



# PROGRAMARE PROCEDURALĂ

Bogdan Alexe

[bogdan.alexe@fmi.unibuc.ro](mailto:bogdan.alexe@fmi.unibuc.ro)

Secția Informatică, anul I,

2017-2018

Cursul 2

# Evaluarea la examenul din iarnă

$$Nota = \min(10, Curs + Laborator + Seminar)$$

6p                  4p                  1p

Seminar: nota pe activitate la seminar

Laborator: 2 teste de laborator (4-8 decembrie + 8-12 ianuarie)

- note intre 1 si 10
- trebuie să obțineți media minim 5 (echivalent cu 2 puncte din nota finală) pentru a putea intra în examenul final
- dacă nu puteți da un test la timp, vă rog să mă contactați din timp (dacă e posibil - cu cel puțin o săptămână înainte de test)

Curs: examen final pe calculator (orientat spre un mic proiect)

- model la penultimul curs (11 ianuarie)
- trebuie să obțineți minim nota 5 (echivalent cu 3 puncte din nota finală) pentru a promova examenul
- rotunjirea notei finale - în funcție de activitatea de laborator
- notele mai mici de 5 (spre exemplu 4.85) se rotunjesc în jos!

# Evaluarea la restanțe (iunie, septembrie)

$$Nota = \min(10, Curs + Laborator + Seminar)$$

6p                  4p                  1p

Seminar: se păstreaza nota de la activitatea la seminar

Laborator: test de laborator pe calculator

- note intre 1 si 10
- trebuie să obțineți minim 5 (echivalent cu 2 puncte din nota finală) pentru a promova examenul (condiție necesară)

Curs: al doilea test pe calculator (orientat spre un mic proiect)

- trebuie să obțineți minim nota 5 (echivalent cu 3 puncte din nota finală) pentru a promova examenul (condiție necesară)
- rotunjirea notei finale - în funcție de activitatea de laborator
- notele mai mici de 5 (spre exemplu 4.85) se rotunjesc în jos!
- cele două teste (laborator + curs) se dau în aceeași zi

# Evaluarea pentru 2018-2019

Refaceti seminarul + laboratorul.

Va conformati eventualelor modificari din anul universitar urmator!

# Recapitulare – cursul trecut

1. Algoritmi.

2. Limbaje de programare.

3. Introducere în limbajul C.

# Programa cursului

## Introducere

- Algoritmi
- Limbaje de programare.

## Fundamentele limbajului C



- Introducere în limbajul C. Structura unui program C.
- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Tipuri derivate de date: tablouri, siruri de caractere, structuri, uniuni, câmpuri de biți, enumerări, pointeri
- Instrucțiuni de control
- Directive de preprocesare. Macrodefiniții.
- Funcții de citire/scriere.
- Etapele realizării unui program C.

## Fișiere text

- Funcții specifice de manipulare.

## Funcții (1)

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

## Tablouri și pointeri

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

## Siruri de caractere

- Funcții specifice de manipulare.

## 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. Introducere în limbajul C. Structura unui program C
2. Tipuri de date fundamentale
3. Variabile și constante
4. Expresii și operatori

# Limbajul C

- ❑ popular, rapid și independent de platformă
- ❑ este un limbaj utilizat cel mai adesea pentru scrierea programelor eficiente și portabile: sisteme de operare, aplicații embedded, compilatoare, interpretoare, etc.
- ❑ limbajul C a fost dezvoltat la începutul anilor 1970 în cadrul Bell Laboratories de către Dennis Ritchie
  - ❑ strâns legat de sistemele de operare UNIX
- ❑ stă la baza pentru majoritatea limbajelor "moderne": C++, Java, C#, Javascript, Objective-C, etc.

# Limbajul C

- ❑ trei **standarde oficiale active ale limbajului**
  - ❑ **C89** (C90) – aprobat în 1989 de ANSI (American National Standards Institute) și în 1990 de către ISO (International Organization for Standardization)
    - ❑ C89 a eliminat multe din incertitudinile legate de sintaxa limbajului
    - ❑ cele mai multe compilatoare de C sunt compatibile cu acest standard (ANSI C)
  - ❑ **C99** – standard aprobat în 1999, care include corecturile aduse C89 dar și o serie de caracteristici proprii care în unele compilatoare apăreau ca extensii ale C89 până atunci
  - ❑ **C11** – standard aprobat în 2011 și care rezolvă erorile apărute în standardul C99 și introduce noi elemente, însă suportul pentru C11 este limitat, majoritatea compilatoarelor nu s-au adaptat încă la acest standard

# Caracteristici ale limbajului C

- ❑ limbaj procedural, structurat, compilat, de nivel de mijloc, scurt
- ❑ limbaj procedural, structurat
  - ❑ instrucțiuni specificate sub forma unor comenzi grupate într-o ierarhie de subprograme (denumite funcții) și care pot forma module
- ❑ limbaj compilat
  - ❑ compilatorul transformă instrucțiunile în C din fișierul sursă în limbaj mașină
- ❑ limbaj de nivel de mijloc
  - ❑ permite accesul la date și comenzi aflate aproape de nivelul fizic folosind o sintaxă specifică limbajelor de nivel înalt
- ❑ limbaj scurt
  - ❑ număr redus de cuvinte cheie
  - ❑ multe funcționalități nu sunt incluse în limbajul de bază ci necesită includerea unor biblioteci standard

# Caracteristici ale limbajului C

- limbaj eficient, portabil, permisiv**, poate fi **dificil de înțeles**
- limbaj eficient**
  - viteză mare de execuție a programelor, destinat și aplicațiilor implementate în limbaj de asamblare
  - reutilizarea ulterioară a subprogramelor
- limbaj portabil**
  - limbaj independent de hardware
- limbaj permisiv**
  - impune puține constrângeri, dă credit programatorului
  - permite introducerea unor erori care sunt foarte greu de depistat
- limbaj dificil de înțeles**
  - un stil de programare adecvat este foarte important
  - obfuscated C code contest: [www.ioccc.org](http://www.ioccc.org)

# Cuvinte cheie

## C89 = ANSI C : 32 de cuvinte cheie

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

## C99: ANSI C + alte 5 cuvinte cheie

\_Bool \_Complex \_Imaginary inline restrict

# Structura generală a unui program C

- modul principal (funcția main)
- zero, unul sau mai multe module (funcții/proceduri) care comunică între ele și/sau cu modulul principal prin intermediul parametrilor și/sau a unor variabile globale
- unitatea de program cea mai mică și care conține cod este funcția/procedura și conține:
  - partea de declarații/definiții;
  - partea imperativă (comenzile care se vor executa);

# Primul program în C

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Primul program scris in C\n");
6     return 0;
7 }
```

# Primul program în C

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Primul program scris in C\n");
6     return 0;
7 }
```

Directivă de preprocesare pentru includerea bibliotecii standard de i/o

Antetul funcției principale

Funcția principală

Corpul funcției

## Observații:

- `main` nu este cuvânt cheie în limbajul C, îl utilizăm pentru numirea funcției principale;
- `printf` nu este cuvânt cheie, este funcție de bibliotecă (print (afişare) +f (format));
- C este case sensitive, se face diferență între litere mici și mari;
- toate cuvintele cheie se scriu cu litere mici;
- instrucțiunile se termină cu `caracterul ;` (punct și virgulă);
- mai multe instrucțiuni pot fi scrise pe aceeași linie;
- spațiile ajută la organizarea codului.

# Structura unui program C simplu

*directive de procesare*

```
int main()
{
    instrucțiuni
}
```

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Primul program scris in C\n");
6 }
7 }
```

- Directive de procesare
  - directive de definiție: #define N 10
  - directive de includere a bibliotecilor: #include <stdio.h>
  - directive de compilare condiționată: #if, #ifdef, ...
  - alte directive (vorbim în cursurile următoare)
- Funcții
  - grupări de instrucțiuni sub un nume;
  - returnează o valoare sau se rezumă la efectul produs;
  - funcții scrise de programator vs. funcții furnizate de biblioteci;
  - programul poate conține mai multe funcții;
    - **main** este obligatoriu;
  - antetul și corpul funcției.

# Structura unui program C simplu

*directive de preprocessare*

```
int main()
{
    instrucțiuni
}
```

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Primul program scris in C\n");
6 }
7 return 0;
```

## □ Instrucțiuni

- formează corpul funcțiilor
  - exprimate sub formă de comenzi
- 5 tipuri de instrucțiuni:
  - instrucțiunea declarație;
  - instrucțiunea atribuire;
  - instrucțiunea apel de funcție;
  - instrucțiuni de control;
  - instrucțiunea vidă;
- toate instrucțiunile (cu excepția celor compuse) se termină cu caracterul ";"
  - caracterul ; nu are doar rol de separator de instrucțiuni ci instrucțiunile încorporează caracterul ; ca ultim caracter
  - omiterea caracterului ; reprezintă eroare de sintaxă

# Structura unui program C complex

*comentarii*

*directive de preprocessare*

*declarații și definiții globale*

```
int main()
```

```
{
```

*declarații și definiții locale*

*instrucțiuni*

```
}
```

# Cuprinsul cursului de azi

1. Introducere în limbajul C. Structura unui program C
2. Tipuri de date fundamentale
3. Variabile și constante
4. Expresii și operatori

# Tipuri de date fundamentale

- ❑ programele manipulează date sub formă de numere, litere, cuvinte, etc.
- ❑ tipul de date specifică:
  - ❑ natura datelor care pot fi stocate în variabilele de acel tip
  - ❑ necesarul de memorie
  - ❑ operațiile permise asupra acestor variabile
- ❑ În C89, limbajul C are **cinci categorii fundamentale** de tipuri de date: **int, char, double, float, void**
- ❑ C99 a introdus alte 3 tipurile de date:
  - ❑ **\_Bool** (true, false), de fapt valori întregi (0 = fals, altceva = adevărat)
  - ❑ **\_Complex** pentru numere complexe
  - ❑ **\_Imaginary** pentru numere imaginare

# Tipuri de date fundamentale

- În C89, limbajul C are **cinci categorii fundamentale** de tipuri de date: **int**, **char**, **double**, **float**, **void**
  - tipul **întreg** – **int**: variabilele de acest tip pot reține valori întregi ca 2, 0, -532
  - tipul **caracter** – **char**: variabilele de acest tip pot reține codul ASCII al unui caracter (număr întreg) sau numere întregi mici
  - tipul **real** (numere în virgulă mobilă) – **simplă precizie** – **float**: variabilele de acest tip pot reține numere care conțin parte fraționară: 4971.185, -0.72561, 2.000, 3.14
  - tipul **real** (numere în virgulă mobilă) **în dublă precizie** – **double**: variabilele de acest tip pot reține valori reale în virgulă mobilă cu o precizie mai mare decât tipul float
  - tipul **void**: indică lipsa unui tip anume

# Tipuri de date fundamentale

- se pot crea noi tipuri de date prin combinarea celor de bază
- reprezentarea și spațiul ocupat în memorie de diferitele tipuri de date depind de:
  - platformă, sistem de operare și compilator
- limitele specifice unui sistem de calcul pot fi aflate din fișierele header `limits.h` și `float.h`
  - exemplu: `CHAR_MAX`, `INT_MAX`, `INT_MIN`, `FLT_MAX`, `DBL_MAX`
- pentru determinarea numărului de octeți ocupați de un anumit tip de date se folosește operatorul `sizeof`
  - `1 octet = 1 byte = 8 biți`

# Spațiul ocupat în memorie

dimensiuneOcteti.c

```
1 #include <stdio.h>
2 #include <limits.h>
3 #include <float.h>
4
5 int main()
6 {
7     //tipul char
8     printf("\nsizeof(char) = %d \n", sizeof(char));
9     printf("valoarea minima pt o variabila de tip char este %d \n", CHAR_MIN);
10    printf("valoarea maxima pt o variabila de tip char este %d \n\n", CHAR_MAX);
11
12    //tipul int
13    printf("sizeof(int) = %d \n", sizeof(int));
14    printf("valoarea minima pt o variabila de tip int este %d \n", INT_MIN);
15    printf("valoarea maxima pt o variabila de tip int este %d \n\n", INT_MAX);
16
17    //tipul float
18    printf("sizeof(float) = %d \n", sizeof(float));
19    printf("valoarea minima > 0 pt o variabila de tip float este %E \n", FLT_MIN);
20    printf("valoarea maxima pt o variabila de tip float este %E \n", FLT_MAX);
21    printf("valoarea maxima pt o variabila de tip float este %lf \n", FLT_MAX);
22    printf("Precizia folosita pentru variabile de tip float este de %d zecimale\n\n\n", FLT_DIG);
23
24    //tipul double
25    printf("sizeof(double) = %d \n", sizeof(double));
26    printf("valoarea minima > 0 pt o variabila de tip double este %E \n", DBL_MIN);
27    printf("valoarea maxima pt o variabila de tip double este %E \n", DBL_MAX);
28    printf("valoarea maxima pt o variabila de tip double este %lf \n", DBL_MAX);
29    printf("Precizia folosita pentru variabile de tip double este de %d zecimale\n\n\n", DBL_DIG);
30
31    return 0;
32 }
```

# Spațiul ocupat în memorie

```
sizeof(char) = 1
valoarea minima pt o variabila de tip char este -128
valoarea maxima pt o variabila de tip char este 127

sizeof(int) = 4
valoarea minima pt o variabila de tip int este -2147483648
valoarea maxima pt o variabila de tip int este 2147483647

sizeof(float) = 4
valoarea minima > 0 pt o variabila de tip float este 1.175494E-38
valoarea maxima pt o variabila de tip float este 3.402823E+38
valoarea maxima pt o variabila de tip float este 340282346638528859811704183484516925440.000000
Precizia folosita pentru variabile de tip float este de 6 zecimale

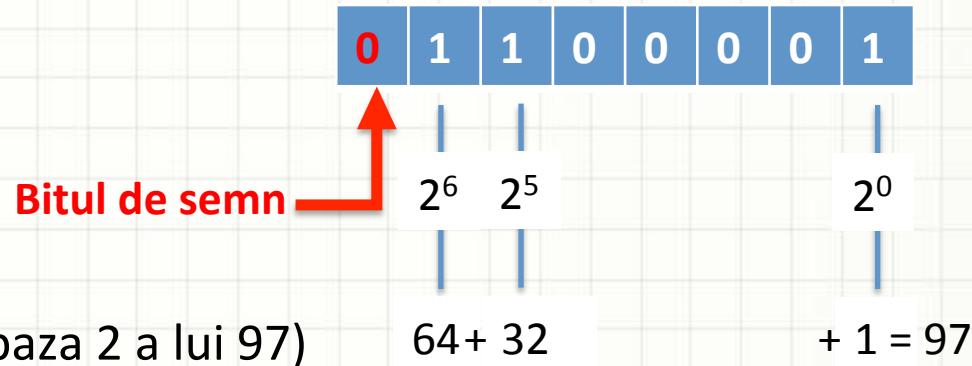
sizeof(double) = 8
valoarea minima > 0 pt o variabila de tip double este 2.225074E-308
valoarea maxima pt o variabila de tip double este 1.797693E+308
valoarea maxima pt o variabila de tip double este 1797693134862315708145274237317043567980705675
258449965989174768031572607800285387605895586327668781715404589535143824642343213268894641827684
675467035375169860499105765512820762454900903893289440758685084551339423045832369032229481658085
59332123348274797826204144723168738177180919299881250404026184124858368.000000
Precizia folosita pentru variabile de tip double este de 15 zecimale
```

# 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



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

$$64 + 32$$

$$+ 1 = 97$$

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

`int i = 190;`



Reprezentarea binara a lui 190 in memoria calculatorului

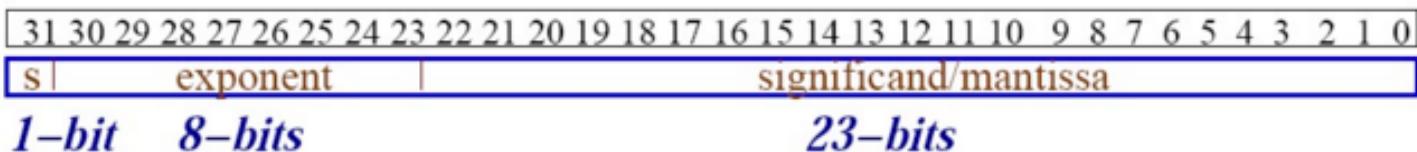
$$190 = 2^7 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 \text{ (scrierea în baza 2 a lui 190)}$$

# Reprezentarea în memorie



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

- 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

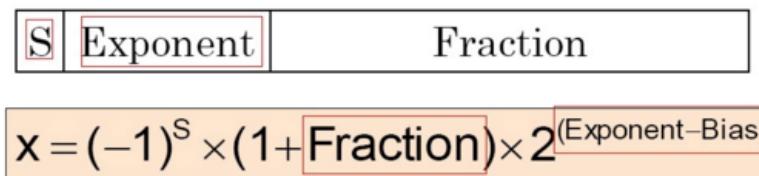
- multimea numerelor reale care pot fi reprezentate de variabile de tip float nu este repartizată uniform
- aproape jumătate din ele sunt în intervalul [-1, 1]

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

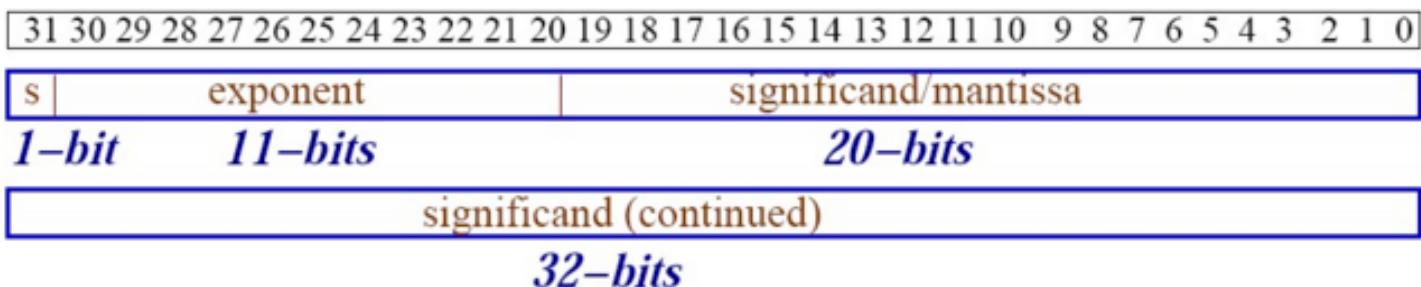
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	exponent																									significand/mantissa					
	1-bit										8-bits										23-bits										

- pentru  $s = 1$  și  $1 \leq \text{exponent} < 127$  obținem un număr pozitiv subunitar. Există  $126 * 2^{23}$  asemenea numere reprezentabile de variabile de tip float. Există  $2^{32}$  numere reale reprezentabile de tip float.
- $126 * 2^{23} / 2^{32} = 126 / 512 \approx 24,6\%$  nr pozitive subunitare
- la fel pentru  $s = -1$
- 49,2% din numere reprezentate de float sunt în [-1, 1]

# Reprezentarea în memorie



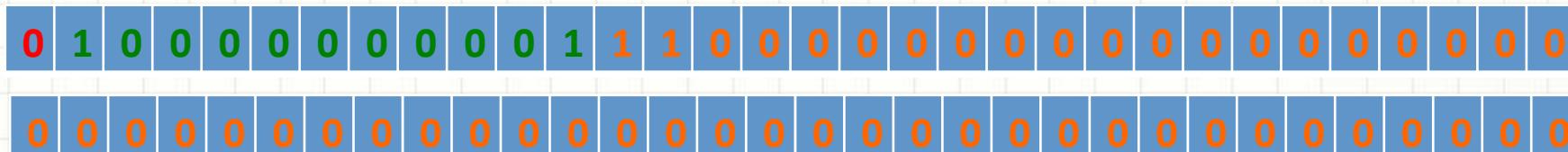
- double**: ocupă 8 octeți = 64 biți, precizie dublă, bias = 1023



**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**
  - modificatorul 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;
```

## Reprezentarea binara a lui 190 in memoria calculatorului

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

- cum se reprezintă -190 în memoria calculatorului?

**1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **0** | **1** | **0** | **0** | **0** | **0** | **1** | **0**

## 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. Introducere în limbajul C. Structura unui program C
2. Tipuri de date fundamentale
3. Variabile și constante
4. Expresii și operatori

# 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
  - `&notaExamen` = 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  
Press ENTER to continue.

# 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. Introducere în limbajul C. Structura unui program C
2. Tipuri de date fundamentale
3. Variabile și constante
4. Expresii și operatori

# 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
- apel de funcție
- expresie între paranteze
- etc.

# Expresii aritmetice și operatori aritmetici

Se aplică asupra unui singur operand

<b>Unari</b>		+ (plus unar) - (minus unar)
	<i>Aditivi</i>	+ (adunare) - (scădere)
<b>Binari</b>		* (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)
```