirea la zero !! - Eroare
  - Operatorii / și % nu pot avea operandul din dreapta 0
- **Trunchierea** la împărțirea întreagă
  - C89 – dependent de implementare
  - **C99** – trunchiere către 0

Exemple

```
c =   7 /  5;       // c este  1 (trunchiat de la 1.4)
c =  -7 /  5;       // c este -1 (trunchiat de la -1.4)
c =   9 /  5;       // c este  1 (trunchiat de la 1.8)
c =   9 / -5;       // c este -1 (trunchiat de la -1.8)
```

Evaluarea expresiilor

Introducem principiile fundamentale pentru evaluarea oricăror expresii prin intermediul expresiilor aritmetice

- Mai ușor de înțeles

Precedența și **asociativitatea** operatorilor

- Dacă într-o expresie apar mai mulți operatori,
  atunci evaluarea expresiei respectă **ordinea de precedență** a operatorilor

- Dacă într-o expresie apar mai mulți operatori de aceeași prioritate,
  atunci se aplică **regula de asociativitate** a operatorilor

Ordinea de precedență

- Ordinea de precedență a operatorilor aritmetici

Cei mai prioritari	+ (plus unar) - (minus unar)
	* (înmulțire) / (împărțire) % (restul împărțirii)
Cei mai puțin prioritari	+ (adunare) - (scădere)

Exemple:

```
a + b * c       este echivalent cu    a + (b * c)
-a * b - c      este echivalent cu    ((-a) * b) - c
+a - b / c      este echivalent cu    (+a) - (b / c)
```

Regula de asociativitate

Regula de asociativitate a operatorilor aritmetici

- Un operator este **asociativ la stânga** dacă se grupează de la stânga la dreapta
  - Exemple: toți operatorii aritmetici binari (+, -, *, /, %)

Exemple:

```
a + b - c  este echivalent cu    (a + b) - c
a * b / c  este echivalent cu    (a * b) / c
```

- Un operator este **asociativ la dreapta** dacă se grupează de la dreapta la stânga
  - Exemple: operatorii aritmetici unari (+, -)

Exemple:

```
- + a            este echivalent cu          -(+a)
+ - a            este echivalent cu          +(-a)
```

Operatori de atribuire

Operatorul de atribuire simplă =

- Efect: evaluarea expresiei din dreapta operatorului și asignarea acestei valori la variabila din stânga operatorului

Exemple:

```
a = 10;                 // a ia valoarea 10
b = a;                  // b ia valoarea 10
c = a + (b-7) * 3;      // c ia valoarea 19
```

Valoarea unei atribuiri `var = expresie` este valoarea lui `var` după asignare

- Obs: expresia de atribuire poate apare ca operand într-o altă expresie unde se așteaptă o valoare de tipul lui `var`
  - Notă: expresia devine greu de înțeles și poate introduce erori greu de depistat

Exemplu:

```
a = 3;
b = 5 - (c = a);        // c ia valoarea 3 care
                        // se scade din 5 și astfel b devine 2
```

Operatori de atribuire

Atribuirea formalizată: `expr1 = expr2`

- Atenție
  - expr1 este *lvalue* (valoare stânga)
    - trebuie să permită stocarea valorii lui expr2 în memorie
      - Corect: a=10
      - Incorect: 10 = a

Dacă tipul lui `expr1` nu este același cu tipul lui `expr2` atunci se aplică regula conversiei implicite

- Valoarea lui `expr2` este convertită la tipul lui `expr1` în momentul asignării

Exemple:

```
int a;
float x;
a = 12.34f;          // a ia valoarea 12
x = 123;             // x ia valoarea 123.0
```

Operatori de atribuire

Regula de asociativitate

- Operatorul de atribuire este asociativ dreapta
  - Atribuirile se pot înlănțui

```
a = b = c = 0;
```

Operatori de atribuire compuși

  - Ex. : +=, -+, *=, /=, %=, șamd. (combinat cu operatori pe biți)
- permit calcularea noii valori a variabilei folosind valoarea veche a acesteia

```
a += 1;          // a se incrementează cu 1: a = a + 1;
b -= 3;          // asemănător cu b = b - 3;
c *= 4;          // asemănător cu c = c * 4;
```

- Dar nu este întotdeauna echivalent cu varianta descompusă
  - Contează ordinea de precedență și efectele secundare

```
a *= b + c;      // nu este echivalent cu a = a * b + c;
                 // este echivalent cu  a = a * (b + c);
```

Operatori de incrementare și decrementare

Operatorii ++ și --

- Incrementarea / decrementarea unei variabile cu 1

Exemplu:

```
++i;
```

Exemplu echivalent:

```
i = i + 1;
i += 1;
```

Forma prefixă (ex. ++i sau --i)

- Preincrementare / predecrementare

Forma postfixă (ex. i++ sau i--)

- Postincrementare / postdecrementare

Efect secundar: modificarea valorii operandului

Valoarea returnată

- preincrementarea (++a) returnează valoarea a+1
- postincrementarea (a++) returnează valoarea a

Operatori de incrementare și decrementare

!

Operatorii de **pre**incrementare și **pre**decrementare
au aceeași prioritate ca și operatorii unari + și - și sunt asociativi dreapta

Operatorii de **post**incrementare și **post**decrementare
sunt mai prioritari decât operatorii unari + și - și sunt asociativi stânga

```
int a = 5, b = 2, c;
c = a - ++b;          // ⇔ b = b+1;  c = a-b;
                      // valorile a: 5, b: 3, c: 2
c = ++a + b--;        // ⇔ a = a+1;   c = a + b;  b = b-1;
                      // valorile a: 6, b: 2, c: 9
```

Evitați expresiile cu multiple efecte secundare. Programul devinde greu de trasat și modificat

Expresii logice

Expresiile logice se evaluează la valori de tip *adevărat* sau *fals*

Sunt construite cu ajutorul a trei categorii de operatori

- Operatori **relaționali**
- Operatori de **egalitate**
- Operatori **logici**

C tratează valorile adevărat și fals ca **valori întregi**

- 0 înseamnă fals
- 1 și orice altă valoare nenulă se interpretează ca adevărat

Operatori relaționali

Operatorii <, >, <=, >=

Produc ca rezultat o valoare logică, adică valoarea 0 (fals) sau 1 (adevărat)

Sunt mai puțin prioritari decât operatorii aritmetici și sunt asociativi stânga

Exemple:

```
5   < 10          // rezultat: 1
10  <  5          // rezultat: 0
3   > 2.5         // rezultat: 1
                  // se pot combina tipurile întreg și real
a + b <= c - 1    // este de fapt (a + b) <= (c - 1)
                  // respectând ordinea de precedență
```

```
a < b < c  // echivalent cu (a < b) < c
           // datorita asociativitatii stanga
```

Operatori relaționali

Operatorii <, >, <=, >=

Produc ca rezultat o valoare logică, adică valoarea 0 (fals) sau 1 (adevărat)

Sunt mai puțin prioritari decât operatorii aritmetici și sunt asociativi stânga

Atenție: NU testează dacă valoarea lui `b` este între `a` și `c` !!!!

Exemple:

```
5   < 10          // rezultat: 1
10  <  5          // rezultat: 0
3   > 2.5         // rezultat: 1
                  // se pot combina tipurile întreg și real
a + b <= c - 1    // este de fapt (a + b) <= (c - 1)
                  // respectând ordinea de precedență
```

```
a < b < c   // echivalent cu (a < b) < c
            // datorita asociativitatii stanga
```

Operatori de egalitate

!

Testează egalitatea dintre două valori

- Nu îl confundați cu operatorul de atribuire (=)

== este operatorul ”egal cu”,

!= este operatorul ”diferit de”

Generează o valoare logică: 0 (fals) sau 1 (adevărat)

Sunt asociativi stânga

În ordinea de precedență a operatorilor sunt mai puțin prioritari decât operatorii relaționali

```
a == 2           // returnează 1 dacă a este 2,
                 // 0 în caz contrar
a != b           // returnează 1 dacă a nu este egal cu b,
                 // 0 dacă a și b au valori identice
a < b == b < c   // este echivalent cu (a < b) == (b < c)
                 // returnează 1 doar dacă expresiile au
                 // aceeași valoare:
                 // ambele sunt adevărate sau ambele false
```

Operatori logici

C furnizează trei operatori logici

- **!** **Negare logică** - operator unar

```
!expr       // 1 dacă expr are valoarea logică 0 (fals)
            // 0 dacă expr are valoarea logică nenulă (adevărat)
```

- && **ȘI logic**: operator binar

```
expr1 && expr2   // este 1 dacă expr1 și expr2 sunt nenule
```

- || **SAU logic**: operator binar

```
expr1 || expr2  // este 1 dacă expr1 sau expr2 este nenulă
```

Generează o valoare logică: 0 (fals) sau 1 (adevărat)

Operatori logici

Evaluarea

- Dacă se poate deduce rezultatul global al expresiei compuse, atunci expresia din dreapta nu se mai evaluează

Exemplu:

```
(a != 0) && (a % 4 == 0)
```

Operatorul ! (negare) are prioritate egală cu cea a operatorilor aritmetici unari (+ și -)

Operatorii && și || sunt mai puțin prioritari decât operatorii relaționali și cei de egalitate

Operatori pe biți

!

Două categorii

- operatori **logici pe biți**
  - & (ȘI pe biți), | (SAU pe biți), ^ (SAU EXCLUSIV pe biți),
    ~ (complement față de 1)

- operatori de **deplasare pe biți**
  - << (deplasare stânga pe biți) și >> (deplasare dreapta pe biți)

Se pot aplica doar asupra operanzilor de tip întreg

Ordinea de precedență - în cadrul acestei categorii

Cei mai prioritari	~	(complement față de unu)
	<<	(deplasare stânga)
	>>	(deplasare dreapta)
	&	(și pe biți)
	^	(sau exclusiv pe biți)
Cei mai puțin prioritari	|	(sau pe biți)

Operatori logici pe biți

!

& (ȘI logic pe biți) respectiv | (SAU logic pe biți) seamănă cu && respectiv ||

- Rol similar, dar la nivelul fiecărei perechi de biți de pe poziții corespunzătoare

~ (complement față de unu) este echivalentul operatorului logic !

- dar aplicat la nivel de biți

Expresie	Reprezentare pe 4 biți				Observație
a = 10	1	0	1	0	
b = 7	0	1	1	1	
a & b	0	0	1	0	1 dacă ambi biți sunt 1, 0 în rest
a | b	1	1	1	1	1 dacă cel puțin unul din cei doi biți este 1, 0 în rest
a ^ b	1	1	0	1	1 dacă doar unul din cei doi biți este 1, 0 în rest
~ a	0	1	0	1	1 unde bitul a fost 0 și 0 unde bitul a fost 1
~ b	1	0	0	0	1 unde bitul a fost 0 și 0 unde bitul a fost 1

Operatori de deplasare pe biți

!

Condiții

- operanzi întregi
- al doilea operand cu valoare mai mică (nu negativ) decât numărul de biți pe care este reprezentat operandul din stânga

Deplasarea spre stânga ⇔ înmulțire cu 2 la puterea deplasamentului

Deplasarea spre dreapta ⇔ împărțire cu 2 la puterea deplasamentului

Expresie	Reprezentare binară	Observație
a = 12	0000 0000 0000 1100	
b = 3600	0000 1110 0001 0000	
a << 1	0000 0000 0001 1000	Valoarea rezultată este 24 = 12 * 2^1
a << 2	0000 0000 0011 0000	Valoarea rezultată este 48 = 12 * 2^2
a << 5	0000 0001 1000 0000	Valoarea rezultată este 384 = 12 * 2^5
a >> 1	0000 0000 0000 0110	Valoarea rezultată este 6 = 12 / 2^1
a >> 2	0000 0000 0000 0011	Valoarea rezultată este 3 = 12 / 2^2
b >> 4	0000 0000 1110 0001	Valoarea rezultată este 225 = 3600 / 2^4

Alți operatori

Operatorul de **acces la elementele tabloului [ ]**

- Primul în ordinea de precedență

```
int a[100];
a[5] = 10;
```

Operatorul **adresă &** și operatorul de **dereferențiere ***

- Strâns legat de pointeri

```
int a, *p;          // p este un pointer la int
p = &a;             // p este pointer la a
*p = 3;             // valoarea lui a devine 3
```

Operatorul **sizeof**

```
sizeof(a)           // este numărul de octeți
                    // ocupați în memorie de a
```

Operatorul de **conversie explicită**

```
int a = 1, b = 2;
float media;

media = ( a + (float)b ) / 2;     // media devine 1.5
media = ( a + b ) / 2;            // media devine 1.0  - incorect!
```

Alți operatori

Operatorul condițional - operator ternar

- Similar cu instrucțiunea if
- *expresie1 ? expresie2 : expresie3*

```
int a=3, b=5, max;
max = a > b ? a : b;
```

```
a % 2 ? printf("numar impar") : printf("numar par");
```

Operatorul virgulă ,

- Evaluarea secvențială a expresiilor (de la stg. la dreapta)
- Valoarea ultimei expresii din înlănțuire este valoarea expresiei compuse
- Cel mai puțin prioritar din lista de precedență

Ordinea de precedență și asociativitate

!

Ordine	Categorie	Operație	Operator	Asociativitate
1	Operatori postfixați	accesare elemement din tablou	`[ ]`	Stânga (stg -> dr)
		apel de funcție	`( )`	
		accesare componentă din structură	`.` `->`	
		post-incrementare / post-decrementare	`++` `--`	
2	Operatori unari	pre-incrementare / pre-decrementare	`++` `--`	Dreapta (dr -> stg)
		semn plus	`+`	
		semn minus	`-`	
		negare logică	`!`	
		negare pe biți	`~`	
		adresă	`&`	
		dereferențiere	`*`	
		dimensiune	`sizeof`	
3	Conversie	conversie explicită	`(tip)`	Dreapta

Ordinea de precedență și asociativitate

!

Ordine	Categorie	Operație	Operator	Asociativitate
4	Operatori multiplicativi	înmulțire, împărțire, rest	* / %	Stânga
5	Operatori aditivi	adunare, scădere	+ -	Stânga
6	Operatori de deplasare	deplasare stânga, deplasare dreapta	<< >>	Stânga
7	Operatori relaționali	mai mic, mai mare, mai mic sau egal, mai mare sau egal	< > <= >=	Stânga
8	Operatori de egalitate	egal, diferit	== !=	Stânga
9	Operatori logici pe biți	și pe biți	&	Stânga
10		sau exclusiv pe biți	^	
11		sau pe biți	|	
12	Operatori logici	și logic	&&	Stânga
13		sau logic	||	Stânga

Ordinea de precedență și asociativitate

!

Ordine	Categorie	Operație	Operator	Asociativitate
14	Operator ternar	operatorul condițional expr_cond ? expr_da : expr_nu	? :	Dreapta
15	Operatori de atribuire	atribuire simplă, atribuire compusă aritmetică atribuire compusă pe biți	= += -= *= /= %= &= |= ^= <<= >>=	Dreapta
16	Operator de secvențializare	operatorul virgulă	,	Stânga

Conversii implicite

Context

- Este permisă combinarea mai multor operanzi de tipuri diferite într-o singură expresie

Problema

- Operatorii binari (care se aplică asupra a doi operanzi) cer ca tipul operanzilor să fie același pentru a putea efectua operația

Soluție: conversia implicită

- Compilatorul convertește valorile operanzilor la același tip într-un mod transparent programatorului înaintea generării codului mașină
  - sau va genera un mesaj de eroare sugestiv

Alternativă

- Conversii explicite: `(tip)`

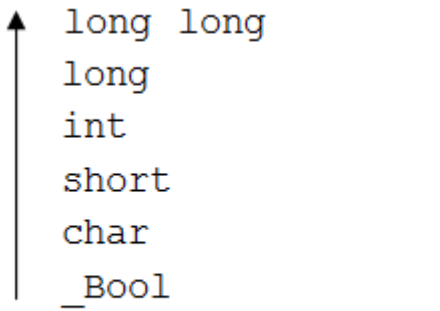
Conversii implicite

Conversii implicite la atribuire

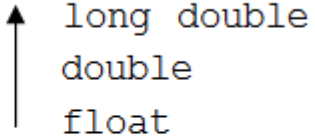
- Valoarea expresiei din dreapta se convertește la tipul expresiei din stânga
  - Probleme
    - Pot apare pierderi – dacă tipul nu este suficient de încăpător
    - Nu poate efectua conversia

Conversia implicită a tipurilor aritmetice

- convesia operanzilor se face la cel mai apropriat tip care poate reprezenta valorile ambilor operanzi



Ierarhia tipurilor întregi standard



Ierarhia tipurilor reale standard

Conversii implicite

Reguli și observații

- Tipul care se reprezintă pe un număr mai mare de octeți are un rang mai mare în ierarhie
- Pentru același tip, varianta fără semn are rang mai mare decât cea cu semn
- Tipurile reale au rang mai mare decât tipurile întregi
- În cazul în care toți operanzii sunt întregi se efectuează promovarea întregilor
  - În standardul C99 mai întâi se efectuează o promovare a întregilor cu rang mai mic de int (_Bool, char, short) la int sau unsigned int

```
char c = 'a';
short sh = 140;
int a = 3, b;
unsigned int u = 1234567u;
long i = 300L;
float f = 80.13f;
double d = 5.75, g;
```

```
b = a + sh;   // val. lui sh convertita la int
a = sh - c;   // val. lui sh si c convertite la int
g = d + f;    // val. lui f convertita la double
f = i + u;    // cal lui u convertita la long
              // rezultatul convertit la float
```

Reguli și recomandări

- Variabilele locale trebuie să fie inițializate explicit înainte de utilizare
  - Altfel pot conține valori neașteptate (valori reziduale ce se află în locațiile de memorie alocate variabilei)

```c
#include <stdio.h>

int main()
{
    int i;
    printf("i: %d \n", i);

    return 0;
}
```

- Valoarea expresiilor trebuie să fie independentă de ordinea de evaluare a operanzilor – deci atenție la operanzii cu efecte secundare!

De evitat:

```c
i = ++i + 1;

v[i++] = i;

a = i + v[++i];
```

```c
i = i + 1;

a[i] = i;

++i;
a = i + b[i];
```

Corect:
Valorile expresiilor sunt independente de ordinea de evaluare a operanzilor

Exemplu preluat de pe www.securecoding.cert.org

Reguli și recomandări

Folosirea parantezelor pentru evaluarea expresiilor compuse

- C are foarte mulți operatori - și regulile de precedență nu sunt întotdeauna intuitive
- Folosirea parantezelor adecvate poate reduce semnificativ riscul evaluării greșite a expresiilor
  - Facilitează înțelegerea și modificările ulterioare ale codului

Exemplu: Verificarea ultimului bit al variabilei x

Atenție:
== mai prioritar decât &

```
x & 1 == 0
```



```
x & (1 == 0)
```



```
x & 0
```

```
(x & 1) == 0
```

Corect:
Parantezele asigură că evaluarea se efectuează în modul așteptat

Excepție: expresiile algebrice

- Operatorii algebrici urmează cu strictețe ordinea cunoscută a operatorilor

Surse bibliografice

- K. N. King, C Programming – A Modern Approach, 2nd edition, W. W. Norton & Co., 2008
  - Capitolul 4
- Deitel & Deitel, C How to Program, 6th edition, Pearson, 2009
  - Capitolele 2, 3 și 4
- CERT - Secure Coding, Rules for expressions, https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=358