



# **Programare procedurala**

**- suport de curs -**

**Dobrovat Anca - Madalina**

**An universitar 2016 – 2017**  
**Semestrul I**

**Curs 1**



## **Agenda cursului**

**1. Regulamente UB si FMI**

**2. Utilitatea cursului de Programare Procedurala**

**3. Prezentarea disciplinei**

**4. Primul curs**



## Agenda cursului

- 1. Regulamente UB si FMI**
2. Utilitatea cursului de Programare Procedurala
3. Prezentarea disciplinei
4. Primul curs



# 1. Regulamente UB si FMI

## Lucruri bine de stiut de studenti:

regulament privind activitatea studenților la UB:

[http://fmi.unibuc.ro/ro/pdf/2016/regulamente/Regulamentul\\_privind\\_activitatea\\_profesionala\\_a\\_studentilor-2016.pdf](http://fmi.unibuc.ro/ro/pdf/2016/regulamente/Regulamentul_privind_activitatea_profesionala_a_studentilor-2016.pdf)

regulament de etică și profesionalism la FMI:

[http://fmi.unibuc.ro/ro/pdf/2015/consiliu/Regulament\\_etica\\_FMI.pdf](http://fmi.unibuc.ro/ro/pdf/2015/consiliu/Regulament_etica_FMI.pdf)

Se consideră **incident minor** cazul în care un student/ o studentă:  
a. preia codul sursă/ rezolvarea unei teme de la un coleg/ o colegă și pretinde că este rezultatul efortului propriu;

Se consideră **incident major** cazul în care un student/ o studentă:  
a. copiază la examene de orice tip;

**3 incidente minore = un incident major = exmatriculare**



## 1. Regulamente UB si FMI

Lucruri bine de stiut de studenti:

Cazuri

Exemplu:  
ELENA, grupa 403, a prezentat o gândire  
e materializată într-o copie a materialului. În  
un loc ocupat de un acufiș. În  
continuarea poate scrie de o altă  
defect, cu o altă acufiș.  
i învedere evenimentul drept incident  
major.

Exmatriculare pentru fraudă  
- cu drept de înscriere la admitere  
- conform consiliului FMI 12.03.2016

B. H. S.



## **Agenda cursului**

**1. Regulamente UB si FMI**

**2. Utilitatea cursului de Programare Procedurala**

**3. Prezentarea disciplinei**

**4. Primul curs**



## 2. Utilitatea cursului de Programare Procedurala

**Sursa:** <https://relus.com/top-10-programming-languages-to-learn-in-2016/>

### TOP 10 PROGRAMMING LANGUAGES

1. Java

2. C

3. C++

4. Python

5. C#

6. R

7. PHP

8. JavaScript

9. Ruby

10. Matlab

**Orientate Obiect**

**Paradigma Procedurala**

**THE IEEE SPECTRUM SURVEY**

**Programare Procedurala – Curs 1**



## 2. Utilitatea cursului de Programare Procedurala

**Paradigme de programare → Stil fundamental de a programa**

Dicteaza:

- **Cum se reprezinta datele problemei** (variabile, functii, obiecte, fapte, constrangeri etc)
- **Cum se prelucreaza reprezentarea** (atribuiri, evaluari, fire de executie, continuari, fluxuri etc)
- **Favorizeaza un set de concepte si tehnici de programare**
- **Influenceaza felul in care sunt ganditi algoritmi de rezolvare a problemelor**
- **Limbaje – in general multiparadigma (ex: Python – imperativ, functional, orientat pe obiecte)**





## 2. Utilitatea cursului de Programare Procedurala

### Paradigme de programare

Sursa: Albeanu G – Programare procedurala (note de curs 2013)

A: PARADIGMA PROGRAMARII PROCEDURALE SI STRUCTURATE :  
Un program este privit ca o multime ierarhica de blocuri si proceduri;  
B: PARADIGMA PROGRAMARII ORIENTATE SPRE OBIECT: 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.



## 2. Utilitatea cursului de Programare Procedurala

### Paradigma programarii procedurale si structurate

PP = paradigma de programare bazata pe conceptul de apel de procedura / functie / rutina / subrutina. Un program = multime ierarhica de functii care manipuleaza datele.

- Bazata pe ideile lui Von Neumann
- Starea programului variaza in functie de timp
- Executie secventiala (reteta)
- Variabile reprezentate ca locatii in memorie si modificate prin atribuirii
- Abstractiunea specifica: procedura
- C, Pascal, Fortran, Basic, Algol

Sursa: <http://andrei.clubcisco.ro/cursuri/f/f-sym/2pp/cb/curs1.pdf>



## 2. Utilitatea cursului de Programare Procedurala

### Paradigma programarii procedurale si structurate – Limbajul C

**Inventat si implementat:** Dennis Ritchie (anii 70)

BCPL (Martin Richards) → Limbajul B (Ken Thompson) → Limbajul C

Standardul “de facto” → versiunea ce insotea sistemul de operare UNIX

**Prima descriere:** “The C Programming Language” (Brian Kernighan si Dennis Ritchie)

**1983:** comitet pentru crearea unui standard ANSI (American National Standards Institute) → definire limbaj C (finalizat decembrie 1989).

stă la baza pentru majoritatea limbajelor “moderne”: C++, Java, C#, Javascript, Objective-C, etc.



## **Agenda cursului**

### **1. Regulamente UB si FMI**

### **2. Utilitatea cursului de Programare Procedurala**

### **3. Prezentarea disciplinei**

#### **3.1 Obiectivele disciplinei**

#### **3.2 Programa cursului**

#### **3.3 Bibliografie**

#### **3.4 Regulament de notare si evaluare**

#### **3.5 Notele de anul trecut (secția Matematica)**

### **4. Primul curs**



## 3. Prezentarea disciplinei

### 3.1 Obiectivele disciplinei

1. Formarea deprinderilor de **programare structurata** in limbaje de programare clasice si moderne (descompunerea unei probleme complexe in subprobleme mai simple si independente).
2. Insusirea **caracteristicilor Limbajului C** (implementarea unei probleme in C; intelegerea unui cod scris de altcineva; depanarea unui cod in C).
3. Deprinderea tehnicilor de **testare** si de **verificare** a corectitudinii programelor.
4. Dezvoltarea unei **gandiri algoritmice** si a unor **abilitati de programare**.



## 3. Prezentarea disciplinei

### 3.2 Programa cursului

<http://boboc.as-mi.ro/matematica/>

#### CONTINUT:

1. Probleme decidable si nedecidable
2. Algoritmi. Structuri de control si structuri elementare de date
3. Complexitatea si corectitudinea algoritmilor
4. Limbajul C. Tipuri de date (intregi, IEEE 754, inregistrari, uniuni, campuri de biti multidimensionale, pointeri)
5. Instructiuni C. Sintaxa si exemple
6. Functii. Transferul parametrilor. Argumente in linia de comanda
7. Alocare dinamica. Aplicatii privind procesarea structurilor: lista, arbore si (di)graf
8. Operatii de intrare – iesire (inclusiv fisiere) si aplicatii privind sortarea datelor stocate e
9. Pointeri la functii si functii cu numar variabil de argumente
10. Macrodefinitii si compilare conditionata
11. Legatura C – limbaj de asamblare
12. Metrice software si testarea programelor





## 3. Prezentarea disciplinei

### 3.2 Programa cursului (detaliata)

Sursa: Alexe B. – Programare procedurala (note de curs 2016)

#### ☐ Introducere

- Algoritmi
- Limbaje de programare.
- Introducere în limbajul C.

#### ☐ Fundamentele limbajului C

- Structura unui program 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 derivate de date

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

#### ☐ Funcții (1)

- Declarare și definire. Apel. Metode de transmitere a paramerilor.
- Pointeri la funcții.



## 3. Prezentarea disciplinei

### 3.2 Programa cursului (detaliata)

Sursa: Alexe B. – Programare procedurala (note de curs 2016)

#### ☐ 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 text și fișiere binare

- Funcții specifice de manipulare.

#### ☐ Structuri de date complex și autoreferite

- Definire și utilizare

#### ☐ Funcții (2)

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

#### ☐ Recursivitate





## 3. Prezentarea disciplinei

### 3.3 Bibliografie

- B.W. Kernighan, D.M. Ritchie, The C programming language, Prentice Hall, 1988 (2nd ed.).

<http://www.ime.usp.br/~pf/Kernighan-Ritchie/C-Programming-Ebook.pdf>

- Kernighan & Ritchie: Limbajul C, Editura Teora, 2003.

- B.W. Kernighan, R. Pike, The practice of programming, Addison-Wesley, 1999.

- Peter Salus, Handbook of Programming Languages: Vol. II: Imperative Programming Languages, Macmillan Technical Publishing, 1998.

- Herbert Schildt: C, manual complet. Editura Teora, 2000.

- Liviu Negrescu: Limbajele C si C++ pentru începători, volumul 1, partea I si II (Limbajul C), Editura Albastra, 2001.

•G. Albeanu, Algoritmi și limbaje de programare, Editura Fundației România de Măine, București, 2000.



## 3. Prezentarea disciplinei

### 3.3 Bibliografie (Webografie)

- Cursuri gratuite de la Universitati de prestigiu

Stanford <https://see.stanford.edu/Course/CS107>

MIT <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-087-practical-programming-in-c-january-iap-2010/>

- tutoriale de programare în C pentru începători (engleză):

1.

<https://www.youtube.com/watch?v=rk2fK2IliiQ&list=PLkB3phqR3X40reMCBYSonuPbDvM4kybMs>

2. <https://www.youtube.com/watch?v=Jlbs8ly6OKA&list=PL76809ED684A081F3>

3. [http://www.tutorialspoint.com/cprogramming/cprogramming\\_tutorial.pdf](http://www.tutorialspoint.com/cprogramming/cprogramming_tutorial.pdf)



### 3. Prezentarea disciplinei

#### 3.4 Regulament de notare si evaluare

$$\text{NOTA} = \text{CURS (5p)} + \text{LABORATOR (5p)}$$

#### **CURS (examen scris) – minim nota 5**

model de lucrare scrisă – ultimul curs (săptămâna 14)

#### **LABORATOR: – activitate (3p) + test (2p) (minim nota 5)**

- activitate
  - (2p) rezolvarea de probleme la laborator (săptămânal)
  - (1p) proiect - termen de predare săptămânile 12-13
    - teme de proiect în săptămâna 6
    - INDIVIDUAL
- test, **minim nota 5**
  - săptămâna 14 (ultima), model de test în săptămâna 13



### 3. Prezentarea disciplinei

#### 3.4 Regulament de notare si evaluare

Nu intrați în examen (=restanță) dacă:

- nu luați peste 5 (1 punct) la testul final din săptămâna 14;
- nu acumulați peste 2.5 puncte la laborator (din teme + test).

Aveți restanță dacă:

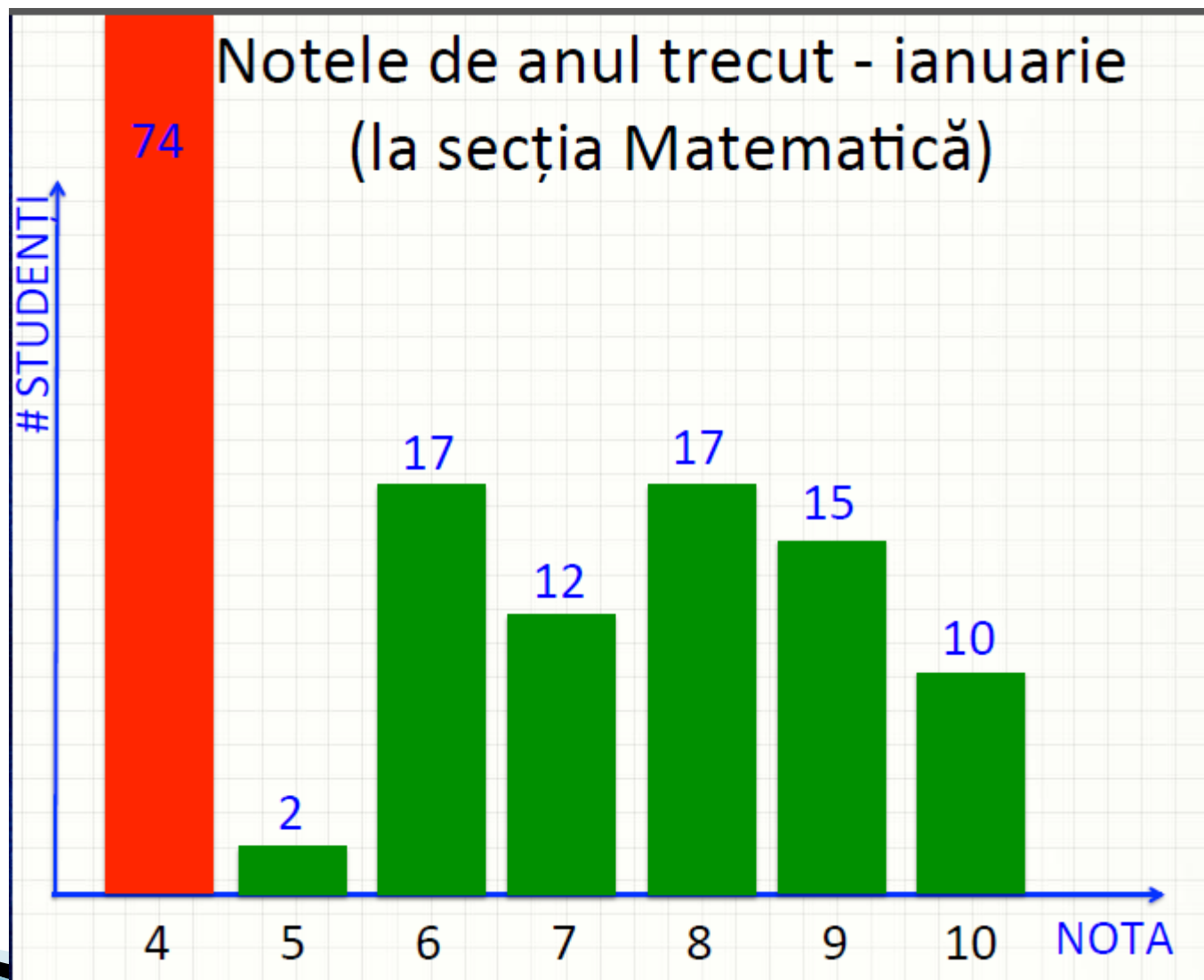
- nu intrați în examen;
- nu luați peste nota 5 (2 puncte) la curs.

- Prezența la curs **NU** e obligatorie
- Laboratoarele **SUNT** obligatorii! (excepție fac cei care participă la concursurile ACM)



### 3. Prezentarea disciplinei

#### 3.5 Notele de anul trecut (secția Matematica)





### 3. Prezentarea disciplinei

#### Semigrupele de laborator

- aveți libertatea să vă împărțiți cum vreți;
- dorim ca semi-grupele să fie echilibrate ca număr;
- dacă nu ajungeți la un consens formăm semigrupele după ordinea alfabetică (prima jumătate a catalogului = prima semi-grupă, a doua jumătate = a doua semi-grupă);
- semi-grupe definitive începând cu săptămâna 3;
- nu permitem apoi să vă transferați de la o semi-grupă la alta.



### 3. Prezentarea disciplinei

#### Compilatoare vs. IDE

Sursa: Alexe B. – Programare procedurala (note de curs 2016)

**Compiler** = program care transformă codul sursă al unui program scris într-un limbaj de programare în cod mașină.

**Example:** gcc, minGW, clang, etc

**IDE** = integrated development environment = mediu de dezvoltare care pune la dispoziția programatorului un editor pentru codul sursă, depanator, compiler

**Example:** Code::Blocks Visual Studio, Eclipse, Dev-C++, etc





## Agenda cursului

1. Regulamente UB si FMI
2. Utilitatea cursului de Programare Procedurala
3. Prezentarea disciplinei
4. **Primul curs**
  - 4.1 Algoritmi
  - 4.2 Limbaje de programare
  - 4.3 Introducere in Limbajul C





## 4. Curs 1

### 4.1 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



## 4. Curs 1

### 4.1 Algoritmi

#### Definitie:

**Algorithm** = secvență finită de comenzi explicite și neambigue care executate pentru o mulțime de date (ce satisfac anumite condiții inițiale), conduce în timp finit la rezultatul corespunzător.

#### Caracteristici:

- Generalitate
- Claritate
- Finititudine
- Corectitudine
- Performanță
- Robustețe



## 4. Curs 1

### 4.1 Algoritmi

#### Descriere / reprezentare:

Limbaj natural / Pseudocod – limbaj natural structurat exprimat formal

Diagramă (schemă logică) – alăturare de simboluri vizuale care desemnează fluxul logic al pașilor

Program - instructiuni



## 4. Curs 1

### 4.1 Algoritmi

#### Structuri de control

- entitățile de bază ale unui limbaj de programare structurat
- controlează modul de executare al unui program
- trei structuri de control fundamentale:
  1. structura secvențială;
  2. structura condițională (de decizie, de selecție);
  3. structura repetitivă (ciclică).



## 4. Curs 1

### 4.1 Algoritmi

#### Structura secventiala

- execută secvențial instrucțiuni

#### Pseudocod:

Instrucțiune1;

Instrucțiune2;

...

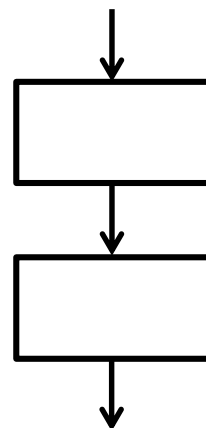
#### **Exemplu:**

aux = B;

A = B;

B = aux;

#### **Schemă logică:**





## 4. Curs 1

### 4.1 Algoritmi

#### Structura decizionala

- ramifică execuția programului în funcție de o condiție;
- instrucțiuni: **if**, **if else**, **switch**.

#### Instrucțiunea **IF**

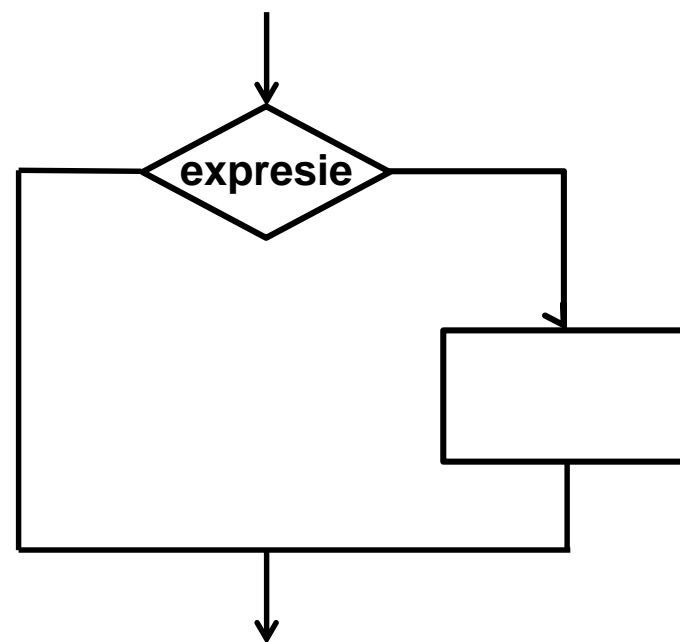
##### Sintaxa:

```
if (expresie)  
    instructiune1;
```

##### Pseudocod:

**dacă** (expresie) e adevărată  
 execută instructiune1;

#### Schemă logică





## 4. Curs 1

### 4.1 Algoritmi

### Structura decizionala

### Instrucțiunea IF ELSE

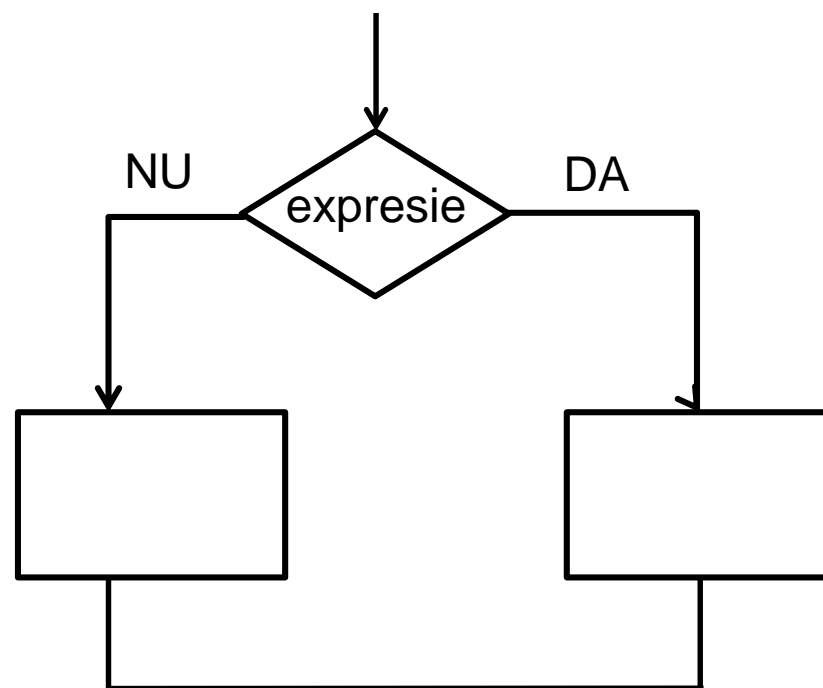
#### Sintaxa:

```
if (expresie)  
    instructiune1;  
else  
    instructiune2;
```

#### Pseudocod:

```
dacă (expresie) e adevărată  
    execută instructiune1;  
altfel  
    execută instructiune2;
```

#### Schemă logică





## 4. Curs 1

### 4.1 Algoritmi

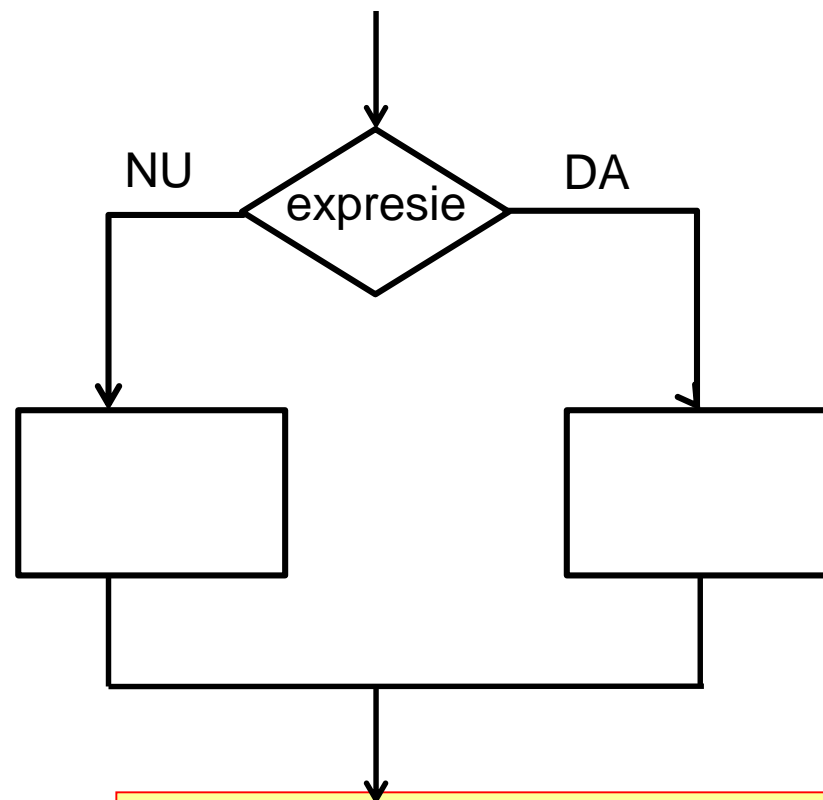
### Structura decizionala

Alternativă la **IF ELSE**: operatorul ternar ?

#### Sintaxa:

(expresie) ? (instructiune1) : instructiune2

#### Schemă logică



#### Pseudocod:

**dacă** (expresie) e adevărată  
    execută instructiune1;

**altfel**

    execută instructiune2;





## 4. Curs 1

### 4.1 Algoritmi

### Structura decizionala

Alternativă la **IF ELSE**: operatorul ternar ?

#### Sintaxa:

(expresie) ? ({set\_de\_instructiuni1}) :  
({set\_de\_instructiuni2});

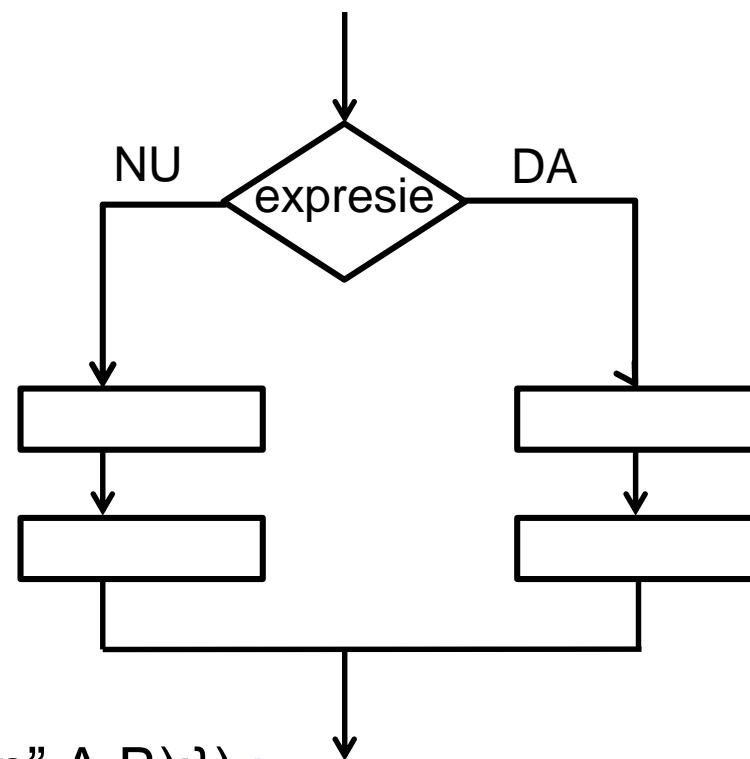
#### Pseudocod:

**dacă** (expresie) e adevărată  
    execută set\_de\_instructiuni1;  
**altfel**  
    execută set\_de\_instructiuni2;

#### Exemplu:

(A > B) ? ({A = A - B; printf("A=%d \n B = %d \n", A, B);}) :  
({B = B - A; printf("A=%d \n B = %d \n", A, B);});

#### Schemă logică





## 4. Curs 1

### 4.1 Algoritmi

### Structura decizionala

Alternativă la **IF ELSE**: operatorul ternar ?

#### Sintaxa IF ELSE:

```
if (expresie)
    instructiune1;
else
    instructiune2;
```

#### Operator ternar:

→ (expresie) ? (instructiune1) : (instructiune2);

dacă

altfel

**Exemplul 1:**  $(A > B) ? (A = A - B) : (B = B - A);$

**Exemplul 2:**  $(A > B) ? (\{A = A - B; \text{printf}("A > B");\}) :$   
 $(\{B = B - A; \text{printf}("B > A \n");\});$



## 4. Curs 1

### 4.1 Algoritmi

### Structura decizionala

### Instrucțiunea **SWITCH**

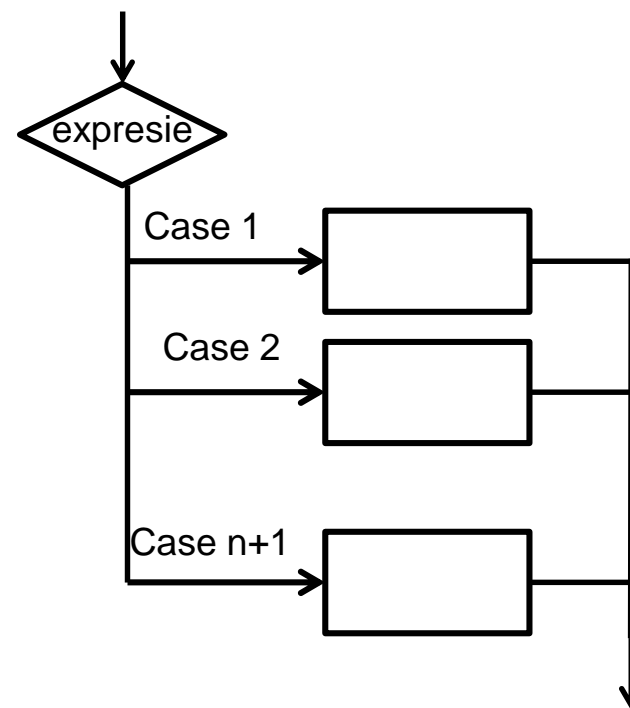
#### Sintaxa:

```
switch (expresie){  
    case (expresie_1):  
        instructiune_1;  
    case (expresie_2):  
        instructiune_2;  
    .....  
    case (expresie_n):  
        instructiune_n;  
    default:  
        instructiune_(n+1);  
}
```

#### Pseudocod:

```
dacă expresie=expresie_1  
    execută instructiune_1;  
altfel dacă expresie=expresie_2  
    execută instructiune_2;  
.....  
altfel dacă expresie=expresie_n  
    execută instructiune_n;  
altfel  
    instructiune_(n+1);
```

#### Schemă logică





## 4. Curs 1

### 4.1 Algoritmi

### Structura decizionala

#### Instrucțiunea **SWITCH**

#### Exemplu:

```
scanf("%c", &oper);  
switch (oper)  
{  
    case ('+'): printf("Operatorul de adunare!\n");  
                break;  
    case ('-'): printf("Operatorul de scadere!\n");  
                break;  
    case ('*'): printf(" Operatorul de inmultire!\n");  
                break;  
    default: printf("Operator ilegal!\n");  
}  
}
```



## 4. Curs 1

### 4.1 Algoritmi

#### Structura repetitiva

- repetă execuția unei [secvențe de] instrucțiuni în funcție de o condiție;

#### Clasificare:

- cu numar cunoscut de pasi (instrucțiunea **FOR**)
- cu numar necunoscut de pasi:
  - cu test initial (instrucțiunea **WHILE**)
  - cu test final (instrucțiunea **DO WHILE**)



## 4. Curs 1

### 4.1 Algoritmi

### Structura repetitiva

#### Instrucțiunea FOR

#### Sintaxa:

**for** (expresie1; expresie2; expresie3)  
    instructiune;

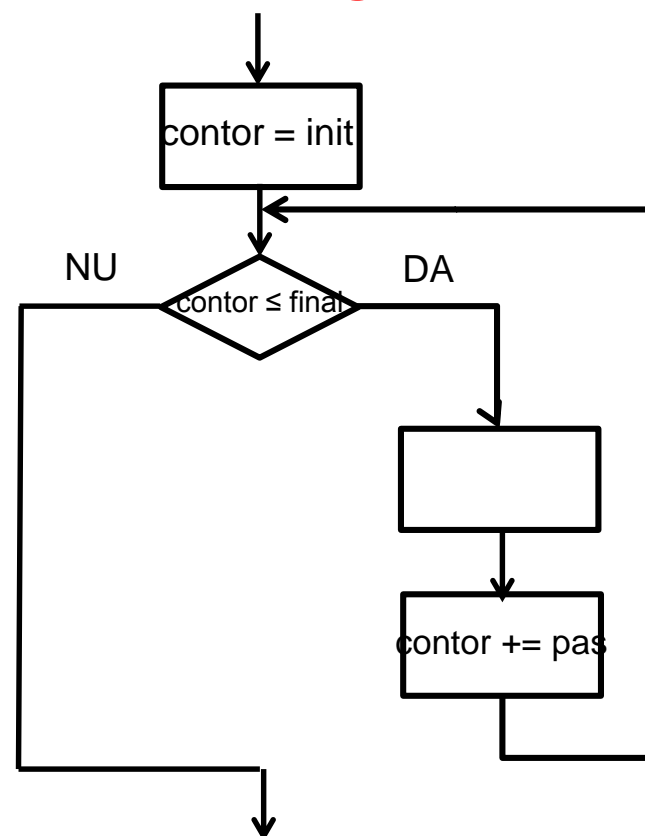
#### Pseudocod:

**pentru** contor  $\leftarrow$  init, final, pas  
    executa [set de instructiuni]

#### Cazuri:

1.  $\text{init} \leq \text{final} \Rightarrow \text{pas} / \text{cadenta pozitiv(a)}$
2.  $\text{init} \geq \text{final} \Rightarrow \text{pas} / \text{cadenta negativ(a)}$

#### Schemă logică





## 4. Curs 1

### 4.1 Algoritmi

### Structura repetitiva

#### Instrucțiunea FOR

**Obs:** Nu este obligatorie prezenta expresiilor, ci doar a instructiunilor vide.

```
for ( ; expresie2; )  
    instructiune;
```

sau:

```
for ( ; ; )  
    instructiune;
```

#### Exemple:

```
int S=0, P=1, k;  
for (k=1; k<=n; k++){  
    S+=k; P*=k;}
```

Suma si produsul  
primelor n nr naturale

```
for( ; c!='@'; ){  
    instructiuni  
}
```

Citirea unui caracter  
pana la intalnirea @



## 4. Curs 1

### 4.1 Algoritmi

### Structura repetitiva

Instrucțiunea **WHILE**

#### Sintaxa:

```
while (expresie)  
    instructiune;
```

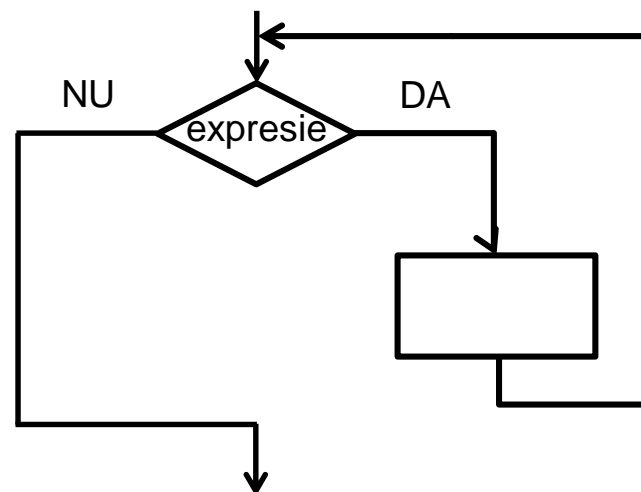
#### Pseudocod:

cât timp (expresie) este adevărată  
execută [set de instructiuni]

#### Exemplu:

```
int S=0, P=1, k=1;  
while (k<=n){  
    S+=k; P*=k;  
    k++;  
}
```

#### Schemă logică







## 4. Curs 1

### 4.1 Algoritmi

### Structura repetitiva

#### Sintaxa:

do  
instrucțiune;  
while (expresie) ;

#### Pseudocod:

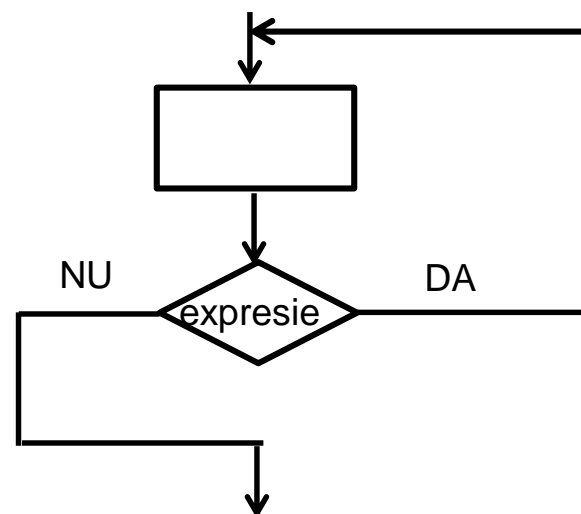
execută [set de instrucțiuni]  
cât timp (expresie) este adevărată

#### Exemplu:

```
int S=0, P=1, k=1;  
do {  
    S+=k; P*=k;  
    k++;  
} while (k<=n);
```

Instrucțiunea DO WHILE

#### Schemă logică





## 4. Curs 1

### 4.1 Algoritmi

### Structuri repetitive

#### EXAMPLE

Suma si produsul primelor n nr naturale

int S=0, P=1, k;

```
for (k=1; k<=n; k++)  
{  
    S+=k; P*=k;  
}
```

```
k=1;  
while (k<=n)  
{  
    S+=k; P*=k;  
    k++;  
}
```

```
k=1;  
do  
{  
    S+=k; P*=k;  
    k++;  
} while (k<=n);
```

Ce valori vor avea variabilele S si P pentru n=0 ?



## 4. Curs 1

### 4.1 Algoritmi

#### Structuri repetitive

#### Facilitati de intrerupere a unei secvente

##### Instructiunea Break

- asigura iesirea dintr-o bucla la nivelul imediat superior
- in cadrul instructiunii switch – pentru directionarea fluxului in afara instructiunii

##### Instructiunea Continue

- se utilizeaza pentru intreruperea executiei iteratiei curente

Se vor da mai multe detalii si exemple in cadrul cursului si al laboratorului.



## 4. Curs 1

### 4.1 Algoritmi

#### Structuri repetitive

##### Sintaxa:

`continue;`

#### Instructiunea **CONTINUE**

##### Pseudocod:

`continuă` execuția programului cu iterația următoare din bucla curentă;

citește 10 numere  
și află câte din ele  
sunt pare

##### Exemplu:

```
int nrPare=0, N=10, k, nr;  
for(k = 0; k < N; k++){  
    scanf("%d",&nr);  
    if ((nr % 2) != 0)  
        continue;  
    nrPare +=1;  
}  
printf("nrPare = %d\n",nrPare);
```



## 4. Curs 1

### 4.1 Algoritmi

#### Sintaxa:

`break;`

#### Pseudocod:

ieși din bucla curentă și continuă execuția programului cu instrucțiunea următoare;

la primul număr  
impar citit ieși din  
bucă (se citesc maxim  
10 numere pare)

#### Structuri repetitive

#### Instrucțiunea **BREAK**

#### Exemplu:

```
int nrPare=0, N=10, k, nr;  
for(k = 0; k < N; k++){  
    scanf("%d",&nr);  
    if ((nr % 2) != 0)  
        break;  
    nrPare +=1;  
}  
printf("nrPare = %d\n",nrPare);
```

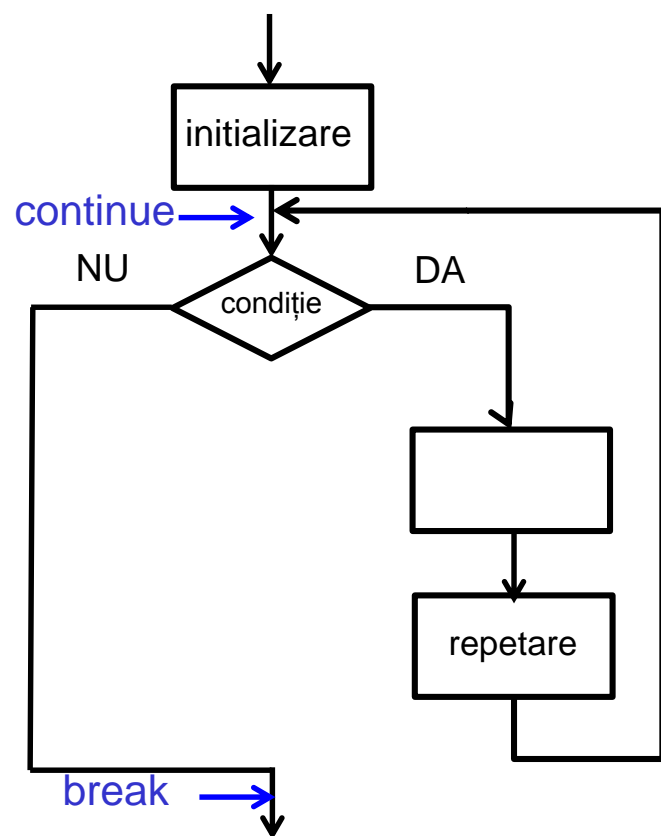


## 4. Curs 1

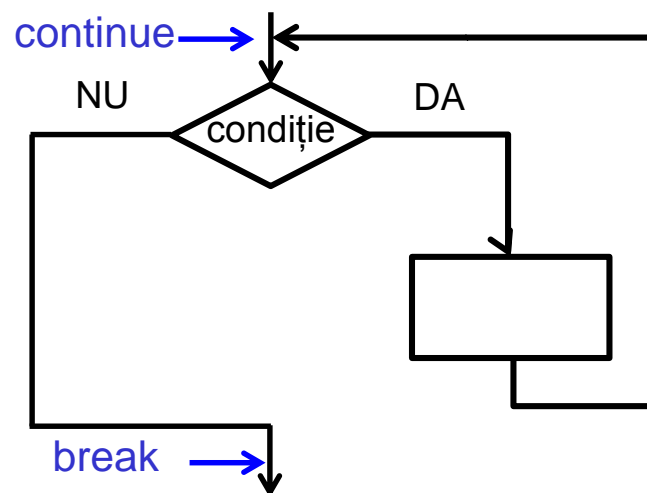
### 4.1 Algoritmi

### Structuri repetitive

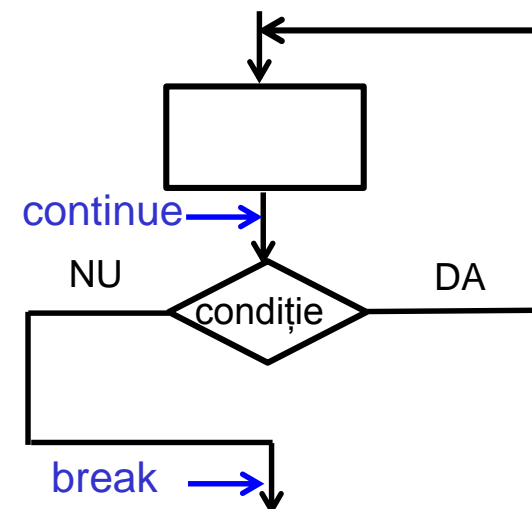
### Locurile în schema logică unde apar **CONTINUE** și **BREAK**



for



while



do ... while



## 4. Curs 1

### 4.2 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



## 4. Curs 1

### 4.2 Limbaje de programare

#### **Evolutie si clasificare**

**Limbaje de programare**

**Limbaje native / limbaje masina**

**Limbaje de asamblare**

**Limbaje de nivel inalt**





## 4. Curs 1

### 4.2 Limbaje de programare

**Limbaj de programare** - notație sistematică prin care este descris un proces de calcul.

**Proces de calcul** - succesiunea de operații elementare (pe care un calculator le poate executa) asociate algoritmului de rezolvare a unei probleme.

Limbajele de programare sunt limbaje artificiale.



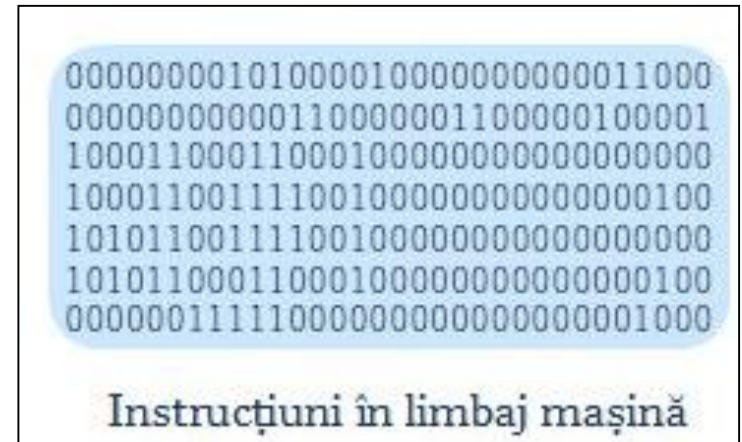
## 4. Curs 1

### 4.2 Limbaje de programare

#### Limbaje native / limbaje masina

constituite din multimea codurilor instructiunilor acceptate pentru executare (recunoscute).

- actuale: X86, MIPS, MMIX, etc.
- tipuri de masini: RISC (Reduced Instruction Set Computer), CISC (Complex Instruction Set Computer) etc.
- puterea masinii => Clasele de instructiuni si mecanismele de procesare implementate, formatul si lungimea instructiunilor, timpul pentru executarea fiecărei instructiuni.
- masini virtuale





## 4. Curs 1

### 4.2 Limbaje de programare

#### Limbaje de asamblare

Limbaje artificiale atribuie:

- fiecarui OpCode al unei masini reale, un nume simbolic (mnemonica),
- fiecarui registru al masinii de calcul, un mod de referire.

Pentru organizarea procesului de calcul descompune entitatile necesare in **sectiuni** (cod, date, extra, etc.)

Are un mecanism de referire prin **adrese simbolice (etichete)**.

```
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



## 4. Curs 1

### 4.2 Limbaje de programare

#### Limbaje de nivel inalt

- 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 declare si definire.
- oferă suport pentru importul/exportul de module (pachete, sub-proiecte).

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

Instrucțiuni în  
limbajul C



## 4. Curs 1

### 4.2 Limbaje de programare

#### Limbaje de nivel inalt

- 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)



## 4. Curs 1

### 4.2 Limbaje de programare Paradigme de programare

Sursa: Albeanu G – Programare procedurala (note de curs 2013)

A: PARADIGMA PROGRAMARII PROCEDURALE SI STRUCTURATE :  
Un program este privit ca o multime ierarhica de blocuri si proceduri;  
B: PARADIGMA PROGRAMARII ORIENTATE SPRE OBIECT: 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.





## 4. Curs 1

### 4.2 Limbaje de programare

### Paradigma programarii 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



## 4. Curs 1

### 4.2 Limbaje de programare

#### Structura generala a unui program in Programarea Procedurala

- modul principal (functia main, clasa aplicatie cu metoda main)
- zero, unul sau mai multe module (functii/proceduri, metode ale clasei aplicatie) care comunica intre ele si/sau cu modulul principal prin intermediul parametrilor si/sau a unor variabile globale

Unitatea de program cea mai mica si care contine cod este **functia / procedura** si contine:

- partea de declaratii/definitii
- partea imperativa (comenzile care se vor executa)

**Avantaje:** compilare separata, reutilizarea codului, lucrul in echipa, lucrul la distanta, testarea/verificarea codului – Unit testing





## 4. Curs 1

### 4.2 Limbaje de programare

#### Programarea Procedurala – Compilarea unitatilor de program

##### Analiza textului sursa

- lexicala – produce sir de atomi lexicali
- sintactica – produce arbori sintactici
- semantica – produce codul intermediar

##### Sinteza codului obiect

- optimizarea codului – produce cod intermediar optimizat
- generarea de cod – produce codul obiect final



## 4. Curs 1

### 4.2 Limbaje de programare

#### Programarea Procedurala – Compilarea unitatilor de program

##### In toate fazele

- se gestioneaza tabele (structuri de date specifice – functii hash pentru cautare rapida)
- se utilizeaza mecanisme de raportarea erorilor.

##### Codul intermediar poate fi:

- absolut (direct executabil)
- relocabil (editare de legaturi, translatarea adreselor)
- in limbaj de asamblare
- un alt limbaj de programare (cross-compilers)



## 4. Curs 1

### 4.2 Limbaje de programare

#### Programarea Procedurala – Editarea de legaturi

Mai multe module obiect (bucati de cod generate separat) se asambleaza impreuna cu module din bibliotecile standard pentru crearea aplicatiei finale.

Aplificatia finala este obtinuta prin segmentare (overlay) sau cu ajutorul bibliotecilor dinamice (dll)

**Build = Compilare + Editare de legaturi**



## 4. Curs 1

### 4.2 Limbaje de programare

#### Programarea Procedurala – Executarea programelor

Se realizeaza in urma pregatirii pentru executare de catre sistemul de operare.

Poate fi intrerupta (cazul sistemelor multitasking primitiv) de catre utilizator sau de catre sistemul de operare (sisteme multitasking pentru mai multi utilizatori) si poate fi reluata pe baza unei strategii de planificarea lucrarilor (prioritati, round robin, etc.)

Deoarece functia main (in C/C++) returneaza un rezultat aceasta face ca mai multe programe sa poata fi apelate dintr-un program “panou de comanda” pentru operare.



## 4. Curs 1

### 4.3 Introducere in 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.



## 4. Curs 1

### 4.3 Introducere in 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





## 4. Curs 1

### 4.3 Introducere in Limbajul C

**Programele C** sunt propozitii formate cu simboluri ale alfabetului C: atomi lexicali (tokens) si separatori.

**Atomii lexicali** - identificatori, constante, operatori, semne de punctuatie.

### Cuvinte cheie (32)

#### Cuvinte cheie adaugate de ANSI C:

enum, const, void,  
volatile, signed

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

Tabelul 1-2: Cele 32 de cuvinte-cheie definite de către standardul ANSI C



## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii

#### 5 tipuri de date de baza:

-char, int, float, double, void

#### Int pe 32 biti (Codeblocks)

#### Cum verificam?

```
sizeof(int) = 4  
sizeof(short int) = 2  
sizeof(long int) = 4  
sizeof(float) = 4  
sizeof(double) = 8  
sizeof(long double) = 8  
sizeof(char) = 1  
sizeof(signed char) = 1  
sizeof(unsigned char) = 1
```

Tip	Dimensiune aproximativă în biți	Domeniu minimal de valori
char	8	de la -127 la 127
unsigned char	8	de la 0 la 255
signed char	8	de la -127 la 127
int	16	de la -32767 la 32767
unsigned int	16	de la 0 la 65535
signed int	16	Similar cu int
short int	16	Similar cu int
unsigned short int	16	de la 0 la 65535
signed short int	16	Similar cu short int
long int	32	de la -2.147.483.647 la 2.147.483.647
signed long int	32	Similar cu long int
unsigned long int	32	de la 0 la 4.294.967.295
float	32	Şase zecimale exacte
double	64	Zece zecimale exacte
long double	80	Zece zecimale exacte

Tabelul 2-1 Toate tipurile de date definite prin standardul ANSI C





## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii

#### Modificarea tipurilor de baza:

- signed
- unsigned
- long
- short

Tip	Dimensiune aproximativă în biți	Domeniu minimal de valori
char	8	de la -127 la 127
unsigned char	8	de la 0 la 255
signed char	8	de la -127 la 127
int	16	de la -32767 la 32767
unsigned int	16	de la 0 la 65535
signed int	16	Similar cu int
short int	16	Similar cu int
unsigned short int	16	de la 0 la 65535
signed short int	16	Similar cu short int
long int	32	de la -2.147.483.647 la 2.147.483.647
signed long int	32	Similar cu long int
unsigned long int	32	de la 0 la 4.294.967.295
float	32	Şase zecimale exacte
double	64	Zece zecimale exacte
long double	80	Zece zecimale exacte

**Tabelul 2-1** Toate tipurile de date definite prin standardul ANSI C



## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii

Diferenta dintre intregii cu semn si fara semn → interpretarea bitului cu ordinul cel mai mare (indicator de semn).

0 – numere pozitive

1 – numere negative

#### Exemplu

int x = 190;

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

int x = -190;

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii

#### Nume de identificatori (variabile, functii, etichete etc.)

- unul sau mai multe caractere
- primul este “\_” sau o litera
- urmatoarele: cifre, litere sau “\_”

##### Corect

numarator  
test23  
bilant\_mare

##### Incorrect

1numarator  
salut!  
bilant...mare

**Case sensitive** – nume, NUME, Nume – identificatori diferiti

Un identificator nu poate fi un cuvânt cheie sau numele unei functii din biblioteca C



## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii

#### Variable

- numele variabilei nu are legatura cu tipul ei

#### Variable

- **locale** (definite in interiorul unei functii)
- **parametri formali** (declarare dupa numele functiei)
- **globale** (cunoscute de intreg programul)



## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii - Variabile locale

```
void func1(void)
{
    int x;
    x = 10;
}

void func2(void)
{
    int x;
    x = -199;
}

void f(void)
{
    int t;
    scanf("%d", &t);

    if(t==1){

        char s[80]; /* aceasta este creata doar la
                     intrarea in acest bloc*/
        printf("introduceti numele:");
        gets(s);
        /*faceti ceva...*/
    }
}
```

**x din func1() e diferit  
de x din func2()**

**Variabila locala s este creata la intrarea in  
blocul de cod if si se distruge la iesirea din el  
- Nu e accesibila din alta parte.**



## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii - Variabile locale

Există o diferență importantă între modul de declarare a variabilelor locale în C față de C++. În C, trebuie să declarați toate variabilele locale la începutul blocului în care le definiți, înainte de orice instrucțiuni ale programului. De exemplu, următoarea funcție este greșită dacă este compilată cu un compilator de C.

```
/* Aceasta functie este gresita daca este compilata cu un
   compilator de C, dar perfect acceptabila pentru un
   compilator de C++.
*/
void f(void)
{
    int i;
    i = 10;
    int j; /*aceasta linie va determina o eroare*/
    j = 20;
}
```



## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii - Variabile locale

Puteți inițializa o variabilă locală cu o valoare cunoscută. Această valoare va fi atribuită variabilei de fiecare dată când se va intra în blocul de cod în care este ea declarată. De exemplu, următorul program afișează numărul 10 de zece ori.

```
#include <stdio.h>
void f(void);
void main(void)
{
    int i;
    for(i=0; i<10; i++) f();
}
void f(void)
{
    int j = 10;
    printf("%d ", j);
    j++; /*aceasta linie nu are nici un efect */
}
```



## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii - Parametri formali

Daca o functie urmeaza sa foloseasca argumente, ea trebuie sa declare Variabilele pe care le accepta ca valori ale argumentelor (i.e. **parametri formali**).

Se comporta ca o variabila locala a functiei.

**Exemplu:**

```
Este_in (char *s, char c)
{
    while(*s)
        if (*s==c) return 1;
        else s++;
    return 0;
}
```

**Obs: Aritmetica pointerilor (se va discuta ulterior)!**

**Programul returneaza 1 daca c face parte din sirul s si 0 in caz contrar.**





## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii - Variabile globale

Se declara in afara oricarei functii si sunt cunoscute in intreg programul

Pot fi utilizate de catre orice zona a codului

Isi pastreaza valoarea pe parcursul intregii executii a programului.

Orice expresie are acces la ele, indiferent de tipul blocului de cod in care se afla expresia.

```
printf("contor este %d", contor); /*va afisa 100 */
}
void func2(void)
{
    int contor;
    for(contor = 1; contor<10; contor++)
        putchar('.');
}
```

```
#include <stdio.h>
int contor; /*contor este global */
void func1(void);

void func2(void);
void main(void)
{
    contor = 100;

    func1();

}
void func1(void)
{
    int temp;
    temp = contor;

    func2();
}
```



## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii - Variabile globale

**Obs: Exista variabila locala contor in func2()**

Cand func2() se refera la variabila contor,  
se refera la cea **locala**!!

```
#include <stdio.h>
int contor; /*contor este global */
void func1(void);

void func2(void);
void main(void)
{
    contor = 100;

    func1();

}
void func1(void)
{
    int temp;
    temp = contor;

    func2();
```

```
    printf("contor este %d", contor); /*va afisa 100 */
}
void func2(void)
{
    int contor;
    for(contor = 1; contor<10; contor++)

        putchar('.');
}
```



## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii - Modelatori de acces

##### Const

Variabilele de tip **const** nu pot fi modificate de program (dar pot primi valori initiale).

```
const int a = 10;
```

Modelatorul **const** poate fi folosit pentru a proteja obiectele indicate de argumentele unei functii pentru a nu fi modificate de acea functie.

##### Volatile

Valoarea unei variabile poate sa fie modificata pe cai nedecarate explicit de program (Expl. Adresa unei variabile globale poate fi transmisa rutinei ceasului sistemul de operare si utilizata pentru a pastra timpul real al sistemului → continutul variabilei se modifica fara o instructiune de atribuire explicita.

(Exemple in laborator).



## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii - Initializari de variabile


**tip nume\_variabila = constanta;**

**Expl.**

```
char ch = 'A';  
int x = 10;  
float media = 8.57;
```

#### Constante ([2])

(c)G.Albeanu

 **Constante intregi**

Orice constanta zecimala (octala sau hexazecimala) care depaseste cel mai mare intreg cu semn este considerata de tip long. Daca constanta depaseste cea mai mare valoare de tip signed long int este considerata a fi unsigned long int. Toate celelalte constantele intregi sunt de tip int. O constanta intreaga poate fi atribuita unei variabile caracter, rezultatul fiind ca la aplicarea conversiei (char).

- zecimale (baza 10; prima cifra nenula); 1234.
- octale (baza 8; prima cifra 0); 01234
- hexazecimale(baza 16, prefixul 0x sau 0X); 0xFF; 0xABBA.
- Efectul sufixului adaugat unei constante intregi (in functie de valoare):  
U sau u      **unsigned int** sau **unsigned long int**  
                 32u, 400000U  
L sau l      **long int** 32L, 32000L  
UL, ul, Ul, uL **unsigned long int** 32uL, 400000UL

Tipuri intregi (VC98) (c)G.Albeanu		
■ char=signed char	8 biti	-128..127
■ unsigned char	8 biti	0..255
■ short int = signed short int	16 biti	-32768..32767
■ unsigned int = unsigned short int	16 biti	0..65535
■ int=signed int = long int = signed long int	32 biti	-2.147.483.648 .. 2.147.483.647
■ unsigned long int	32 biti	0..4.294.967.295

Constantele intregi sunt formate din cifre ale bazei 10 (nu incep cu zero), bazei 8 (incep cu 0), si ale bazei 16 (incep cu 0x sau 0X). Tipul constantelor intregi depinde de forma, valoarea si



## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii - Constante ([2])

- ▶ Sunt formate din unul mai multe caractere incluse intre apostrofu. Pentru un caracter, tipul constantei este char, iar valoarea este reprezentata de codul ASCII.
- ▶ Daca sint mai multe caractere, intre apostrofu, tipul constantei este int, iar valoarea depinde de implementare.
- ▶ Constantele predefinite sint reprezentate de secventele speciale (escape).



#### Caractere - ASCII

(c)G.Albeanu

- un singur caracter, intre apostrofu, 'a'
- secvente speciale (escape – de evitare a situatiilor 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



## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii - Constante ([2])

## Constantele IEEE 754

- ▶ Aceste constante sunt compuse din semn, parte intreaga, punctul zecimal, parte fractionara, marcajul pentru exponent (e sau E si exponentul ca intreg cu semn).
- ▶ Partea intreaga sau partea fractionara pot lipsi (nu ambele).
- ▶ Punctul zecimal sau marcajul exponential pot lipsi (nu ambele).

### Constante in VIRGULA MOBIL

- format aritmetic; 3.1415
- format exponential; 31415E-4; 6.023E+23
- Sufixul F sau f forteaza tipul **float**
- Sufixul L sau l forteaza tipul **long double**
- Implicit constantele in virgula mobila sunt stocate conform tipului **double**

### Tipuri in virgula mobila

- **float** 32 biti  
+/- (3.4E-38..3.4E+38) 9 cifre.
- **double** 64 biti  
+/- (1.7E-308..1.7E+308) 15 cifre.
- **long double** 80 biti  
+/- (3.4E-4932..1.1E4932) 19 cifre.





## 4. Curs 1

### 4.3 Introducere in Limbajul C

#### Expresii - Constante ([2])

## Constantele sir de caractere

- ▶ String.h
- ▶ Au tipul **char[]**.
- ▶ Au clasa de memorare **static**
- ▶ "null-terminating string"
- ▶ Un sir de n caractere va avea n+1 octeti
- ▶ Unicode (Java - 2 octeti)

(c)G.Albeanu

### Tipuri de date fundamentale

- patru tipuri aritmetice de baza: char, int, float, double
- modificatori de tip - afecteaza domeniul de valori: signed, unsigned, short, long
- Tipul void - indica absenta oricarei valori. Este utilizat pentru: functii fara parametri, functii fara rezultat (proceduri!), tip pointer generic - conversie de tip cu operatorul cast pentru pointeri.
- sizeof(tip) sau sizeof <expresie> - pentru aflarea dimensiunii zonei de memorie a unui tip sau ocupata de o variabila.

### Siruri de caractere (c)G.Albeanu

- Secventa de caractere, inclusiv secvente escape, intre ghilimele.
- Exemplu: "Acesta este un sir\n"
- Au o reprezentare interna speciala. Sunt, de fapt, tablouri de caractere (array of char) care se incheie cu caracterul nul ('\0').



## Concluzii

### **1. S-a trecut in revista agenda cursului:**

- obiectivele disciplinei / programa cursului / bibliografia
- regulament de notare si evaluare

### **2. In cursul introductiv discutiile s-au axat pe 2 directii:**

- Algoritmi. Structuri de control si structuri elementare de date
- Limbaje de programare





## Perspective

**1. Se vor discuta directiile principale ale cursului, feedback-ul studentilor fiind hotarator in acest aspect**

- intelegerea notiunilor
- intrebari si sugestii

**Va doresc un an universitar usor!**

**Succes!**

**2. Cursul 2:**

- Complexitatea si corectitudinea algoritmilor