



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,
2016-2017

Cursul 3

InfO'Clock 2016-2017

- Începe săptămâna viitoare, cu ajutorul colegilor din anii 2 și 3;
- dacă participați și rezolvați teme + mergeți la concursuri obținând un punctaj bun (primiți detalii la InfO'Clock):
 - puteți opta să va echivalați seminarul (maxim 1 punct) + laboratorul (teme + test: 3+2 puncte = maxim 5 puncte) la Programare Procedurală (mai rămâne doar să dați examenul scris la curs);

SAU

- puteți opta să primiți notă separată pentru curs facultativ (contribuie la media generală, fără credite).

Recapitulare – cursul trecut

1. Introducere în limbajul C. Structura unui program C.
2. Complexitatea algoritmilor.
3. Tipuri de date fundamentale.

Programa cursului

□ Introducere

- Algoritmi.
- Limbaje de programare.
- Introducere în limbajul C. Structura unui program C.
- Complexitatea algoritmilor.

□ Fundamentele limbajului C



Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.

- Instrucțiuni de control
- Directive de preprocesare. Macrodefiniții.
- Funcții de citire/scriere.
- Etapele realizării unui program C.

□ Tipuri deriveate de date

- Tablouri. Siruri de caractere.
- Structuri, uniuni, câmpuri de biți, enumerări.
- Pointeri.

□ Funcții (1)

- Declarație și definire. Apel. Metode de transmisie a parametrilor.
- Pointeri la funcții.

□ Tablouri și pointeri

- Legătura dintre tablouri și pointeri
- Aritmetică a pointerilor
- Alocarea dinamică a memoriei
- Clase de memorare

□ Siruri de caractere

- Funcții specifice de manipulare.

□ Fișiere text și fișiere binare

- Funcții specifice de manipulare.

□ Structuri de date complexe și autoreferite

- Definire și utilizare

□ Funcții (2)

- Funcții cu număr variabil de argumente.
- Preluarea argumentelor funcției main din linia de comandă.
- Programare generică.

□ Recursivitate

Cuprinsul cursului de azi

1. Tipuri de date fundamentale
2. Variabile și constante
3. Expresii și operatori
4. Conversii

Tipuri de date fundamentale

- În C89, limbajul C are **cinci categorii fundamentale** de tipuri de date: **int**, **char**, **double**, **float**, **void**
 - tipul **întreg** – **int**: poate reține valori întregi: 2, 0, -532
 - tipul **caracter** – **char**: poate reține un singur caracter sub forma codului elementelor din setul de caractere specific (codul ASCII), sau numere întregi mici
 - tipul **real** (numere în virgulă mobilă) – **simplă precizie** – **float**: pot reține numere care conțin parte fracționară: 4971.185, -0.72561, 2.000, 3.14
 - tipul **real** (numere în virgulă mobilă) **în dublă precizie** – **double**: pot reține valori reale în virgulă mobilă cu o precizie mai mare decât tipul float
 - tipul **void**: indică lipsa unui tip anume

Reprezentarea în memorie

- **char**: ocupă 1 octet = 8 biți, valori între $-2^7 = -128$ și $2^7 - 1 = 127$

`char ch = 'a';` 'a' are codul ASCII 97

$$97 = 2^6 + 2^5 + 2^0 \text{ (scrierea în baza 2 a lui 97)}$$



- **int**: ocupă 4 octeți = 32 biți, valori între -2^{31} și $2^{31} - 1$

`int i = 190;`

$$190 = 2^7 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 \text{ (scrierea în baza 2 a lui 190)}$$

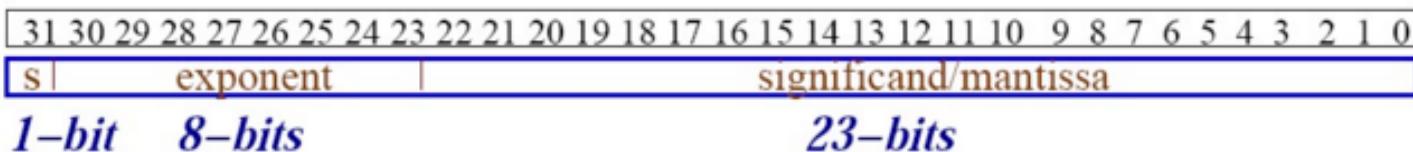


Reprezentarea în memorie



$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

- float: ocupă 4 octeți = 32 biți, precizie simplă, bias = 127



float f = 7.0; $7.0 = 1.75 * 4 = (-1)^0 * (1+0.75) * 2^{(129-127)}$

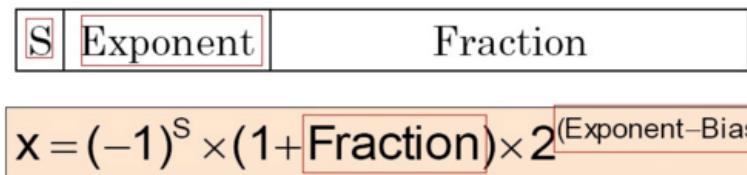
Reprezentare binara exponent: $129 = 128 + 1 = 2^7 + 2^0$

Reprezentare binara fractie: $0.75 = 0.5 + 0.25 = 2^{-1} + 2^{-2}$

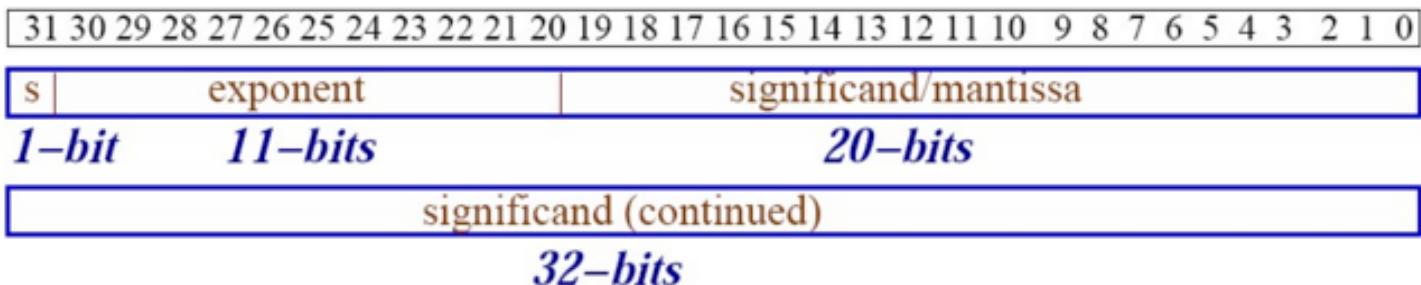


Reprezentarea binara a lui 7.0 (float) in memoria calculatorului

Reprezentarea în memorie



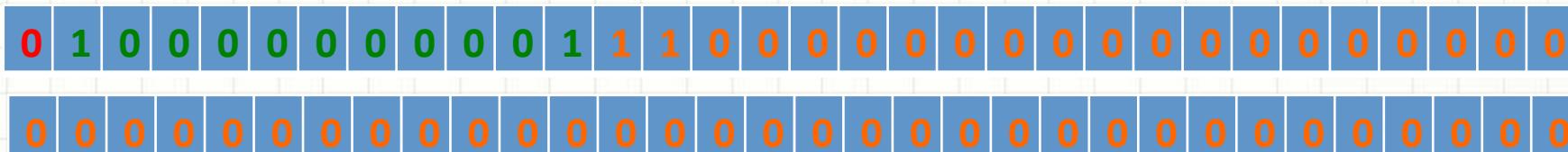
- **double**: ocupă 8 octeți = 64 biți, precizie dublă, bias = 1023



`double d = 7.0;` $7.0 = 1.75 * 4 = (-1)^0 * (1+0.75) * 2^{(1025-1023)}$

Reprezentare binara exponent: $1025 = 1024 + 1 = 2^{10} + 2^0$

Reprezentare binara fractie: $0.75 = 0.5 + 0.25 = 2^{-1} + 2^{-2}$



Există 10 tipuri de
studenți la FMI:
cei care înțeleg sistemul
binar și cei care nu îl
înțeleg

Modificatori de tip

- **signed**
 - modificadorul implicit pentru toate tipurile de date
 - bitul cel mai semnificativ din reprezentarea valorii este semnul
- **unsigned**
 - restricționează valorile numerice memorate la valori pozitive
 - domeniul de valori este mai mare deoarece bitul de semn este liber și participă în reprezentarea valorilor
- **short**
 - reduce dimensiunea tipului de date întreg la jumătate
 - se aplică doar pe întregi
- **long**
 - permite memorarea valorilor care depășesc limita specifică tipului de date
 - se aplică doar pe int sau double: la int dimensiunea tipului de bază se dublează, la double se mărește dimensiunea de regulă cu doi octeți (de la 8 la 10 octeți)
- **long long**
 - introdus în C99 pentru stocarea unor valori întregi de dimensiuni foarte mari

Tipuri de date + modificatori

Tip de date + modificator	Dimensiune în biți	Domeniu de valori
char	8	de la -128 la 127
unsigned char	8	de la 0 la 255
signed char	8	de la -128 la 127
int	32	de la -2^{31} la $2^{31}-1$
unsigned int	32	de la 0 la $2^{32}-1$
signed int	32	de la -2^{31} la $2^{31}-1$
short int	16	de la -2^{15} la $2^{15}-1$
unsigned short int	16	de la 0 la $2^{16}-1$
signed short int	16	de la -2^{15} la $2^{15}-1$
long int	64	de la -2^{63} la $2^{63}-1$
float	32	...
double	64	...

Signed vs. unsigned

- ❑ **signed int** – întreg cu semn (pozitiv sau negativ)
- ❑ **unsigned int** – întreg fără semn (pozitiv)

```
int i = 190;
```



A horizontal bar divided into 32 equal segments, each representing a bit of a 32-bit integer. The first segment contains a red '0', while all other 31 segments contain black '0's.

Reprezentarea binara a lui 190 in memoria calculatorului

$$190 = 2^7 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 \text{ (scrierea în baza 2 a lui 190)}$$

- ❑ cum se reprezintă -190 în memoria calculatorului?



A horizontal bar divided into 32 equal segments, each representing a bit of a 32-bit integer. The first segment contains a red '1', while all other 31 segments contain black '0's.

Reprezentarea binara a lui -190 in memoria calculatorului

- ❑ care este logica unei asemenea reprezentări?

Signed vs. unsigned

- ❑ **signed int** – întreg cu semn (pozitiv sau negativ)
 - ❑ **unsigned int** – întreg fără semn (pozitiv)



Signed vs. unsigned

- ❑ **signed int** – întreg cu semn (pozitiv sau negativ)
- ❑ **unsigned int** – întreg fără semn (pozitiv)

$$\begin{array}{r} 190 \\ + \\ -190 \\ \hline = \\ 0 \end{array}$$

Cuprinsul cursului de azi

1. Tipuri de date fundamentale
2. Variabile și constante
3. Expresii și operatori
4. Conversii

Variabile și constante

- stochează datele necesare programului
 - valorile stocate în memoria sistemului în mod transparent de către programator
 - referirea la aceste date se face prin numele lor simbolice, adică prin identificatori
- **variabilele** stochează date care pot fi modificate în timpul execuției
- **constantele** păstrează aceeași valoare (cea cu care au fost inițializate) până la terminarea programului

Variabile

- se characterizează printr-un nume (identifier), un tip, o valoare, adresa de memorie unde se află stocată valoarea variabilei, domeniu de vizibilitate
- oricărei variabile i se alocă un spațiu de memorie corespunzător tipului variabilei
- exemplu: `int notaExamen = 10;`
 - `int` = tipul variabilei (de obicei se va stoca pe 32 de biți)
 - `notaExamen` = numele variabilei
 - `10` = valoarea variabilei
 - `¬aExamen` = adresa din memorie unde se află stocată valoarea variabilei cu numele notaExamen

Domeniul de vizibilitate al variabilelor

- domeniul de vizibilitate al unei variabile = porțiunea de cod la carei execuție variabila respectivă este accesibilă
- variabile **locale** – vizibile local, numai în funcția sau blocul de instrucțiuni unde au fost declarate.
- variabile **globale** – vizibile global, din orice zonă a codului.
- parametri **formali** ai unei funcții se comportă asemenea unor variabile locale.

Variabile locale

- variabile definite în corpul unei funcții sau a unui bloc de instrucțiuni. Sunt locale (vizibile) acelei funcții sau bloc.

```
variabileLocale1.c
```

```
1 #include <stdio.h>
2
3 void f1()
4 {
5     int x=10;
6     printf("\nIn functia f1 valoarea lui x este %d \n",x);
7 }
8
9 void f2()
10 {
11     int x=20;
12     printf("In functia f2 valoarea lui x este %d \n",x);
13 }
14
15 int main()
16 {
17     int x = 30;
18     f1();
19     f2();
20     printf("In main valoarea lui x este %d \n \n",x);
21     return 0;
22 }
```

```
In functia f1 valoarea lui x este 10
In functia f2 valoarea lui x este 20
In main valoarea lui x este 30
```

Variabile locale

- variabile definite în corpul unei funcții sau a unui bloc de instrucțiuni. Sunt locale (vizibile) acelei funcții sau bloc.

```
variabileLocale2.c ✘
1 #include <stdio.h>
2
3 int main()
4 {
5     int i;
6     for(i=0;i<10;i++)
7     {
8         int j = 2*i;
9         printf("j = %d \n",j);
10    }
11    printf("j = %d \n",j); ←
12
13    return 0;
14 }
```

In function 'main':

11 error: 'j' undeclared (first use in this function)
11 error: (Each undeclared identifier is reported only

Eroare la linia 11: variabila j nu a fost declarată. Ea este vizibilă numai în blocul de instrucțiuni anterior.

Variabile locale

- variabile definite în corpul unei funcții sau a unui bloc de instrucțiuni. Sunt locale (vizibile) acelei funcții sau bloc.

```
variabileLocale2.c ✎
1 #include <stdio.h>
2
3 int main()
4 {
5     int i;
6     for(i=0;i<10;i++)
7     {
8         int j = 2*i;
9         printf("j = %d \n",j);
10    }
11    int j = i;
12    printf("j = %d \n",j);
13
14    return 0;
15 }
```

```
j = 0
j = 2
j = 4
j = 6
j = 8
j = 10
j = 12
j = 14
j = 16
j = 18
j = 10
```

Variabile globale

- ❑ se declară în afara oricărei funcții și sunt vizibile în întreg programul
- ❑ pot fi accesate de către orice zonă a codului
- ❑ orice expresie are acces la ele, indiferent de tipul blocului de cod în care se află expresia

Variabile globale

variabileGlobale.c

```
1 #include <stdio.h>
2
3 int x = 10;
4
5 void f1(int x)
6 {
7     x = x + 10;
8     printf("\nIn functia f1 valoarea lui x este %d \n",x);
9 }
10
11 void f2(int x)
12 {
13     x = x + 20;
14     printf("In functia f2 valoarea lui x este %d \n",x);
15 }
16
17 int main()
18 {
19     f1(x);
20     x = x * 5;
21     f2(x);
22     printf("In main valoarea lui x este %d \n \n",x);
23     return 0;
24 }
```

Ce afișează programul?

In functia f1 valoarea lui x este 20
In functia f2 valoarea lui x este 70
In main valoarea lui x este 50

La apelul lui f1 și f2 se realizează o copie locală a lui x. După ieșirea din f1, copia se distrugе. Întrucât f1 nu întoarce nicio valoare, x rămâne cu aceeași valoare înainte de apelarea lui f1.

Variabile globale

variabileGlobale.c

```
1 #include <stdio.h>
2
3 int x = 10;
4
5 void f1(int x)
6 {
7     x = x + 10;
8     printf("\nIn functia f1 valoarea lui x este %d \n",x);
9 }
10
11 void f2(int x)
12 {
13     x = x + 20;
14     printf("In functia f2 valoarea lui x este %d \n",x);
15 }
16
17 int main()
18 {
19     f1(x);
20     int x = 5;
21     f2(x);
22     printf("In main valoarea lui x este %d \n \n",x);
23     return 0;
24 }
```

Ce afișează programul?

In functia f1 valoarea lui x este 20
In functia f2 valoarea lui x este 25
In main valoarea lui x este 5

Variabila locală ia locul variabilei globale

Constante întregi

- zecimale (baza 10; prima cifră nenulă): 1234
- octale (baza 8; prima cifră 0): 01234
- hexazecimale (baza 16, prefixul 0x sau 0X): 0xFA; 0XABBA
- efectul sufixului adăugat unei constante întregi (în funcție de valoare):
 - U sau u: unsigned int sau unsigned long int -> 52u, 400000U
 - L sau l: long int -> 52L, 32000L
 - UL sau uL sau Ul sau ul unsigned long int 52uL, 400000UL

Constante întregi

```
main.c ✘
01 #include <stdio.h>
02 #include <stdlib.h>
03
04
05 int main(){
06     int x;
07     x = 123;
08     printf("%d \n",x);
09
10     x = 0123;
11     printf("%d \n",x);
12
13     x = 0xAA;
14     printf("%d \n",x);
15
16     return 0;
17 }
18
```

Ce afișează programul?

123
83
170

Process returned 0 (0x0) execution time : 0

Constante în virgulă mobilă

- ❑ compuse din semn, parte întreagă, punctul zecimal, parte fracționară, marcajul pentru exponent (e sau E)
- ❑ partea întreagă sau fracționară pot lipsi (dar nu ambele)
- ❑ punctul zecimal sau marcajul exponențial pot lipsi (dar nu ambele)
- ❑ **format aritmetic**: 3.1415
- ❑ **format exponențial**: 31415E-4,6.023E+23
- ❑ implicit constantele în virgulă mobilă sunt **stocate ca double**

Constante caracter

- au ca valoarea codul ASCII al caracterelor pe care le reprezintă
- caractere imprimabile: caractere grafice (coduri ASCII între 33 și 126) + spațiu (cod ASCII = 32)
- o constantă caracter corespunzătoare unui caracter imprimabil se reprezintă prin caracterul respectiv inclus între caractere apostrof: ‘a’ (codul ASCII 97), ‘A’ (codul ASCII 65)
- cum se reprezintă caracterul apostrof?
 - apostrof = ‘\’;
- cum se reprezintă caracterul backslash?
 - backslash = ‘\\’;

Constante caracter

- au ca valoarea codul ASCII al caracterelor pe care le reprezintă
- caractere imprimabile: caractere grafice (coduri ASCII între 33 și 126) + spațiu (cod ASCII = 32)

- sevenete speciale (escape – de evitare a situațiilor care ar parea ambigue)

\a	BELL	generator de sunet
\b	BS	backspace
\f	FF	form feed
\n	LF	line feed
\r	CR	carriage return
\t	HT	Horizontal TAB
\v	VT	Vertical TAB
\\\	\	backslash
\'	'	apostrof
\"	"	ghilimele
\?	?	semnul ?
'\0'..'\0377'		orice caracter ASCII specificat OCTAL
'\0x0'..'\0xFF'		orice caracter ASCII specificat HEXAZECIMAL

Identifieri

- fiecare constantă și variabilă trebuie să aibă un nume unic
- reguli:
 - sunt permise doar literele alfabetului, cifrele și _(underscore)
 - identifierul nu poate începe cu o cifră
 - nu putem declara variabile având numele: 2win, etc.
 - literele mari sunt tratate diferit de literele mici
 - Maxim, maxim, maXim și MaxiM ar desemna variabile diferite
 - numele nu poate fi cuvânt cheie al limbajului C
 - nu putem declara variabile având numele for, while, exit, etc.

Cuprinsul cursului de azi

1. Tipuri de date fundamentale
2. Variabile și constante
3. Expresii și operatori
4. Conversii

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 operand;

□ operatori

- elemente fundamentale ale expresiilor
- operatori aritmetici, relaționali, etc.
- C are foarte mulți operatori (46 în tabelul de la sfârșit)

□ operanzi

- variabilă, o constantă, etc.
- apel de funcție, identificatorul unui tip de date, etc.
- expresie între paranteze

Expresii aritmetice și operatori aritmetici

Se aplică asupra unui singur operand

Unari		+ (plus unar) - (minus unar)
	<i>Aditivi</i>	+ (adunare) - (scădere)
Binari		* (înmulțire) / (împărțire) % (restul împărțirii)
	<i>Multiplicativi</i>	

Necesită doi operanzi

Expresii aritmetice și operatori aritmetici

□ exemple

```
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
```

- operatorii aritmetici se pot 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

Expresii aritmetice și operatori aritmetici

□ observații:

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

```
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
```

- împărțirea la zero !!
 - operatorii / și % nu pot avea operandul din dreapta 0
- trunchierea la împărțirea întreagă
 - C89 – dependent de implementare
 - C99 – trunchiere către 0

```
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)
```

Cursul 2: Căutarea unei valori într-un vector ordonat

```
int cautareBinara(int v[], int n, int val)
{
    int stanga = 0, dreapta = n - 1;
    int mijloc = (stanga + dreapta) / 2;

    while (stanga <= dreapta && val != v[mijloc])
    {
        if (val < v[mijloc])
            dreapta = mijloc - 1;
        else
            stanga = mijloc + 1;

        mijloc = floor((stanga + dreapta) / 2);
    }
    if (v[mijloc] == val)
        return mijloc;
    else
        return -1; // valoare nu se afla in vector
}
```

- ❑ căutarea binara – complexitate $O(\log_2(n))$
- ❑ este corect codul scris?
- ❑ $\text{int mijloc} = (\text{stanga} + \text{dreapta})/2;$ //posibil overflow
- ❑ $\text{int mijloc} = \text{stanga} + (\text{dreapta}-\text{stanga})/2;$ //e ok asa

Evaluarea expresiilor

- introducem **principii fundamentale** pentru evaluarea oricăror expresii prin intermediul expresiilor aritmetice
 - mai ușor de înțeles astfel
- **precedență 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

Prioritate crescută	+ (plus unar) - (minus unar)
	* (înmulțire) / (împărțire) % (restul împărțirii)
Prioritate scăzută	+ (adunare) - (scădere)

□ exemple:

$$\begin{array}{l} a + b * c \\ -a * b - c \\ +a - b / c \end{array}$$

este echivalent cu
este echivalent cu
este echivalent cu

$$\begin{array}{l} a + (b * c) \\ ((-a) * b) - c \\ (+a) - (b / c) \end{array}$$

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 (+, -, *, /, %)

$a + b - c$	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 (+, -)

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

Operatori

1. Operatori aritmetici
2. Operatorul de atribuire
3. Operatori de incrementare și decrementare
4. Operatori de egalitate, logici și relaționali
5. Operatori pe biți
6. Alți operatori:
 - de acces la elemente unui tablou, de apel de funcție, de adresa,
 - de referențiere, sizeof, de conversie explicită, condițional,
 - virgulă

Operatori de atribuire

□ operatorul de atribuire simplă =

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

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

□ valoarea unei atribuirii **var = expresie** este valoarea lui var după asignare

- expresia de atribuire poate apărea ca operand într-o altă expresie unde se așteaptă o valoare de tipul var

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

- expresia devine greu de înțeles și poate introduce erori greu de depistat

Operatori de atribuire

- atribuirea formalizată: **expr1 = expr2**
 - expr1 este *lvalue* (valoare stânga)
 - trebuie să permită stocarea valorii lui expr2 în memorie
 - corect: $v[i+1] = 10$
 - incorrect: $10 = v[i+1]$
- dacă tipul lui **expr1** și **expr2** nu este același, atunci se aplică **regula conversiei implice**
 - valoarea lui **expr2** este convertită la tipul lui **expr1** în momentul asignării

```
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 înlántui

$$a = b = c = 0$$

- ❑ operatori de atribuire compuși (operator =)
 - ❑ exemplu : `+=`, `-+`, `*=`, `/=`, `%=`, și-md. (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

Operatori de incrementare și decrementare

- operatorii **++** și **-**
 - incrementarea/decrementarea unei variabile cu 1
- forma **prefixă** (**++i** sau **--i**)
 - preincrementare/predecrementare
- forma **postfixă** (**i++** sau **i--**)
 - postincrementare/postdecrementare
- efect secundar: modificarea valorii operandului
- valoarea returnată
 - preincrementarea (**++a**) returnează valoarea **a+1**
 - postincrementarea (**a++**) returnează valoarea **a**

i++;

Exemplu echivalent:

i = i + 1;

i += 1;

Operatori de incrementare și decrementare

```
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
```

- operatorii de **preincrementare** și **predecrementare** au aceeași prioritate ca și operatorii unari + și - și sunt asociativi dreapta
- operatorii de **postincrementare** și **postdecrementare** au prioritate crescută în raport cu operatorii unari + și - și sunt asociativi stânga

Expresii logice

- ❑ expresiile logice se evaluatează la valori de tip *adevărat* sau *fals*
- ❑ sunt construite cu ajutorul a trei categorii de operatori
 - ❑ operatori **relaționali**
 - ❑ operatori de **egalitate**
 - ❑ operatori **logici**
- ❑ limbajul C tratează valorile *adevărat* și *fals* ca valori întregi
 - ❑ 0 înseamnă fals
 - ❑ orice altă valoare nenulă se interpretează ca adevărat

Operatori relaționali

- ❑ operatorii `<`, `>`, `<=`, `>=`
- ❑ rezultatul este o valoare logică, adică valoarea 0 (fals) sau 1 (adevărat)
- ❑ sunt mai puțin prioritari decât operatorii aritmetici și sunt asociativi stânga

```
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
- **==** 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

- limbajul C furnizează 3 operatori logici

- ! - operatorul unar, negare logică

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

- && - operator binar, **ȘI** logic

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

- || - operator binar, **SAU** logic

```
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 din evaluarea expresiei din stânga, atunci expresia din dreapta nu se mai evaluează

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

- operatorul ! (negare logică) 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**, operator binar
 - **| SAU pe biți**, operator binar
 - **^ SAU EXCLUSIV pe biți**, operator binar
 - **~ complement față de 1**, operator unar
 - operatori de **deplasare pe biți**
 - **<< deplasare stânga pe biți**, operator binar
 - **>> deplasare dreapta pe biți**, operator binar
- operanzi de tip întreg (nu merg pe float, double)
- ordinea de precedență - în cadrul acestei categorii

Prioritate crescută	\sim (complement față de unu)
	$<<$ (deplasare stânga)
	$>>$ (deplasare dreapta)
	$\&$ (și pe biți)
	$^$ (sau exclusiv pe biți)
Prioritate scăzută	$ $ (sau pe biți)

Operatori pe biți

- $\&$ seamănă cu $\&\&$
- $|$ seamănă cu $||$
- au un rol similar, dar la nivelul fiecărei perechi de biți de pe poziții identice
- \sim este echivalentul operației $!$ 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
$\sim a$	0	1	0	1	1 unde bitul a fost 0 și 0 unde bitul a fost 1
$\sim b$	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 \Leftrightarrow înmulțire cu 2 la puterea deplasamentului
- deplasarea spre dreapta \Leftrightarrow 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 []

```
int a[100];  
a[5] = 10;
```
- ❑ operatorul de apel de funcție (): b = f(a);
- ❑ operatorul **adresă** & și operatorul de **dereferețiere** *
 - ❑ strâns legat de pointeri (cursurile următoare)

```
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  
// ocupări 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

```
int i,n,s;  
  
printf("n=");scanf("%d",&n);  
  
for(i=1,s=0;i<=n;s=s+i,i=i+1);
```

Ordinea de precedență și asociativitate

Operator	Description	Associativity
()	Parentheses (function call) (see Note 1)	left-to-right
[]	Brackets (array subscript)	
.	Member selection via object name	
->	Member selection via pointer	
++ --	Postfix increment/decrement (see Note 2)	
++ --	Prefix increment/decrement	right-to-left
+ -	Unary plus/minus	
! ~	Logical negation/bitwise complement	
(type)	Cast (convert value to temporary value of type)	
*	Dereference	
&	Address (of operand)	
sizeof	Determine size in bytes on this implementation	
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^= =	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	
,	Comma (separate expressions)	left-to-right