

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



# Programarea Calculatoarelor (limbajul C)

## Curs 3 – Bazele programării în limbajul C

Prof. Bogdan IONESCU

2016-2017

# Cuprins

- 3.1. Execuția algoritmilor
- 3.2. Introducere în limbajul C
- 3.3. Bazele programării în limbajul C

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

1/52

## 3.1. Execuția algoritmilor

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

2/52

Un sistem de calcul lucrează cu reprezentări ale informației prin execuția unei anumite înșirări de operații simple:

→ **algoritm**

**Algoritm** = o metodă, un procedeu care asigură rezolvarea unei probleme complexe de calcul prin executarea unor operații punctuale.

> Foarte importantă este **înțelegerea** operațiilor și a modului în care se succed operațiile într-un algoritm, și prin urmare într-un program.

- permite localizarea *eventualelor erori*,
- algoritmul unei probleme poate fi destul de diferit de modul "natural" de implementare,

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

3/52

**P** *Enunț:* să se implementeze algoritmul lui Euclid: fiind date două numere întregi pozitive ( $m$  și  $n$ ), să se găsească cel mai mare divizor comun (cmdc).

Algoritm în limbaj natural:

"Se aleg două numere naturale.  
Se împarte primul număr la al doilea.  
Cât timp restul împărțirii este nenul, se înlocuiește primul număr cu al doilea și al doilea cu restul împărțirii.  
Se reia apoi împărțirea între cele două numere obținute.  
Rezultatul căutat este ultimul rest nenul"

$m, n \rightarrow m:n=0$  și  $r, m \leftarrow n$  și  $n \leftarrow r, m:n=0$  și  $r, r=0$  ?

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

4/52

Algoritmul într-un limbaj mai structurat:

ordine a operațiilor

*pasul 1:* citește  $m, n$ ;  
*pasul 2:* atribuie lui cmdc valoarea lui  $n$ ;  
*pasul 3:* atribuie lui  $r$  restul împărțirii  $m:n$ ;  
*pasul 4:* cât timp  $r \neq 0$ ;  
*pasul 5:* atribuie lui  $m$  valoarea lui  $n$ ;  
*pasul 6:* atribuie lui  $n$  valoarea lui  $r$ ;  
*pasul 7:* atribuie lui cmdc valoarea lui  $n$ ;  
*pasul 8:* atribuie lui  $r$  restul împărțirii  $m:n$ ;  
*pasul 9:* repetă *pasul 4*;  
*pasul 10:* scrie valoare cmdc;  
*pasul 11:* stop;

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

5/52

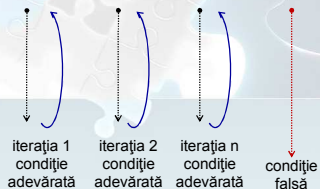
### Înțelegerea operațiilor:

Cum se calculează **restul împărțirii** ?

$$r = m - \text{int}(m / n) * n \quad \text{unde } \text{int}(m / n) = \text{parte întregă}$$

Cum se implementează **bucloa cât timp** ?

```
while (condiție)
{
    instrucțiuni;
}
```



### Verificarea modului în care se succed operațiile

Se ia un exemplu practic pentru verificarea algoritmului:

```
m = 42, n = 28
2: cmdc = 28
3: r = 14
4: r ≠ 0
5: m = 28
6: n = 14
7: cmdc = 14
8: r = 0
10: rezultat cmdc = 14
```

```
pasul 1: citește m,n;
pasul 2: atribuie lui cmcd valoarea lui n;
pasul 3: atribuie lui r restul împărțirii m:n;
pasul 4: cât timp r≠0;
pasul 5: atribuie lui m valoarea lui n;
pasul 6: atribuie lui n valoarea lui r;
pasul 7: atribuie lui cmcd valoarea lui n;
pasul 8: atribuie lui r restul împărțirii m:n;
pasul 9: repetă pasul 4;
pasul 10: scrie valoare cmcd;
pasul 11: stop;
```

Să fim pesimiști, totuși să mai încercăm un exemplu ce implică mai multe iterații, poate ne scapă ceva ...

Testul 2, alte valori:

```
m = 28, n = 42
2: cmdc = 42
3,4: r = 28, r ≠ 0
5: m = 42
6: n = 28
7: cmdc = 28
8,4: r = 14, r ≠ 0
5: m = 28
6: n = 14
7: cmdc = 14
8: r = 0
19: rezultat cmdc = 14
```

```
pasul 1: citește m,n;
pasul 2: atribuie lui cmcd valoarea lui n;
pasul 3: atribuie lui r restul împărțirii m:n;
pasul 4: cât timp r≠0;
pasul 5: atribuie lui m valoarea lui n;
pasul 6: atribuie lui n valoarea lui r;
pasul 7: atribuie lui cmcd valoarea lui n;
pasul 8: atribuie lui r restul împărțirii m:n;
pasul 9: repetă pasul 4;
pasul 10: scrie valoare cmcd;
pasul 11: stop;
```

*Detaliu:* numerele trebuie introduse în ordine "mare-mic" (optimal).

**Concluzie:** la elaborarea unui algoritm trebuie verificat modul de execuție al acestuia:

→ *inițial*, din punct de vedere al algoritmului exprimat în limbaj natural (**validarea teoretică**),

→ *ulterior*, din punct de vedere al algoritmului exprimat într-un limbaj structurat, apropiat de limbajul de programare, simulând modul de execuție al calculatorului (**validarea practică**),

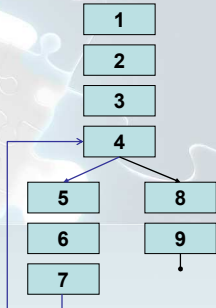
- prin
- identificare operații elementare
  - identificare operații complexe
  - identificare erori de principiu
  - identificare erori ascunse
- ↔ date de test (exemple)

### Modul de execuție al calculatorului:

**P** *Enunț:* să se implementeze algoritmul de calcul al factorialului numărului n (n!)

```
pasul 1: citește n;
pasul 2: atribuie lui i valoarea n-1;
pasul 3: atribuie lui p valoarea n;
pasul 4: cât timp i >= 1
pasul 5: atribuie lui p valoarea p*i;
pasul 6: atribuie lui i valoarea i-1;
pasul 7: repetă pasul 4
pasul 8: scrie valoare p;
pasul 9: stop;
```

Cum sunt executate instrucțiunile?



> Criterii de apreciere a calității algoritmului ?

- evident, în primul rând să **funcționeze corect**,
- să aibă un **număr redus de pași** (să fie eficient),

criterii oarecum subiective

- să aibă o **complexitate de calcul redusă**.

criteriu științific, anticipează necesarul de resurse de calcul

**Complexitate de calcul** = numărul de operații de bază, elementare, efectuate de algoritmul.

> Operație elementară ?

- o adunare, scădere, ...
- o comandă, apelarea unei funcții simple, ...

n operații elementare  
**O(n)**

*Exemplu (factorial):*

1: citește n;	1 : O(1)
2: atribuie lui i valoarea n-1;	2 : O(1)
3: atribuie lui p valoarea n;	3 : O(1)
4: cât timp i>=1;	4,5,6,7: O(2*(n-1))
5: atribuie lui p valoarea p*i;	8 : O(1)
6: atribuie lui i valoarea i-1;	
7: repetă pasul 4	$O(1+1+1+2*(n-1)+1)$
8: scrie valoarea p;	<b>→O(2*(n+1))</b>
9: stop;	

> Este totuși un calcul aproximativ, de ce ?


Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 12/52

## 3.2. Introducere în limbajul C

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 13/52

### Scurt istoric al limbajului C:

- **1969:** începe dezvoltarea sistemului de operare Unix în limbajul de asamblare al calculatorului DEC PDP-7:
  - limbaj greoi, număr mare de erori
- **1970:** pe baza limbajului de asamblare **A** (assembler) este dezvoltat un nou limbaj **B** (Bell Labs):
  - un singur tip de date: word.
- **1971:** Dennis Ritchie dezvoltă un limbaj complet nou numit **C**,
- **1989-90:** ANSI (American National Standards Institute) și ISO (International Standards Organization) definesc standardul ANSI-C.



Digital Equipment Corporation PDP-7

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 14/52

### Principalele caracteristici ale limbajului C

- > dimensiuni reduse ale pachetului de programe de dezvoltare (**eficient**),
- > utilizare extensivă a apelurilor de funcții (**modularitate**),
- > control relaxat al tipurilor de date,
- > limbaj de programare structurat,
- > permite și programarea de nivel scăzut (limbaj de asamblare - assembler),
- > permite utilizarea pointerilor pentru manipularea memoriei, tablourilor, structurilor și funcțiilor (**execuție rapidă**)

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 15/52

### Principalele caracteristici ale limbajului C (continuare)

- > permite alocarea dinamică a memoriei (**folosire eficientă a memoriei**), ???

<code>int V[100];</code>	→	<b>alocare statică:</b> se alocă în memorie 100x32biți.
<code>int *V, size; ... scanf("%d",&amp;size); V=(int *)malloc(size);</code>	→	<b>alocare dinamică:</b> se alocă memorie mai întâi pentru adresa lui V, și apoi în funcție de necesitate. → se poate elibera memoria.

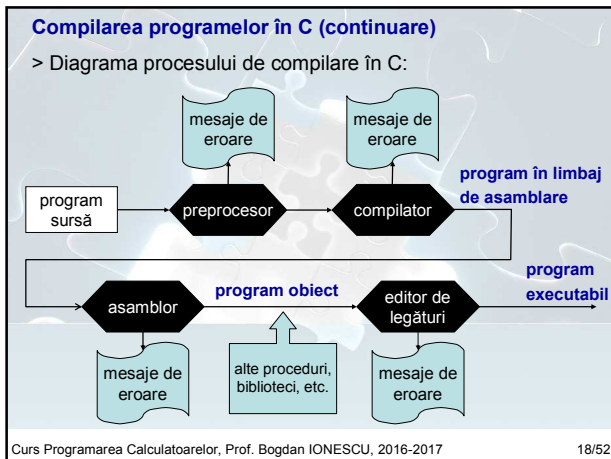
> programele scrise în C pot fi compilate pe o varietate mare de sisteme de calcul (**portabilitate**),

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 16/52

### Compilarea programelor în C

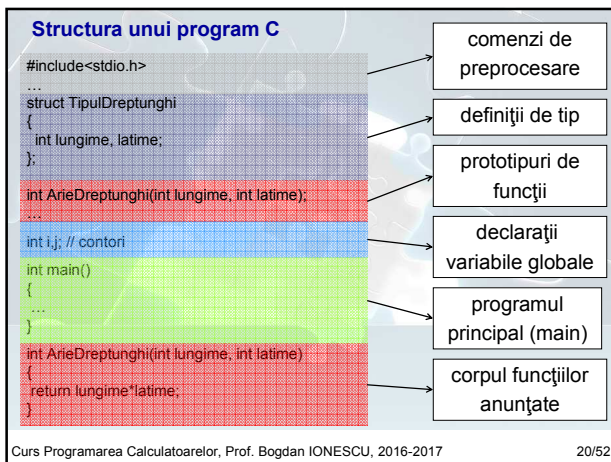
- > Modelul compilării în C diferă un pic față de cel prezentat pentru cazul general în Cursul 2.
- > Compilatorul este împărțit în trei programe:
  - **preprocesor:** responsabil pentru înlăturarea comentariilor, interpretarea *directivelor de preprocesare* (ex.: #include),
  - **compilatorul:** traduce sursa C primită de la preprocesor în limbaj de asamblare,
  - **asamblorul:** crează fișierele obiect,

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 17/52



## 3.3. Bazele programării în limbajul C

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 19/52



### Structura unui program C (continuare)

> Semnificația componentelor structurale ale programului este următoarea:

- comenzile de preprocesare:** specificate de caracterul #, sunt în general folosite pentru a facilita schimbarea și compilarea programelor în diferite medii de execuție.
  - informează preprocesorul să realizeze anumite acțiuni: să înlocuiască text, să insereze conținutul altor fișiere în fișierul sursă, să evite compilarea unei anumite porțiuni din fișier, etc.

De exemplu: **#include** <nume\_librarie.h>  
**#define** lungime 80

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 21/52

### Structura unui program C (continuare)

- comenzile de preprocesare (continuare)**

**#include** <nume\_fisier.h>

- include conținutul fișierului desemnat (nume\_fisier.h) în program. Acesta este numit și fișier antet (*header* - .h)
- fișierele antet conțin funcții definite de utilizator sau biblioteci externe de funcții puse la dispoziția utilizatorului de către limbaj sau dezvoltate de alți programatori.
- prin includere acestora, funcțiile din fișierul .h devin disponibile utilizatorului în momentul programării.

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 22/52

### Structura unui program C (continuare)

- comenzile de preprocesare (continuare)**

> Bibliotecile de funcții uzuale disponibile în C:

- math.h** { conține *funcții matematice*, de exemplu: sqrt, pow, log, sin, fabs, fmod, etc.
- stdio.h** { conține *funcții de lucru cu dispozitivele de intrare și ieșire*, de exemplu: citire date de la tastatură (scanf), scriere date pe ecran (printf), lucrul cu fișiere (fopen, fscanf, fprintf, fclose), etc.
- string.h** { conține *funcții de lucru cu șiruri de caractere*, de exemplu: concatenare, copiere, căutare, etc.

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 23/52



### Structura unui program C (continuare)

#### • comenzile de preprocesare (continuare)

> Bibliotecile de funcții uzuale disponibile în C (continuare):

<b>stdlib.h</b>	{	conține <i>funcții utilitare</i> , de exemplu: alocare memorie (malloc, calloc), eliberare memorie (free), terminare execuție program (exit), executare comenzi sistem de operare (system)...
<b>time.h</b>	{	conține <i>funcții de lucru cu timpul și date calendaristice</i> , de exemplu: timpul în secunde, ore, zile, luni, an, timpul procesorului, etc.
<b>limits.h</b>	{	conține <i>constante ce specifică valorile maxime ale tipurilor de date</i> , de exemplu: INT_MAX, INT_MIN, SHRT_MAX, LONG_MAX, etc.

### Structura unui program C (continuare)

#### • comenzile de preprocesare (continuare)

**#define** <nume\_constanta> <valoare>

→ definește un nume simbolic sau o constantă. În program simbolul <nume\_constanta> va fi înlocuit cu valoarea specificată de <valoare>.

Exemplu: **#define** PI 3.14

**#define** MAX 1000

```
int main(void)
{
    ...
    return MAX*3;
}
```

cuvântul "MAX" va fi înlocuit cu valoarea 1000

### Structura unui program C (continuare)

• **definiții de tip:** specificarea unor noi tipuri de date ce sunt de regulă definite pe baza tipurilor de date de bază furnizate de limbajul C.

De exemplu:

```
struct persoana
{
    int varsta;
    float greutate;
    char nume[25];
} membru_familie;

enum tipuri {MIC=10, MEDIU=50, MARE=100};
...
enum tipuri Var;
Var=MEDIU;           → Var=50
Var=MARE;           → Var=100
```

### Structura unui program C (continuare)

• **prototipuri de funcții:** anunțarea și specificarea funcțiilor ce vor fi definite și folosite în cadrul programului (*prototip = model*).

Prototipul unei funcții are următoarea formă în C:

<tip\_date\_de\_iesire> **NumeFuncție**(<lista\_date\_de\_intrare>);

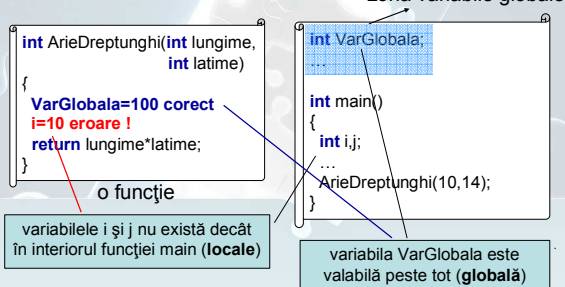
tipul valorii returnate de funcție, ex.: int, float, ... ( <b>returnează o singură valoare</b> )	identificatorul funcției, unic,	lista variabilelor de intrare folosite în cadrul funcției, ex.: (int i, float j)	“.” corpul funcției nu este specificat aici.
--	---------------------------------	--	--

Exemplu: int **NumarPrim**(int x); sau void **Punct**(int X1, int X2);

### Structura unui program C (continuare)

• **declarații și definiții globale:** definirea de variabile globale.

Ce înseamnă variabilă globală ?



### Structura unui program C (continuare)

• **corpul funcțiilor:** în această secțiune sunt detaliate funcțiile ce alcătuiesc programul.

→ funcția **main()** ce indică programul principal, aceasta nu trebuie să lipsească (obligatorie).

funcția main trebuie să returneze un integer (norma ANSI).
void = vid, funcția nu are nici un argument.
convenție: return 0 – nu sunt erori return n – se returnează de regulă codul erorii.

### Structura unui program C (continuare)

- **corpul funcțiilor** (continuare)
  - funcțiile ce au fost anunțate prin intermediul prototipurilor de funcții din secțiunea dedicată,

```

int ArieDreptunghi(int lungime, int latime);
int MultipluDe2(int x);

int ArieDreptunghi(int lungime, int latime)
{
    return lungime*latime;
}

int MultipluDe2(int x)
{
    if ((x%2)==0) return 1;
    else return 0;
}

```

prototip funcții

corpul funcțiilor

Atenție: prototipul și funcția trebuie să fie identice

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 30/52

### Structura unui program C (continuare)

> Alternativă: este posibil și următorul mod de structurare al programului C:

```

int i,j; // contori
...
int ArieDreptunghi(int lungime, int latime)
{
    return lungime*latime;
}
...
int main()
{
    ...
}

```

declarații variabile globale

nu mai folosim prototipuri, funcțiile sunt scrise integral

Ce s-a schimbat față de structura prezentată anterior: prototipuri variabile globale, corp funcții

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 31/52

### Structura unui program C (continuare)

- **comentariile** :
  - = sunt pasaje de text ce sunt înlăturate de preprocesor, cu alte cuvinte, nu sunt compilate și nici interpretate.

Atunci de ce să scriem comentarii ?

- să explicăm funcționalitatea programului,
- să explicăm funcționalitatea anumitor funcții și proceduri,
- să rezumăm anumite linii de cod complexe pentru a înțelege rapid modul de execuție fără a fi nevoie să refacem calculul,
- “de dragul” lizibilității programului.

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 32/52

### Structura unui program C (continuare)

- **comentariile** (continuare)
  - > Cum specificăm comentariile în C ?

**Varianta 1:**  
“//” → se comentează o singură linie

```
// acest text este un comentariu
// și acest text este tot un comentariu
```

**Varianta 2:**  
“/\*” și “\*/” → se comentează tot ce este cuprins între “/\*” și “\*/”

```
/* acest text este un comentariu
și acest text este tot un comentariu */
```

cazuri particulare:

```
/* comentariu // iar comentariu */ tolerat
/* text /* sa fim siguri */ text */
```

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 33/52

### Structura unui program C (continuare)

- **comentariile** : exemple

```

// deschide fisier .avi
res=AVIFileOpen(&avi, szFileName, OF_SHARE_DENY_WRITE, NULL);

// deschide primul flux video
res=AVIFileGetStream(avi, pStream, streamtypeVIDEO, 0);

// recupereaza informatii fisier, avi info
AVIFileInfo(avi, &avi_info, sizeof(AVIFILEINFO));
FrameNumber=AVIStreamLength("pStream");
...

/* creeza o imagine binara in care deplasarea unui bloc de pixeli
sau situatia de discontinuitate este marcata cu valoarea 1 */
for (y=0; y<SizeY; y++)
for (x=0; x<SizeX; x++)
if ((int)MotionAmpl[y][x]==0)
    binaryImg[y][x]=0;
else
    binaryImg[y][x]=1;

```

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 34/52

### Tipuri de date fundamentale în C

> Limbajul de programare C propune următoarele tipuri fundamentale de date:

- tipul alfanumeric
- tipuri întregi
- tipuri reale
- tipul “void”
- tipul enum

> Există și tipuri compuse, ce folosesc tipurile fundamentale, acestea vor fi însă discutate ulterior.

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 35/52

### Tipuri de date fundamentale în C (continuare)

> Tipul alfanumeric:

char

- variabile de tip caracter

- întreg mic

→ stocare date pe 8 biți

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
...
1 1 1 1 1 1 1 1
```

> Variante:

char

signed char

unsigned char

}

[-128 ; 127]

[0 ; 255]

> Este număr întreg sau caracter ?  
**este și una și alta ...**

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 36/52

### Tipuri de date fundamentale în C (continuare)

> Tipul alfanumeric (continuare)

O variabilă de tip caracter poate stoca și numere întregi deoarece *caracterele* sunt identificate de sistemul de calcul pe baza unui cod unic numit și **cod ASCII**.

**ASCII = American Standard Code for Information Interchange**

Dec	Hex	Oct	Char	Dec	Hex	Oct	Html	Chr	Dec	Hex	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	#32;	Space	64	40	100	#64;	0
1	1	001	SOH (start of heading)	33	21	041	#33;	!	65	41	101	#65;	A
2	2	002	STX (start of text)	34	22	042	#34;	"	66	42	102	#66;	B
3	3	003	ETX (end of text)	35	23	043	#35;	#	67	43	103	#67;	C
4	4	004	EOF (end of transmission)	36	24	044	#36;	^	68	44	104	#68;	D
5	5	005	ENQ (enquiry)	37	25	045	#37;	^	69	45	105	#69;	E
6	6	006	ACK (acknowledge)	38	26	046	#38;	^	70	46	106	#70;	F
7	7	007	BEL (bell)	39	27	047	#39;	^	71	47	107	#71;	G
8	8	010	BS (backspace)	40	28	050	#40;	(	72	48	110	#72;	H
									104	68	150	#104;	h

Extras din tabela de coduri ASCII

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 37/52

### Tipuri de date fundamentale în C (continuare)

> Tipul alfanumeric (continuare):

Există mai multe extensii ale standardului ASCII pentru a putea reprezenta și caractere specifice fiecărei limbi:

- **ISO 8859-16** se numește *South-Eastern European* și cuprinde limbile albaneză, croată, maghiară, poloneză, română, și slovenă, dar și franceză, italiană și gaelică (ortografie nouă).
- **ISO 8859-5** se numește *Cyrillic* și cuprinde limbile burgară, belorusă, macedoniană, rusă, sârbă (și în trecut și limba ucrainiană).
- **ISO 8859-7** se numește *Greek* și cuprinde limba greacă modernă.
- etc.

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 38/52

### Tipuri de date fundamentale în C (continuare)

> Tipul alfanumeric (continuare):

*Exemple:*

char a;

unsigned char initiala;

signed char c;

```
a=10;
initiala='f';
c=-80;
```

**Atenție:** constantele de tip caracter se specifică între apostrof ' '

Cum știe sistemul când să folosească codul ASCII sau caracterul ?

Se subînțelege din context:

a+initiala+'!' = 10+73+33 = 116

}

→ cod ASCII

→ întreg

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 39/52

### Tipuri de date fundamentale în C (continuare)

> Tipuri întregi:

int

- variabile de tip întreg

> Variante:

short int

signed short int

}

stocare date pe 16 biți

[- 32768; 32767], (short)

unsigned short int

unsigned short

}

stocare date pe 16 biți

[0; 65535],

signed int

int

}

dependent de sistem (~32 biți)

[- 2<sup>31</sup>; 2<sup>31</sup>-1], (signed)

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 40/52

### Tipuri de date fundamentale în C (continuare)

> Tipuri întregi (continuare)

> Variante:

unsigned int

}

dependent de sistem (~32 biți)

[0; 2<sup>32</sup>-1], (unsigned)

long int

signed long int

}

stocare date pe 32 biți

[- 2<sup>31</sup>; 2<sup>31</sup>-1], (long)

unsigned long int

unsigned long

}

stocare date pe 32 biți

[0; 2<sup>32</sup>-1],

*Exemple:* signed int a=-1000;

unsigned short int b= 65535;

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 41/52

### Tipuri de date fundamentale în C (continuare)

#### > Tipuri reale:

**float** - variabile de tip real în precizie simplă.

→ stocare date pe **32 biți**  
[1.175494351x10<sup>-38</sup> ; 3.402823466x10<sup>+38</sup>]

> Se observă că numerele reale sunt reprezentate în mod diferit de numerele întregi.

Reprezentarea numerelor **în virgulă mobilă**:

-1.3123437 = -13123437x10<sup>-7</sup>  
mantisă                      exponent

### Tipuri de date fundamentale în C (continuare)

#### > Tipuri reale (continuare):

**float**                      1bit                      23biți                      8biți  
sgn.                      mantisă                      exponent

**double** - variabile de tip real în precizie dublă.

→ stocare date pe **64 biți**  
[2.2250738585072014 x10<sup>-308</sup> ; 1.7976931348623158 x10<sup>+308</sup>]

1bit                      52biți                      11biți  
sgn.                      mantisă                      exponent

**long double** - identic cu double.

### Tipuri de date fundamentale în C (continuare)

#### > Tipul "void":

**void** = vid, nu este un tip propriu-zis de date, deoarece nu conține nici o valoare.

Este folosit în trei situații:

- pentru a specifica că o funcție nu returnează nici o valoare,
- pentru a specifica că o funcție nu primește nici un parametru,
- la definirea generică a *pointerilor*, aceștia putând indica astfel orice tip de variabilă, cu excepția constantelor.

Exemplu:

**void** Functie1(**int** a, **int** b);    sau    **int** Functie2(**void**);

### Tipuri de date fundamentale în C (continuare)

#### > Tipul enum:

**enum** = reprezintă un tip de date definit de utilizator ce constă într-un set de constante șiruri de caracter numite și enumeratori.

```
enum <nume_tip> { listă constante string };  
<nume_tip> var1,var2;
```

Exemple:

```
enum Figuri { Romb, Patrat, Cerc, Triunghi };  
enum Bool { True, False };
```

### Tipuri de date fundamentale în C (continuare)

#### > Tipul enum (continuare)

Exemple:

```
enum Figuri { 0 Romb, 1 Patrat, 2 Cerc, 3 Triunghi };  
...  
Figuri proba;  
...  
proba=Romb;  
printf("%d", proba);  
proba=Cerc;  
printf("%d", proba);  
...
```

variabila **proba** este de tip Figuri.  
se afișează pe ecran valoarea 0 ?  
se afișează pe ecran valoarea ???

### Tipuri de date fundamentale în C (continuare)

#### > Tipul enum (continuare)

Exemple:

```
enum Bool { 0 False, 1 True};  
...  
Bool proba;  
...  
proba=True;  
printf("%d", proba);  
proba=0;  
proba=TRue;
```

variabila **proba** este de tip Bool.  
se afișează pe ecran valoarea 1.  
~proba=False  
Eroare: limbajul C este case-sensitive, TRue != True



### Tipuri de date fundamentale în C (continuare)

> Tipul enum (continuare)

Atenție: valori întregi

Exemple:

```
enum Figuri { Romb=10, Patrat=5, Cerc=8 };
...
Figuri proba;
...
proba=Romb;
printf("%d", proba);
proba=Cerc;
printf("%d", proba);
...
```

variabila **proba** este de tip Figuri.

se afișează pe ecran valoarea 10 ?

se afișează pe ecran valoarea ???

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 48/52

### Tipuri de date fundamentale în C (continuare)

> Tipul enum (continuare)

Variabilele de tip enum sunt folosite în felul următor: peste tot în cod se înlocuiește constanta șir de caracter (enumerator) cu valoarea numerică a acesteia

- ↳ implicită de la 0 la n,
- ↳ specificată de utilizator (întreagă)

NB: limbajul C este case-sensitive, alpha!=Alpha!=ALPHA ...

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 49/52

### Tipuri de date fundamentale în C (continuare)

> Constante = un caz particular de variabilă a cărei valoare nu poate fi modificată pe parcursul execuției programului.

**const** - se specifică cuvântul cheie **const** înaintea tipului variabilei.

Exemplu:

```
int main()
{
  const int x=10;
  const char c;
  x=14;
  return 0;
}
```

x este o variabilă întreagă ce are valoarea 10.

Eroare: constantă neinițializată !

Eroare: se atribuie o valoare unei variabile ce poate fi doar citită

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 50/52

### Tipuri de date fundamentale în C (continuare)

> În limbajul C se pot defini tipuri sinonime pentru tipurile de date existente, astfel denumirea acestora poate fi înlocuită cu o denumire specificată de utilizator.

**typedef** - se specifică cuvântul cheie **typedef** urmat de tipul substituit și noua denumire.

Exemplu:

```
typedef long int intreglung;
typedef double tipreal;
...
intreglung x,y;
tipreal a,b;
```

tipul numit "intreglung" este definit ca fiind long int

tipul numit "tipreal" este definit ca fiind double

x,y sunt long int  
a,b sunt double

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 51/52

# Sfârșitul Cursului 3

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 52/52