



# PROGRAMAREA CALCULATOARELOR

Andrei Patrascu

[andrei.patrascu@fmi.unibuc.ro](mailto:andrei.patrascu@fmi.unibuc.ro)

Secția Calculatoare si Tehnologia  
Informatiei, anul I, 2018-2019

Cursul 1

# Cuprinsul cursului de azi

1. Prezentarea cursului de Programarea Calculatoarelor
2. Primul curs

# Utilitatea cursului de PC

- PP = paradigma de programare bazată pe conceptul de apel de procedură/funcție/rutină/subrutină. Un program este privit ca o mulțime ierarhică de funcții care manipulează datele.
- vom studia limbajul C = limbaj fundamental de programare (1970), exponent al programării procedurale. Alte limbaje (C++, Java, PHP, Python ) împrumută multe din caracteristicile limbajului C.

# De ce C?

<http://www.tiobe.com/tiobe-index/>

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.

Sep 2017	Sep 2016	Change	Programming Language	Ratings	Change
1	1		Java	12.687%	-5.55%
2	2		C	7.382%	-3.57%
3	3		C++	5.565%	-1.09%
4	4		C#	4.779%	-0.71%
5	5		Python	2.983%	-1.32%
6	7	▲	PHP	2.210%	-0.64%
7	6	▼	JavaScript	2.017%	-0.91%
8	9	▲	Visual Basic .NET	1.982%	-0.36%
9	10	▲	Perl	1.952%	-0.38%
10	12	▲	Ruby	1.933%	-0.03%
11	18	▲▲	R	1.816%	+0.13%
12	11	▼	Delphi/Object Pascal	1.782%	-0.39%
13	13		Swift	1.765%	-0.17%
14	17	▲	Visual Basic	1.751%	-0.01%
15	8	▼▼	Assembly language	1.639%	-0.78%
16	15	▼	MATLAB	1.630%	-0.20%
17	19	▲	Go	1.567%	-0.06%
18	14	▼▼	Objective-C	1.509%	-0.34%
19	20	▲	PL/SQL	1.484%	+0.04%
20	26	▲▲	Scratch	1.376%	+0.54%



# De ce C?


































<http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017>

*“Rankings are created by weighting and combining 12 metrics from 10 sources. We offer preset weightings—the default is our IEEE Spectrum ranking—but there are presets for those interested in what's trending or most looked for by employers.”*

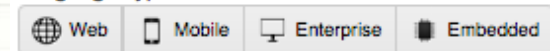
Language Types (click to hide)







































Language Rank    Types    Spectrum Ranking

1. Python	 	100.0
2. C	  	99.7
3. Java	 	99.4
4. C++	  	97.2
5. C#	 	88.6
6. R		88.1
7. JavaScript	 	85.5
8. PHP		81.4
9. Go	 	76.1
10. Swift	 	75.3
11. Arduino		73.0
12. Ruby	 	72.4
13. Assembly		72.1
14. Scala	 	68.3
15. Matlab		68.0
16. HTML		67.0
17. Shell		66.3
18. Perl	 	57.6
19. Visual Basic		55.4
20. Cuda		53.9

Language Types (click to hide)



Language Rank    Types    Jobs Ranking

1. Java	  	100.0
2. C	  	99.3
3. Python	 	99.3
4. C++	  	92.6
5. JavaScript	 	90.2
6. C#	  	86.6
7. PHP		81.0
8. HTML		79.6
9. Ruby	 	77.2
10. Swift	 	77.2
11. Assembly		76.2
12. Shell		74.7
13. Scala	 	71.5
14. R		69.6
15. Perl	 	66.4
16. SQL		64.1
17. Matlab		61.6
18. Go	 	61.2
19. Objective-C	 	56.6
20. Visual Basic		54.5

## Orar

	8	9	10	11	12	13	14	15
Lu			ProgrCalc 153 8		ProgrCalc 151 8		151/152/153/154 2(Pompeiu)	
Ma			ProgrCalc 152 8		ProgrCalc 154 8			
Mi			ProgrCalc 152 Gr 1 L.121D		ProgrCalc 153 Gr 2 L.121D		ProgrCalc 151 Gr 1 L.121D	
Jo	ProgrCalc 152 Gr 2 L.310		ProgrCalc 151 Gr 2 L.310		ProgrCalc 153 Gr 1 L.310			

- Curs – 2 ore/săptămână
- Laborator – 2 ore/săptămână (cu semi-grupa)
- Seminar – 2 ore/ 2 săptămâni (cu grupa)

# Material

- moodle.fmi.unibuc.ro

## Programarea calculatoarelor

[Acasă](#) ► [Cursurile mele](#) ► [ProgrCalc2](#)

### Navigation



#### Acasă

- [Acasă](#)
- [Site pages](#)
- [Profilul meu](#)
- ▼ [Cursurile mele](#)
  - ▼ [ProgrCalc2](#)
    - [Participanți](#)
    - [Rapoarte](#)
    - [General](#)
    - [6 November - 12 November](#)
    - [13 November - 19 November](#)
    - [20 November - 26 November](#)
    - [27 November - 3 December](#)

### Rezumat săptămână



[News forum](#) → ↕ 🔊 x2 ✕ 🌐 👤 👥

Adaugă resursă...

2 October - 8 October



Adaugă resursă...

9 October - 15 October



Adaugă resursă...

16 October - 22 October



Adaugă resursă...

# Obiectivele cursului

1. Formarea deprinderilor de **programare structurată (modularizare)** în limbaje de programare clasice și moderne (descompunerea unei probleme complexe în subprobleme relativ simple și independente);
1. Însușirea caracteristicilor **limbajului C**: alocarea memoriei, lucrul cu pointerii, lucrul cu fișierele, programarea generică. Vrem să știți să codați în C, să vă dați seama ce face un cod scris de altcineva, să depanați un cod în C;
1. Dezvoltarea unei **gândiri algoritmice + abilitate de programare** - foarte utile în rezolvarea diverselor probleme cu care vă veți întâlni în facultate sau în viața reală.



# Programa cursului

## □ Introducere

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

## □ Fundamentele limbajului C

- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Tipuri derivate de date: tablouri, șiruri 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 paramerilor. Pointeri la funcții.

## □ Tablouri și pointeri

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

## □ Șiruri de caractere

- Funcții specifice de manipulare.

## □ Fișiere binare

- Funcții specifice de manipulare.

## □ Structuri de date complexe și autoreferite

- Definiție și utilizare

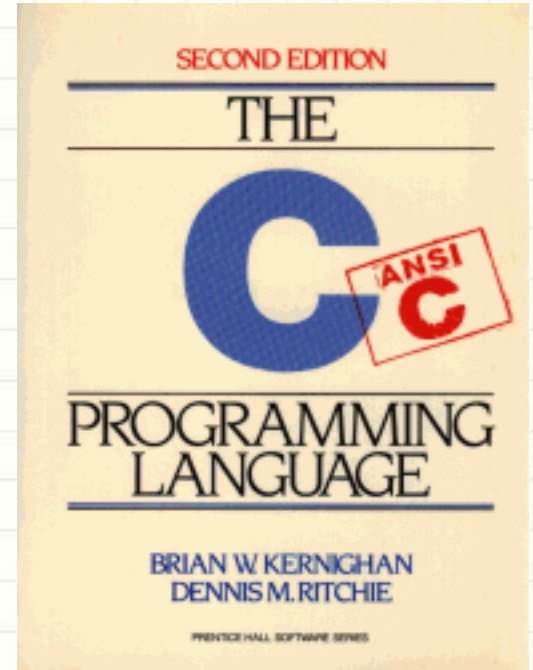
## □ Funcții (2)

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

# Bibliografie

1. Kernighan & Ritchie: The C programming language

<http://zanasi.chem.unisa.it/download/C.pdf>



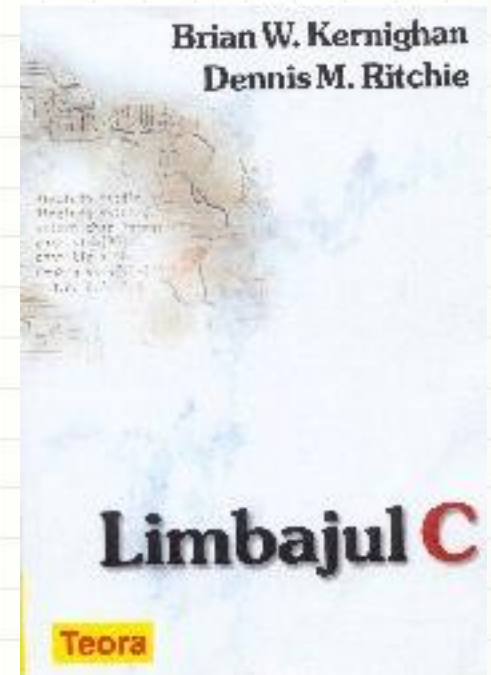
# Bibliografie

1. Kernighan & Ritchie: The C programming language

<http://zanasi.chem.unisa.it/download/C.pdf>

2. Kernighan & Ritchie: Limbajul C

Editura Teora, 2003



# Bibliografie

1. Kernighan & Ritchie: The C programming language

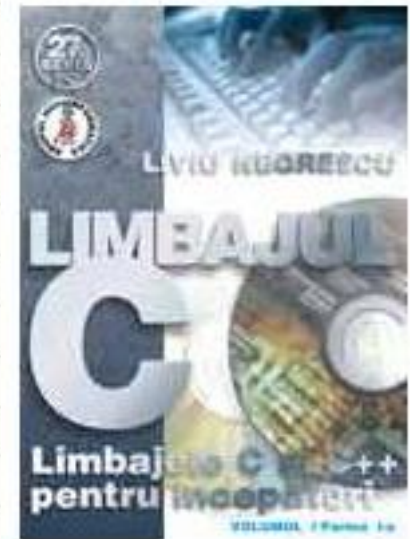
<http://zanasi.chem.unisa.it/download/C.pdf>

2. Kernighan & Ritchie: Limbajul C

Editura Teora, 2003

3. Liviu Negrescu: Limbajele C si C++ pentru începători,  
volumul 1, partea I si II (Limbajul C)

Editura Albastra, 2001



# Bibliografie

1. Kernighan & Ritchie: The C programming language

<http://zanasi.chem.unisa.it/download/C.pdf>

2. Kernighan & Ritchie: Limbajul C

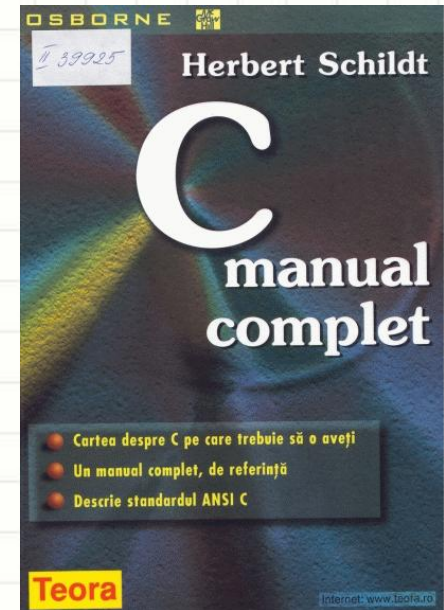
Editura Teora, 2003

3. Liviu Negrescu: Limbajele C si C++ pentru începători,  
volumul 1, partea I si II (Limbajul C)

Editura Albastra, 2001

4. Herbert Schildt: C, manual complet.

Editura Teora, 2000?





# Bibliografie

1. Kernighan & Ritchie: The C programming language

<http://zanasi.chem.unisa.it/download/C.pdf>

2. Kernighan & Ritchie: Limbajul C

Editura Teora, 2003

3. Liviu Negrescu: Limbajele C si C++ pentru începători,  
volumul 1, partea I si II (Limbajul C)

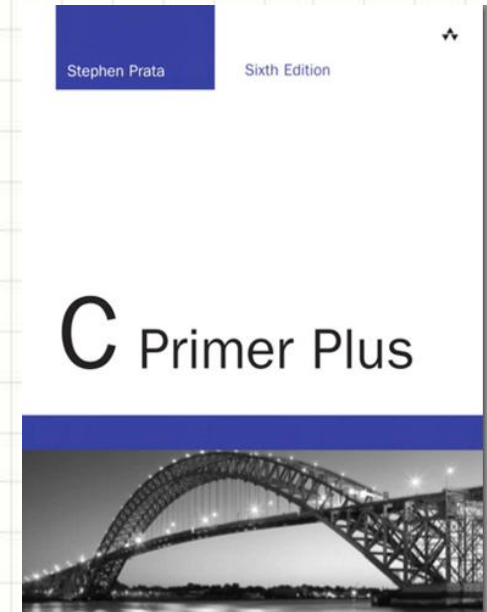
Editura Albastra, 2001

4. Herbert Schildt: C, manual complet.

Editura Teora, 2000?

5. Stephan Prata: C primer plus, 6<sup>th</sup> Edition

[https://vk.com/doc190970339\\_430409589?hash=2d2b4245bd65b25e27&dl=cd4e96f98aeddd5c1e](https://vk.com/doc190970339_430409589?hash=2d2b4245bd65b25e27&dl=cd4e96f98aeddd5c1e)



# Resurse online

- Cursuri online:
  - cautare după cuvinte cheie “online lectures C programming”
  - universități americane de prestigiu au conținutul cursurilor gratuit:
    - Stanford:  
<https://see.stanford.edu/Course/CS107> (online video lectures)
    - MIT: <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-087-practical-programming-in-c-january-iap-2010/lecture-notes/>
- Site-uri cu probleme de programare pentru concursuri
  - [www.infoarena.ro](http://www.infoarena.ro)
  - [www.acm.ro](http://www.acm.ro)
  - [www.topcoder.com](http://www.topcoder.com)

# Regulament de evaluare și notare

Nu este într-o formă finală detaliată, trebuie fixat cu asistentul de laborator. Îl definitivăm și vi-l prezint la cursul al doilea.

$$Nota = \min(10, \underbrace{Curs}_{6p} + \underbrace{Laborator}_{4p} + \underbrace{Seminar}_{1p})$$

# Mulțumiri pentru materiale/slide-uri:

- Anca Dobrovăț (FMI)
- Radu Boriga (FMI)
- Cristina Dăscălescu (FMI)
- Grigore Albeanu (FMI)
- Vlad Posea (Politehnică București)
- Traian Rebedea (Politehnică București)
- Kinga Marton (Politehnică Cluj)
- Ion Giosan (Politehnică Cluj)
- mulți alții ...

# Cuprinsul cursului de azi

1. Prezentarea cursului de Programare Procedurală
2. Primul curs



# Programa cursului

## ☐ Introducere



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

## ☐ Fundamentele limbajului C

- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Tipuri derivate de date: tablouri, șiruri 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 parametrelor. Pointeri la funcții.

## ☐ Tablouri și pointeri

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

## ☐ Șiruri de caractere

- Funcții specifice de manipulare.

## ☐ Fișiere binare

- Funcții specifice de manipulare.

## ☐ Structuri de date complexe și autoreferite

- Definiție și utilizare

## ☐ Funcții (2)

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

# Cursul 1:

1. Algoritmi

2. Limbaje de programare

3. Introducere în limbajul C. Structura unui program C.

# Algoritmi

Rezolvarea oricărei probleme implică mai multe etape:

1. Analiza problemei
2. Găsirea soluției [optime]
3. Elaborarea algoritmului
4. Implementarea algoritmului într-un limbaj de programare
5. Verificarea corectitudinii algoritmului propus
6. Analiza complexității

# Algoritmi

**Algoritm** = o succesiune finită, ordonată și bine definită (exprimată clar și precis) de operații executabile (instrucțiuni, pași) care constituie o metodă corectă de rezolvare a unei probleme pornind dintr-o stare inițială, folosind datele disponibile și ajungând în starea finală dorită.

**Exemplu:** algoritmul lui Euclid pentru determinarea celui mai mare divizor comun a două numere naturale (scăderi repetate, resturi)

$$1599 = 650 \times 2 + 299$$

$$650 = 299 \times 2 + 52$$

$$299 = 52 \times 5 + 39$$

$$52 = 39 \times 1 + 13$$

$$39 = 13 \times 3 + 0$$

$$\text{cmmdc}(1599, 650) = 13$$

1599

# Reprezentarea algoritmilor

1. Pseudocod/ limbaj natural
2. Schemă logică
3. Program într-un limbaj de programare



# Pseudocod

- limbaj natural structurat exprimat formal
- fiecare pas al algoritmului este reprezentat de o linie separată, ca o propoziție
- acțiuni (verbe) aplicate unor date (substantive)
- indentarea poate reda ierarhia instrucțiunilor

*Algoritmul lui Euclid  
prin scăderi repetate*

cât timp  $B > 0$

dacă  $A > B$

$A = A - B;$

altfel

$B = B - A;$

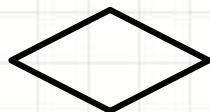
afișează  $A$

# Schemă logică

- alăturare de simboluri vizuale care desemnează fluxul logic al pașilor



Bloc de instrucțiuni



Structura alternativă



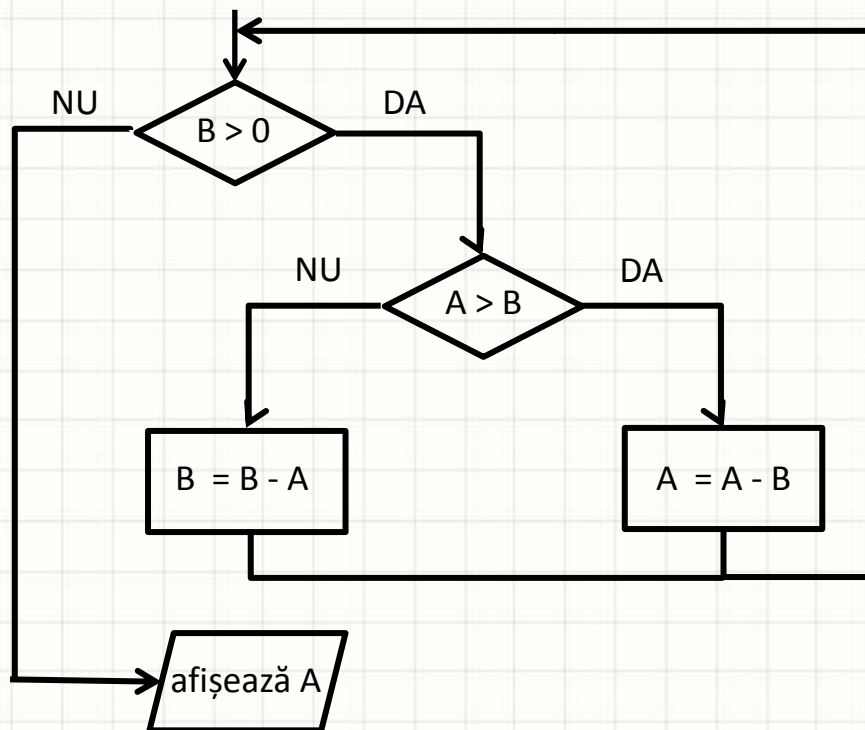
Direcția fluxului



Operația de intrare/ieșire

# Schemă logică

- alăturare de simboluri vizuale care desemnează fluxul logic al pașilor



*Algoritmul lui Euclid  
prin scăderi repetate*

cât timp  $B > 0$

dacă  $A > B$

$A = A - B;$

altfel

$B = B - A;$

afișează  $A$

# Cursul 1:

1. Algoritmi

2. Limbaje de programare

3. Introducere în limbajul C. Structura unui program C.

# Limbaje de programare

Rezolvarea oricărei probleme implică mai multe etape:

1. Analiza problemei
2. Găsirea soluției [optime]
3. Elaborarea algoritmului
4. Implementarea algoritmului într-un limbaj de programare
5. Verificarea corectitudinii algoritmului propus
6. Analiza complexității

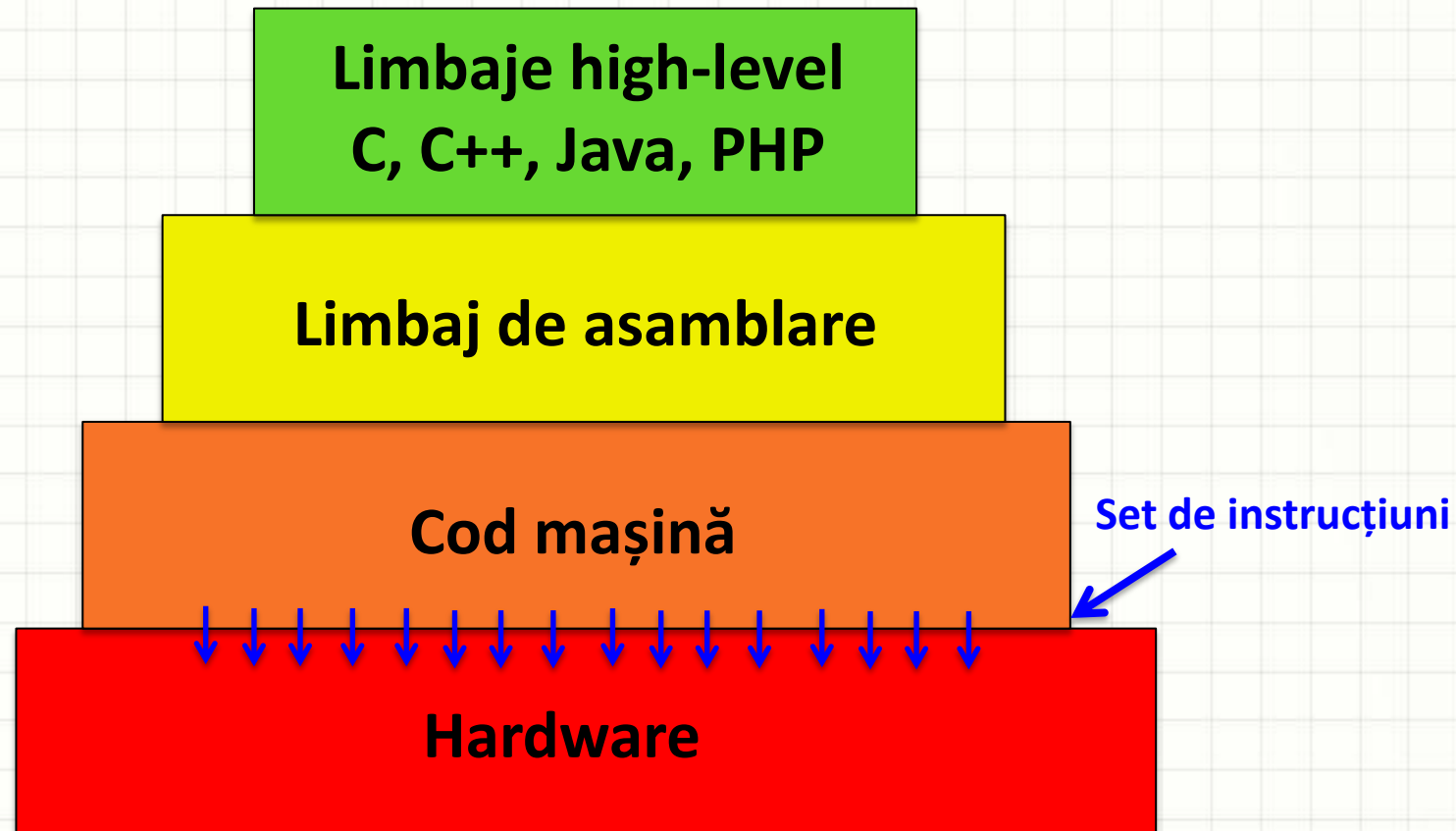


# Limbaaj de programare

- ❑ limbaaj artificial cu sintaxă și semantică bine definite
- ❑ pune la dispoziția programatorilor construcții sintactice prin care sunt specificate succesiunea de operații/instrucțiuni elementare (pe care un calculator le poate executa) asociate algoritmului de rezolvare a unei probleme;
- ❑ este necesară cunoașterea setului de operații/instrucțiuni elementare al calculatorului la care ne referim.
- ❑ **limbaaj mașină** = limbaajul nativ al unui calculator (mașină)

# Limbaje low-level și high-level

- dată de apropierea unui limbaj de limbajul nativ al calculatorului (limbaj mașină = cod mașină)



# Limbaje low-level (de nivel scăzut)

## Limbaj mașină

- ❑ limbajul nativ al unui calculator (mașină);
- ❑ șabloane de numere binare (reprezintă modul binar de codificare a instrucțiunilor și datelor în memorie )
- ❑ depinde de arhitectura sistemului

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
100011001111001000000000000000100
101011001111001000000000000000000
101011000110001000000000000000100
00000011111000000000000000001000
```

Instrucțiuni în limbaj mașină

## Limbaj de asamblare

- ❑ în loc de cod mașină folosește o desemnare simbolică a elementelor programului (instrucțiuni, date)
- ❑ 01011011 = ADD, 01011100 = SUB

```
swap:
    muli $2, $5, 4
    add $2, $4, $2
    lw $15, 0($2)
    lw $16, 4($2)
    sw $16, 0($2)
    sw $15, 4($2)
    jr $31
```

Instrucțiuni în  
limbaj de asamblare

# Limbaje high-level (de nivel înalt)

- ❑ cuprind mecanisme de exprimare apropiate de limbajul natural;
- ❑ folosesc verbe pentru a desemna acțiuni (**do, repeat, read, write, continue, switch, call, goto**, etc.), conjunctii (**if, while**), adverbe (**then, else**), mecanisme de declarare și definire.

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

Instrucțiuni în  
limbajul C

```
swap:
mul $2, $5, 4
add $2, $4, $2
lw $15, 0($2)
lw $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)
jr $31
```

Instrucțiuni în  
limbaj de asamblare

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
0000001111100000000000000001000
```

Instrucțiuni în limbaj mașină

# Limbaje high-level (de nivel înalt)

- ❑ au o descriere sintactică si semantică bine definită
- ❑ descurajează greșelile de programare
- ❑ independente de procesor (pentru asigurarea portabilității codului)
- ❑ independente de sistemul de operare (pentru a permite realizarea de software multi-platforma)
- ❑ codul sursă se convertește în cod mașină folosind compilatoare sau interpretoare



# Paradigme de programare

A: PARADIGMA PROGRAMARII PROCEDURALE SI STRUCTURATE :

Un program este privit ca o multime ierarhica de blocuri si proceduri;

B: PARADIGMA PROGRAMARII ORIENTATE SPRE OBJECT: Un program este constituit dintr-o colectie de obiecte care interactioneaza;

C: PARADIGMA PROGRAMARII CONCURENTE SI DISTRIBUITE:

Executia unui program este constituita din actiuni multiple posibil a fi executate in paralel pe una sau mai multe masini;

D: PARADIGMA PROGRAMARII FUNCTIONALE: Un program este descris pe baza unor functii de tip matematic (fara efecte secundare), utilizate de obicei recursiv;

E: PARADIGMA PROGRAMARII LOGICE: Un program este descris printr-un set de relatii intre obiecte precum si de restrictii ce definesc cadrul in care functioneaza acele obiecte. Executia inseamna activarea unui proces deductiv.

F: PARADIGMA PROGRAMARII LA NIVELUL BAZELOR DE DATE:

Actiunile programului sunt dictate de cerintele unei gestiuni corecte si consistente a bazelor de date asupra carora actioneaza programul.

# Paradigma programării procedurale

- ❑ execuție secvențială
- ❑ variabile reprezentate ca poziții în memorie și modificate prin atribuiri
- ❑ unitatea de program de bază: procedura = funcția = rutină = subrutină = subprogram
- ❑ C, Pascal, Fortran, Basic, Algol

# Cursul 1:

1. Algoritmi

2. Limbaje de programare

3. Introducere în limbajul C. Structura unui program C.

# Limbaajul 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 și gramatica 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
  - ☐ compilatoarele oferă suport limitat și în multe cazuri incomplet pentru acest standard
  
- ☐ **C11** – standard aprobat în 2011 și care rezolvă erorile apărute în standardul C99 și introduce noi elemente, însă suportul pentru C11 este și mai limitat decât suportul pentru C99, 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 limbajului C 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)

```
#include <stdio.h>
#define TA q=/*XYXY*/
#define/*X YXY*/CG r=
void p(int n,int c){;
for(;n--;) putchar(c)
#define Y(z)d;d=c++\
%2<1?x=x*4 +z,c%8>5?\
x=x?p(1,x), 0:x:0:0;d=
#define/*X YX*/C Y(1)
#define/*X YX*/G Y(2)
;}int(*f)( void),d,c,
#define/*X YX*/A Y(0)
#define/*XY*/AT int\
m(void/**/){d=
#define/*XYX*/T Y(3)
#define GC d; return\
0;}int(*f) (void )=m;
x,q,r; int main(){if(
f)f();else {for(puts(
#include "\40\"pro\
g.c\"\\n\\n \\101T"+0);
d=!d?x=(x=
<0?0:x,8*8 :d,TA++c%8
,TA(1+7*q- q*q)/3,r=c
*15-c*c-36 ,p(r<0?!q+
4:r/6+!q+4 ,32),q||x;
c%=16)q?p( 1,"ACGT"[x
/d&3]),p(q ,126),p(1,
"TGCA"[x/d &3]),d/=4,
p(001,10): puts(c%8?\
"CG":"TA") ;puts("GC"
);}return 0;}/**/
```

main.c

```
1 #include <stdio.h>
2 #define TA q=/*XYXY*/
3 #define/*X YXY*/CG r=
4 void p(int n,int c){;
5 for(;n--;) putchar(c)
6 #define Y(z)d;d=c++\
7 %2<1?x=x*4 +z,c%8>5?\
8 x=x?p(1,x), 0:x:0:0;d=
9 #define/*X YX*/C Y(1)
10 #define/*X YX*/G Y(2)
11 ;}int(*f)( void),d,c,
12 #define/*X YX*/A Y(0)
13 #define/*XY*/AT int\
14 m(void/**/){d=
15 #define/*XYX*/T Y(3)
16 #define GC d; return\
17 0;}int(*f) (void )=m;
18 x,q,r; int main(){if(
19 f)f();else {for(puts(
20 #include "\40\"pro\
21 g.c\"\\n\\n \\101T"+0);
22 d=!d?x=(x=
23 <0?0:x,8*8 :d,TA++c%8
24 ,TA(1+7*q- q*q)/3,r=c
25 *15-c*c-36 ,p(r<0?!q+
26 4:r/6+!q+4 ,32),q||x;
27 c%=16)q?p( 1,"ACGT"[x
28 /d&3]),p(q ,126),p(1,
29 "TGCA"[x/d &3]),d/=4,
30 p(001,10): puts(c%8?\
31 "CG":"TA") ;puts("GC"
32 );}return 0;}/**/
```

"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live."

~ John Woods

```
@staticmethod
def format_bytes(bytes):
    if bytes is None:
        return 'N/A'
    bytes = float(bytes)
    exponent = 0
    while bytes > 1024:
        bytes = bytes / 1024
        exponent += 1
    suffix = ['B', 'KIB', 'MIB', 'GIB', 'TIB', 'PiB', 'EIB', 'ZIB', 'YIB'][exponent]
    converted = float(bytes) / float(1024 ** exponent)
    return '%.2f%s' % (converted, suffix)

@staticmethod
def calc_percent(byte_counter, data_len):
    if data_len is None:
        return '---.-%'
    return '%s' % ('%.1f%%' % (float(byte_counter) / float(data_len) * 100.0))
```

# 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 C

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

# Primul program C explicat

```
1 #include <stdio.h>
```

Directivă de preprocesare pentru  
includerea bibliotecii standard de i/o

Antetul funcției principale

```
3 int main()  
4 {  
5     printf("Primul program scris in C\n");  
6     return 0;  
7 }
```

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 (afișare) +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 preprocesare*

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

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

## □ Directive de preprocesare

- 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 preprocesare*

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

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

## ❑ 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 preprocesare*

*declarații și definiții globale*

```
int main()
```

```
{
```

*declarații și definiții locale*

*instrucțiuni*

```
}
```