



# PROGRAMARE PROCEDURALĂ

Bogdan Alexe

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

Secția Informatică, anul I,

2016-2017

Cursul 11

# Evaluarea activităților didactice pe semestrul 1

- studenții au posibilitatea să evaluateze activitățile cadrelor didactice
  
- evaluare = completarea unui **chestionar** pentru fiecare disciplină (curs/seminar/laborator) ce conține **11 întrebări = 3-5 minute**
  
- evaluările au caracter anonim, fiecare student se va loga folosind un token nenominal de unică folosință primit de la secretariat de către șeful de grupă și distribuit apoi membrilor grupei.
  
- evaluarea se desfășoară în **perioada 5 – 15 ianuarie**
  
- concluziile din evaluarea de anul trecut vor fi făcute publice într-un raport (cel mai probabil în acest weekend)

# Chestionarul pentru curs

## 1. Aprecieri asupra materiei predate la curs

1.1. Dificultatea materiei a fost corespunzatoare nivelului meu de înțelegere  
a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

1.2. Resursele de învățare puse la dispozitie au fost accesibile și utile  
a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

1.3. Am înțeles utilitatea materiei în pregătirea mea profesională  
a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

## 2. Aprecieri asupra profesorului de curs

2.1 Ne-a prezentat de la începutul semestrului obiectivele cursului și cerințele de evaluare  
a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

2.2 A respectat pe parcursul semestrului obiectivele anunțate  
a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

2.3 Expunerea sa a fost clara, sistematica și coerenta  
a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

2.4 A fost entuziasmat și a prezentat materia într-o manieră atractivă  
a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

2.5 A prezentat materia într-o manieră interactivă și a fost disponibil pentru întrebări  
a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

2.6 A fost punctual și a utilizat eficient timpul programat activității de curs  
a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

2.7 A cultivat respectul reciproc între profesor și studenti  
a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

## 3. Ce mi-a plăcut / ce nu mi-a plăcut / ce am îmbunătăți

---

# Organizare

1. Examenul scris în sesiune: luni, 6 februarie 2017, ora 12:00. (neconfirmat)
2. Test de laborator: sâmbătă, 14 sau 21 ianuarie, ora ?? (neconfirmat)

**ANUL I 2016/2017**  
**PROGRAMAREA EXAMENELOR SESIUNEA DE IARNĂ 22.01–12.02 2016**

	<b>131</b> 32 studenți	<b>132</b> 29 studenți	<b>133</b> 31 studenți	<b>134</b> 31 studenți	<b>135</b> 25 studenți	<b>141</b> 31 studenți	<b>142</b> 26 studenți	<b>143</b> 31 studenți	<b>144</b> 33 studenți
DUMINICA 22.01.2017									
LUNI 23.01.2017		ARHITECTURA SISTEMELOR DE CALCUL PROF. GH. ȘTEFĂNESCU				ARHITECTURA SISTEMELOR DE CALCUL PROF. D. DRĂGULICI ORA 16,00			
MARTI 24.01.2017									
MIERCURI 25.01.2017		ALGORITMI ȘI STRUCTURI DE DATE PROF. R. CETERCHI ORA 11,00-14,00							

# Regulament de evaluare și notare

$$Nota = \min (10, 1 + Curs + Laborator + Seminar)$$
$$4p \qquad \qquad \qquad 5p \qquad \qquad \qquad 1p$$

Curs (4 puncte): examen scris se dă în sesiune o singură dată cu toți studenții care au intrat în examen.

Laborator (5 puncte): teme (3 puncte) + test (2 puncte) în săptămâna 14 cu toată seria.

Seminar (1 punct): 1 punct pentru activitate, posibil o lucrare.

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.

# Restanță – iunie și septembrie

$$Nota = \min (10, 1 + Curs + Laborator + Seminar)$$
$$\qquad\qquad\qquad 4p \qquad\qquad\qquad 5p \qquad\qquad\qquad 1p$$

Curs (4 puncte): examen scris.

Laborator (5 puncte): dacă ați luat laboratorul (minim 2.5 puncte la teme și test (minim 1 punct)) vi se păstrează nota, altfel dați un test (5 puncte)

Seminar (1 punct): vi se păstrează nota din timpul semestrului

# Restanță – anul universitar 2017-2018

$$Nota = \min (10, 1 + Curs + Laborator + Seminar)$$

*Curs*      *Laborator*      *Seminar*

4p                  5p                  1p

*Curs* (4 puncte): examen scris.

*Laborator* (5 puncte): refaceti laboratorul

*Seminar* (1 punct): refaceti seminarul

# Info'Clock

- curs facultativ ținut la FMI adresat studenților pasionați de programare, algoritmică, structuri de date, tehnici de programare care vor să participe la concursuri
- evaluare Info'Clock: marti, 11 ianuarie, 18-20 (confirmat)
- cei care obțin notă mare își pot echivala seminarul + laboratorul

# Programul în următoarele 5 săptămâni

- test Info'Clock – 11 ianuarie
- test laborator – 14 sau 21 ianuarie (?)
- notare seminar – veniți la seminar! (grupa 132 a mutat seminarul de azi pt luni, 9 ianuarie, ora 8, sala 1)
- 3 cursuri + 1 seminar (unele grupe mai au 2 seminarii)
- în ultimul curs (19 ianuarie) – subiect pentru examenul scris
- notele de la seminar + laborator le punem online imediat ce le vom avea
- examen scris final – 6 februarie, ora 12:00 (?)

# Cursul trecut (2016)

1. Fișiere: noțiuni generale
2. Fișiere text: funcții specifice de manipulare
3. Fișiere binare: funcții specifice de manipulare.
4. Funcții de poziționare în fișiere text și binare

# Programa cursului

- |   |   |
|---|---|
| <ul style="list-style-type: none"><li>□ <b>Introducere</b><ul style="list-style-type: none"><li>• Algoritmi.</li><li>• Limbaje de programare.</li><li>• Introducere în limbajul C. Structura unui program C.</li><li>• Complexitatea algoritmilor.</li></ul></li><br/><li>□ <b>Fundamentele limbajului C</b><ul style="list-style-type: none"><li>• Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.</li><li>• Instrucțiuni de control</li><li>• Directive de preprocesare. Macrodefiniții.</li><li>• Funcții de citire/scriere.</li><li>• Etapele realizării unui program C.</li></ul></li><br/><li>□ <b>Tipuri deriveate de date</b><ul style="list-style-type: none"><li>• Tablouri. Siruri de caractere.</li><li>• Structuri, uniuni, câmpuri de biți, enumerări.</li><li>• Pointeri.</li></ul></li><br/><li>□ <b>Funcții (1)</b><ul style="list-style-type: none"><li>• Declarare și definire. Apel. Metode de trasmitere a parametrilor.</li><li>• Pointeri la funcții.</li></ul></li></ul> | <ul style="list-style-type: none"><li>□ <b>Tablouri și pointeri</b><ul style="list-style-type: none"><li>▪ Legătura dintre tablouri și pointeri</li><li>▪ Aritmetică pointerilor</li><li>▪ Alocarea dinamică a memoriei</li><li>▪ Clase de memorare</li></ul></li><br/><li>□ <b>Siruri de caractere</b><ul style="list-style-type: none"><li>▪ Funcții specifice de manipulare.</li></ul></li><br/><li>□ <b>Fișiere text și fișiere binare</b><ul style="list-style-type: none"><li>▪ Funcții specifice de manipulare.</li></ul></li><br/><li>□ <b>Structuri de date complexe și autoreferite</b><ul style="list-style-type: none"><li>▪ Definire și utilizare</li></ul></li><br/><li>□ <b>Funcții (2)</b><ul style="list-style-type: none"><li>▪ Funcții cu număr variabil de argumente.</li><li>▪ Preluarea argumentelor funcției main din linia de comandă.</li><li>▪ Programare generică.</li></ul></li><br/><li>□ <b>Recursivitate</b></li></ul> |
|---|---|

# Cursul de azi

1. Declarații complexe
2. Structuri de date complexe și autoreferite
3. Preluarea argumentelor funcției main din linia de comandă

# Declarații complexe

- declarație complexă = combinație de pointeri, tablouri și funcții.
- combinăm *atributele*:
  - () – funcție: `int f();`
  - [] – tablou: `int t[20];`
  - \* - pointer: `int* p;`
- importanța parantezelor:
  - `int* f()` – f este o funcție ce returnează un pointer
  - `int *f()` – f este o funcție ce returnează un pointer (la fel ca sus)
    - \* este un operator prefixat și are o precedență mai mică decât cea a lui ()
  - `int (*f)()` - pointer la o funcție
    - parantezele sunt necesare pentru a impune asocierea corespunzătoare;
- declarațiile nu pot fi citite de la stânga la dreapta

# Declarații complexe

- *combinatii valide de atribute*:
  - **int \* f()** – funcție ce returnează un pointer la int
  - **int (\*f)()** – pointer la o funcție cu nici un argument ce returnează un int
  - **int \*t[10]** - tablou de 10 pointeri la int
  - **int (\*p) [13]** – pointer la un tablou de 13 int
  - **int t[5][6]** – tablou bidimensional
  - **int\* (\*f)()** - pointer la o funcție ce returnează un pointer
  - **int \*\*p** - pointer la un pointer (pointer dublu) la int
  - **int (\* p[7]) ()** - tablou de 7 pointeri la funcții
  - **int (\*(\*f)())[3][4]** - pointer la o funcție ce returnează un pointer la un tablou cu 3 linii si 4 coloane de int
  - **etc**

# Declarații complexe

- *combinatii invalide de atribute:*
  - []() – funcție ce returnează un tablou
  - ()[] – tablou de funcții
  - ()() – funcție ce returnează o funcție
- în limbajul C nu sunt permise declararea unui tablou de funcții, unei funcții care returnează un tablou/o funcție
- dacă vrem ca o funcție să întoarca rezultatul sub forma de tablou
  - transmitem tabloul ca argument prin adresa primului element
    - `void numeFunctie(int tablou[ ])`, `void numeFunctie(int* tablou)`
    - sau creăm tabloul (în Heap) și întoarcem pointerul corespunzător
      - `int* numeFunctie()`

# Declarații complexe

- interpretarea unei declarații complexe se face prin înlocuirea *atributelor* (pointeri, funcții , tablouri) prin următoarele *șabloane text*:

Atribut	Şablon text
()	funcția returnează
[n]	tablou de n
*	pointer la

- descifrarea unei declarații complexe se face aplicând *regula dreapta – stânga*, care presupune următorii pași:
  - se incepe cu identificatorul
  - se caută în dreapta identificatorului un atribut
  - dacă nu există, se caută în partea stângă
  - se substituie atributul cu şablonul text corespunzător
  - se continuă substituția dreapta-stânga
  - se oprește procesul la întâlnirea tipului datei.

# Declarații complexe

- ❑ *regula dreapta – stânga:*
  - ❑ se incepe cu identificatorul
  - ❑ se caută începând de la dreapta identificatorului un atribut
  - ❑ dacă nu există, se caută în partea stângă
  - ❑ se substituie atributul cu şablonul text corespunzător
  - ❑ se continuă substituția dreapta-stânga
  - ❑ se oprește procesul la întâlnirea tipului datei.
- ❑ **int (\* a[10]) ( );**
  - ❑ tablou de 10 pointeri la funcții ce returnează int
- ❑ **double (\*(\*pf())())[5][5];**
  - ❑ pointer la o funcție ce returnează un pointer la un tablou cu 5 linii și 5 coloane de double

# Declarații complexe

- <http://cdecl.org/> - instrument pentru conversia declarațiilor din C în limbaj natural

cdecl

C gibberish ↔ English

```
int (* a[10]) ( );
```

declare a as array 10 of pointer to function returning int

cdecl

C gibberish ↔ English

```
double (*(*pf())())[5][5];
```

declare pf as pointer to function returning pointer to array 5 of array 5  
of double

# Cursul de azi

1. Declarații complexe
2. Structuri de date complexe și autoreferite
3. Preluarea argumentelor funcției main din linia de comandă

# Structuri de date complexe și autoreferite

- ❑ structură – colecție de variabile grupate sub același nume.
- ❑ sintaxa:

```
struct <nume> {
    < tip 1 >    <variabila 1>;
    < tip 2 >    <variabila 2>;
    -----
    < tip n >    <variabila n>;
} lista_identificatori_de_tip_struct;
```

- ❑ variabilele care fac parte din structură sunt denumite membri (elemente sau câmpuri) ai structurii.

# Structuri

- **struct <nume> {**  
    < tip 1 >    <variabila 1>;  
    < tip 2 >    <variabila 2>;  
    -----  
    < tip n >    <variabila n>;  
} lista\_identificatori\_de\_tip\_struct;

## □ observații:

- dacă numele structurii (<nume>) lipsește, structura se numește **anonimă**. Dacă lista identificatorilor declarați lipsește, se definește doar tipul structură. Cel puțin una dintre aceste specificații trebuie să existe.
- dacă <nume> este prezent → se pot declara noi variabile de tip structura **struct <nume> <lista noilor identificatori>**;
- referirea unui membru al unei variabile de tip structură → operatorul de selecție punct **.** care precizează identificatorul variabilei și al câmpului.

# Structuri

## □ exemplu:

```
struct student {  
    char nume[30];  
    char prenume[30];  
    float medieIntrare;  
} A, B, C;
```

*Definește un tip de structură numit **student** și declară ca fiind de acest tip variabilele **A, B, C***

```
struct {  
    char nume[30];  
    char prenume[30];  
    float medieIntrare;  
} A;
```

*Declară o variabilă numită **A** definită de structura care o precede.*

```
typedef struct {  
    char nume[30];  
    char prenume[30];  
    float medieIntrare;  
} student;
```

```
student A;
```

*Definește un tip de date numit **student** și declară ca fiind de acest tip variabila **A***

# Structuri – inițializare

## □ exemplu:

student.c

```
01 #include<stdio.h>
02
03
04     typedef struct {
05         char nume[30];
06         char prenume[30];
07         float medieIntrare;
08     } student;
09
10 int main()
11 {
12     student A = {"Popescu", "Maria", 9.25};
13     student B = {.medieIntrare = 9.25, .prenume = "Maria", .nume = "Popescu"};
14     student C = {"Popescu"};
15
16     printf("%s %s %f \n", A.nume, A.prenume, A.medieIntrare);
17     printf("%s %s %f \n", B.nume, B.prenume, B.medieIntrare);
18     printf("%s %s %f \n", C.nume, C.prenume, C.medieIntrare);
19
20     return 0;
21 }
```

```
typedef struct {
    char nume[30];
    char prenume[30];
    float medieIntrare;
} student;
```

```
P/curs11/student
Popescu Maria 9.250000
Popescu Maria 9.250000
Popescu 0.000000
```

# Transmiterea structurilor ca parametri

```
student1.c ✘
1 #include<stdio.h>
2
3
4     typedef struct {
5         char nume[30];
6         char prenume[30];
7         float medieIntrare;
8     } student;
9
10    void adaugaUnPunct(student x)
11    {
12        x.medieIntrare++;
13        if (x.medieIntrare > 10)
14            x.medieIntrare = 10;
15    }
16
17    int main()
18    {
19        student A = {"Popescu", "Maria", 9.25};
20        adaugaUnPunct(A);
21        printf("%s %s %f \n", A.nume, A.prenume, A.medieIntrare);
22        return 0;
23    }
```

P:/curs11/student1  
Popescu Maria 9.250000  
Poder Aluno Macbook Pro

# Transmiterea structurilor ca parametri

- ❑ când o structură este transmisă ca parametru unei funcții se face o copie a zonei de memorie respective
- ❑ transmitere prin valoare
- ❑ la ieșirea din funcție se distrugе copia locală (x în exemplul anterior)
- ❑ modificările efectuate asupra structurii în funcție nu vor afecta și structura originală

# Pointeri la structuri

- folosim operatorul `->` pentru a accesa campurile

```
student2.c ✘
1 #include<stdio.h>
2
3
4     typedef struct {
5         char nume[30];
6         char prenume[30];
7         float medieIntrare;
8     } student;
9
10 void adaugaUnPunct(student* x)
11 {
12     x->medieIntrare++;
13     if (x->medieIntrare > 10)
14         x->medieIntrare = 10;
15 }
16
17 int main()
18 {
19     student A = {"Popescu", "Maria", 9.25};
20     adaugaUnPunct(&A);
21     printf("%s %s %f \n", A.nume, A.prenume, A.medieIntrare);
22     return 0;
23 }
```

P/curs11/student2  
Popescu Maria 10.000000

# Pointeri la structuri

- ❑ folosim operatorul `->` pentru a accesa campurile
- ❑ respectă aceleasi reguli ca și ceilalți pointeri
- ❑ trebuie sa-i initializam si sa avem grija sa facem conversii explicite cand este cazul
- ❑ pointeri la structuri vs. structuri care conțin pointeri

# Structuri imbricate și tablouri de structuri

- ❑ o structură este imbricată (nested) dacă ea conține ca membru o altă structură

```
struct student{  
    char nume[30];  
    char prenume[30];  
    float medieIntrare;  
    struct adresa;  
}
```

Structura **adresa** trebuie să fie definită în prealabil

- ❑ tablou de structuri: tablou cu elemente de tip struct  
**struct student grupa[30];**

# Structuri imbricate și tablouri de structuri

- ❑ fișierul text *triunghi.txt* contine pe prima linie un număr natural n,  $n > 0$ , apoi n linii. Fiecare linie conține coordonatele reale (abscisa si ordonata) a 3 puncte date sub forma:

abscisa1 ordonata1 abscisa2 ordonata 2 abscisa 3 ordonata 3

Să se afișeze aria celui mare triunghi daca acesta există, sau mesajul “nu există” daca nici un triplet de puncte de pe o linie nu poate defini un triunghi.



```
triunghi.txt
3
0.5 0.5 1.5 1.5 2.5 2.5
0 -1 -1 1 1 1
2 2 0 0 0.5 0
```

# Structuri imbricate și tablouri de structuri

- ❑ fișierul text *triunghi.txt* contine pe prima linie un număr natural n,  $n > 0$ , apoi n linii. Fiecare linie conține coordonatele reale (abscisa si ordonata) a 3 puncte date sub forma:

abscisa1 ordonata1 abscisa2 ordonata 2 abscisa 3 ordonata 3

Să se afișeze aria celui mare triunghi daca acesta există, sau mesajul “nu există” daca nici un triplet de puncte de pe o linie nu poate defini un triunghi.

```
typedef struct {  
    float abscisa;  
    float ordonata;  
} punct2D;
```

Definește un tip de date *punct2D*.

```
typedef struct {  
    punct2D A, B, C;  
    float AB, AC, BC;  
    int triunghiValid;  
    float perimetru;  
    float arie;  
} triunghi;
```

# Structuri imbricate și tablouri de structuri

- ❑ fișierul text *triunghi.txt* contine pe prima linie un număr natural n,  $n > 0$ , apoi n linii. Fiecare linie conține coordonatele reale (abscisa si ordonata) a 3 puncte date sub forma:

abscisa1 ordonata1 abscisa2 ordonata 2 abscisa 3 ordonata 3

Să se afișeze aria celui mare triunghi daca acesta există, sau mesajul "nu există" daca nici un triplet de puncte de pe o linie nu poate defini un triunghi.

## Formula lui Heron

De la Wikipedia, enciclopedia liberă

În geometrie, **formula lui Heron**, descoperită de [Heron din Alexandria](#), este o expresie matematică prin care se poate calcula suprafața unui [triunghi](#) oarecare fiind date cele trei laturi.

Dacă ABC este un triunghi oarecare, cu laturile a, b și c, atunci aria sa este dată de formula:

$$S_{ABC} = \sqrt{p(p - a)(p - b)(p - c)}$$

unde  $p = \frac{(a+b+c)}{2}$  reprezintă semiperimetru triunghiului dat.

# Structuri imbricate și tablouri de structuri

- ❑ fișierul text *triunghi.txt* contine pe prima linie un număr natural  $n$ ,  $n > 0$ , apoi  $n$  linii. Fiecare linie conține coordonatele reale (abscisa și ordonata) a 3 puncte date sub forma:

abscisa1 ordonata1 abscisa2 ordonata2 abscisa3 ordonata3

Să se afișeze aria celui mare triunghi dacă acesta există, sau mesajul "nu există" dacă nici un triplet de puncte de pe o linie nu poate defini un triunghi.

## Formula lui Heron

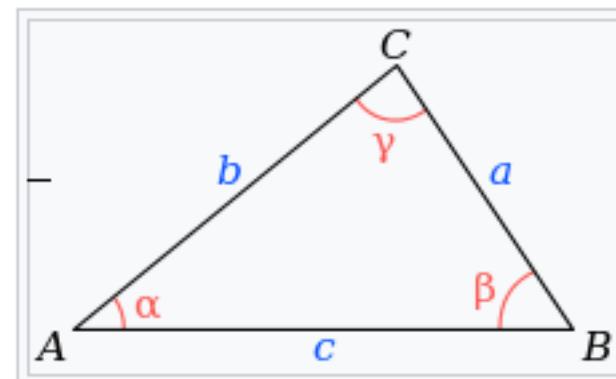
De la Wikipedia, enciclopedia liberă

În geometrie, **formula lui Heron**, descoperită de [Heron din Alexandria](#), este o expresie pentru suprafața unui triunghi oarecare fiind date cele trei laturi.

Dacă  $ABC$  este un triunghi oarecare, cu laturile  $a$ ,  $b$  și  $c$ , atunci aria sa este dată de formula

$$S_{ABC} = \sqrt{p(p - a)(p - b)(p - c)}$$

unde  $p = \frac{(a+b+c)}{2}$  reprezintă semiperimetru triunghiului dat.



Un triunghi de laturi  $a$ ,  $b$  și  $c$ .

[https://ro.wikipedia.org/wiki/Formul%C3%A3\\_lui\\_Heron](https://ro.wikipedia.org/wiki/Formul%C3%A3_lui_Heron)

# Structuri imbricate și tablouri de structuri

triunghi.c 

```
1 #include <stdio.h>
2
3 typedef struct {
4     float abscisa;
5     float ordonata;
6 } punct2D;
7
8 typedef struct{
9     punct2D A, B, C;
10    float AB, AC, BC;
11    int triunghiValid;
12    float perimetru;
13    float arie;
14 } triunghi;
15
16 int citesteTriunghiuri(char*, triunghi**);
17 int afisareTriunghiuri(triunghi*,int);
```

# Structuri imbricate și tablouri de structuri

```
49 int citesteTriunghiuri(char *nume, triunghi **pT)
50 {
51     FILE* f = fopen(nume,"r");
52     if(f==NULL) { printf("Eroare la citirea fisierului \n"); exit(0); }
53     int i,n;
54     fscanf(f,"%d",&n);
55     triunghi* p = (triunghi *) malloc(n*sizeof(triunghi));
56     if (p==NULL) { printf("Eroare la alocare"); exit(0); }
57     punct2D A,B,C;
58     for(i=0;i<n;i++)
59     {
60         fscanf(f,"%f %f %f %f %f",&A.abscisa,&Aordonata,&B.abscisa,
61                 &Bordonata,&C.abscisa,&Cordonata);
62         float AB = sqrt(pow(A.abscisa - B.abscisa,2) + pow(Aordonata - Bordonata,2));
63         float AC = sqrt(pow(A.abscisa - C.abscisa,2) + pow(Aordonata - Cordonata,2));
64         float BC = sqrt(pow(B.abscisa - C.abscisa,2) + pow(Bordonata - Cordonata,2));
65         p[i].A = A; p[i].B = B; p[i].C = C; p[i].AB = AB; p[i].AC = AC; p[i].BC = BC;
66         //validare triunghi
67         if ((AB + BC == AC) || (AB + AC == BC) || (AC + BC == AB))
68             p[i].triunghiValid = 0;
69         else
70             p[i].triunghiValid = 1;
71         if(p[i].triunghiValid)
72         {
73             p[i].perimetru = AB + AC + BC;
74             float sp = p[i].perimetru/2;
75             p[i].arie = sqrt(sp * (sp - AB) * (sp - BC) * (sp - AC));
76         }
77     }
78     *pT = p;
79     fclose(f);
80     return n;
81 }
```

# Structuri imbricate și tablouri de structuri

```
81 int afisareTriunghiuri(triunghi* pT,int n)
82 {
83     int i;
84     for(i=0;i<n;i++)
85     {
86         printf("*****\n");
87         if(pT[i].triunghiValid)
88             printf("Triunghiul %d: \n",i);
89         printf("Punctul A are coordonatele (%f,%f) \n",pT[i].A.abscisa,pT[i].Aordonata);
90         printf("Punctul B are coordonatele (%f,%f) \n",pT[i].B.abscisa,pT[i].Bordonata);
91         printf("Punctul C are coordonatele (%f,%f) \n",pT[i].C.abscisa,pT[i].Cordonata);
92         printf("Segmentul AB are lungimea %f \n",pT[i].AB);
93         printf("Segmentul AC are lungimea %f \n",pT[i].AC);
94         printf("Segmentul BC are lungimea %f \n",pT[i].BC);

95         if(!pT[i].triunghiValid)
96             printf("Punctele sunt coliniare si nu formeaza un triunghi\n");
97         if(pT[i].triunghiValid)
98             printf("Aria triunghiului este %f \n");
99     }
100 }
101 }
```

# Structuri imbricate și tablouri de structuri

```
19 int main()
20 {
21     char numeFisier[] = "/Users/bogdan/Desktop/triunghi.txt";
22     triunghi *T = NULL;
23     int n = citesteTriunghiuri(numeFisier,&T);
24     afisareTriunghiuri(T,n);
25
26     int i;
27     float arieMaxima = 0;
28     int indiceTriunghi = -1;
29     for (i=0;i<n;i++)
30     {
31         if (T[i].triunghiValid)
32             if (arieMaxima < T[i].arie)
33                 {arieMaxima = T[i].arie; indiceTriunghi = i;}
34     }
35     if (arieMaxima)
36     {
37         printf("Triunghiul de arie maxima are aria = %f \n",T[indiceTriunghi].arie);
38         printf("Triunghiul contine punctele: (%f,%f) (%f,%f) (%f,%f) \n",
39                 T[indiceTriunghi].A.abscisa,T[indiceTriunghi].Aordonata,
40                 T[indiceTriunghi].B.abscisa, T[indiceTriunghi].Aordonata,
41                 T[indiceTriunghi].C.abscisa, T[indiceTriunghi].Cordonata);
42     }
43     else
44         printf("Nu exista \n");
45
46     return 0;
47 }
```

# Structuri imbricate și tablouri de structuri

```
*****
Punctul A are coordonatele (0.500000,0.500000)
Punctul B are coordonatele (1.500000,1.500000)
Punctul C are coordonatele (2.500000,2.500000)
Segmentul AB are lungimea 1.414214
Segmentul AC are lungimea 2.828427
Segmentul BC are lungimea 1.414214
Punctele sunt coliniare si nu formeaza un triunghi
*****
Triunghiul 1:
Punctul A are coordonatele (0.000000,-1.000000)
Punctul B are coordonatele (-1.000000,1.000000)
Punctul C are coordonatele (1.000000,1.000000)
Segmentul AB are lungimea 2.236068
Segmentul AC are lungimea 2.236068
Segmentul BC are lungimea 2.000000
Aria triunghiului este 2.000000
*****
Triunghiul 2:
Punctul A are coordonatele (2.000000,2.000000)
Punctul B are coordonatele (0.000000,0.000000)
Punctul C are coordonatele (0.500000,0.000000)
Segmentul AB are lungimea 2.828427
Segmentul AC are lungimea 2.500000
Segmentul BC are lungimea 0.500000
Aria triunghiului este 0.500000
Triunghiul de arie maxima are aria = 2.000000
Triunghiul contine punctele: (0.000000,-1.000000) (-1.000000,-1.000000) (1.000000,1.000000)
```

# Sortarea unui tablou de structuri

```
108 void sorteazaTriunghiuri(triunghi* p, int n)
109 {
110     triunghi aux;
111
112     int i,j;
113     for(i=0;i<n;i++)
114     {
115         if(!p[i].triunghiValid)
116             p[i].arie = 0;
117     }
118     for(i=0;i<n;i++)
119         for(j=i+1;j<n;j++)
120             if(p[i].arie < p[j].arie)
121             {
122                 aux = p[i];
123                 p[i] = p[j];
124                 p[j] = aux;
125             }
126     return;
127 }
```

```
sorteazaTriunghiuri(T,n);
afisareTriunghiuri(T,n);
```

# Sortarea unui tablou de structuri

```
*****
Triunghiul 0:  
Punctul A are coordonatele (0.000000,-1.000000)  
Punctul B are coordonatele (-1.000000,1.000000)  
Punctul C are coordonatele (1.000000,1.000000)  
Segmentul AB are lungimea 2.236068  
Segmentul AC are lungimea 2.236068  
Segmentul BC are lungimea 2.000000  
Aria triunghiului este 2.000000  
*****  
Triunghiul 1:  
Punctul A are coordonatele (2.000000,2.000000)  
Punctul B are coordonatele (0.000000,0.000000)  
Punctul C are coordonatele (0.500000,0.000000)  
Segmentul AB are lungimea 2.828427  
Segmentul AC are lungimea 2.500000  
Segmentul BC are lungimea 0.500000  
Aria triunghiului este 0.500000  
*****  
Punctul A are coordonatele (0.500000,0.500000)  
Punctul B are coordonatele (1.500000,1.500000)  
Punctul C are coordonatele (2.500000,2.500000)  
Segmentul AB are lungimea 1.414214  
Segmentul AC are lungimea 2.828427  
Segmentul BC are lungimea 1.414214  
Punctele sunt coliniare si nu formeaza un triunghi
```

# Structuri autoreferite

- ❑ structuri care contin o declarație recursivă pentru anumiți membri de tip pointer

```
struct T{  
    char ch;  
    int i;  
    struct T *t;  
}
```

declaratie valida

```
struct T{  
    char ch;  
    struct S *p;  
}
```

```
struct S{  
    int i;  
    struct T *q;  
}
```

declaratie valida – structurile S și T se invoca reciproc

- ❑ este ilegal ca o structură să conțină o instanțiere a sa

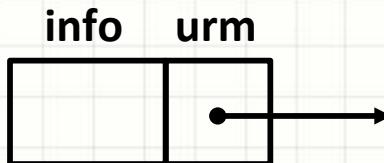
```
struct T{  
    char ch;  
    int i;  
    struct T t;  
}
```

declaratie invalida

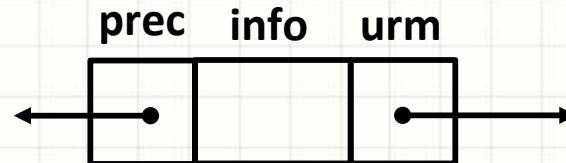
# Structuri autoreferite

- ❑ aplicații pentru structuri de date în alocare dinamică

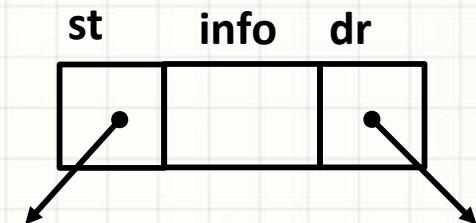
```
struct nod{  
    int info;  
    struct nod *urm;  
}  
  
declaratie unui  
structuri nod
```



```
struct nod{  
    int info;  
    struct nod *urm;  
    struct nod *prec;  
}
```



```
struct nod{  
    int info;  
    struct nod *fiuSt;  
    struct nod *fiuDr;  
}
```



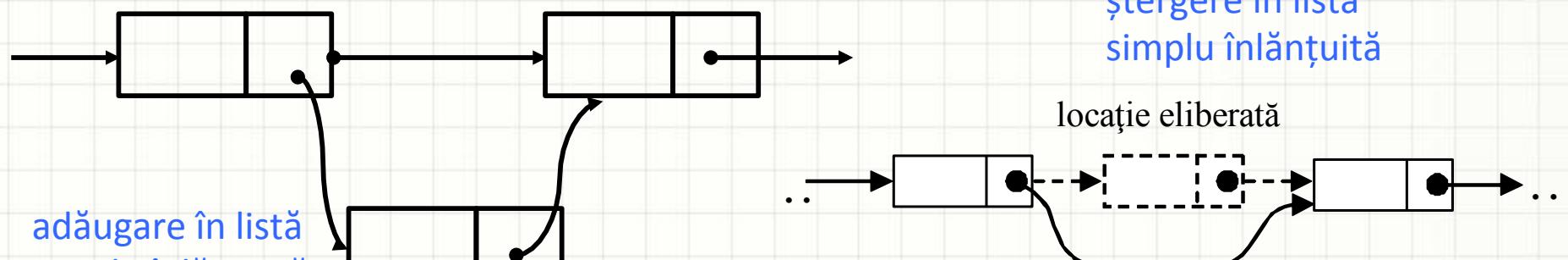
- ❑ fiecare nod conține:
  - ❑ un câmp/mai multe câmpuri cu informația nodului - **info**
  - ❑ un pointer/mai mulți pointeri către nodurile vecine: următor, precedent, fiuStang, fiuDrept

# Structuri autoreferite

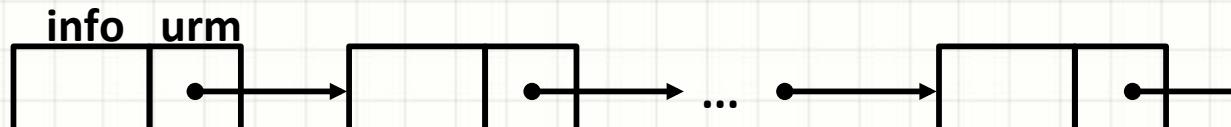
- aplicații pentru structuri de date în alocare dinamică
- liste, stive, cozi, arbori
  - avantaj față de implementarea statică
    - operatiile de adaugare sau stergere sunt foarte rapide
  - dezavantaj față de implementarea statică :
    - accesul la un nod se face prin parcurgerea nodurilor precedente
    - adresa nodurilor vecine ocupa memorie suplimentara

# Structuri autoreferite

- ❑ aplicații pentru structuri de date în alocare dinamică
- ❑ operațiile de adăugare, stergere, traversare, căutare sunt specifice pentru fiecare structură de date



traversare, căutare în listă simplu înlănțuită



# Structuri autoreferite

- ❑ aplicații pentru structuri de date în alocare dinamică
- ❑ operații cu vectori rari: suma și produsul scalar a doi vectori rari
  - ❑ procent mare dintre elementele vectorului egale cu 0.
  - ❑ reprezentare eficientă → liste simplu înlăntuite alocate dinamic
  - ❑ fiecare nod din lista retine:
    - ❑ valoarea
    - ❑ poziției din vector pe care se găsește elementul nenul
- ❑ citesc vectorii din două fișiere text ce specifică valoarea și poziția elementelor nenele din ambii vectori

```
struct nod{  
    float info;  
    int poz;  
    struct nod *urm;  
}  
declaratie a structurii nod  
folosită la reprezentarea listei
```

# Structuri autoreferite

- operații cu vectori rari: suma și produsul scalar a doi vectori rari

  	 vector1
1 10 10 2.5 100 -5.6 1000 3.6	   5 18 10 3.1 90 46 100 5.2 500 3.4

```
1 # include<stdio.h>
2 # include<stdlib.h>
3
4 typedef struct
5 {
6     float info;
7     int poz;
8     struct nod *urm;
9 } nod;
10
11 void adaugare(nod**, nod**, float, int);
12 void construiesteLista(nod**, nod**, char* );
13 void suma(nod*, nod*, nod**, nod**);
14 float produsScalar(nod*, nod*);
```

# Structuri autoreferite

- operații cu vectori rari: suma și produsul scalar a doi vectori rari

```
15
16     int main()
17     {
18         char* numeFisiere[] = {"./Users/bogdan/Desktop/vector1.txt", "./Users/bogdan/Desktop/vector2.txt"};
19
20         nod *prim1 = NULL, *ultim1 = NULL;
21         construiesteLista(&prim1,&ultim1,numeFisiere[0]);
22         afiseazaLista(prim1);
23
24         nod *prim2 = NULL, *ultim2 = NULL;
25         construiesteLista(&prim2,&ultim2,numeFisiere[1]);
26         afiseazaLista(prim2);
27
28         nod *prim3 = NULL, *ultim3 = NULL;
29         suma(prim1,prim2,&prim3,&ultim3);
30         afiseazaLista(prim3);
31         printf("*****\n");
32         printf("Produsul scalar a celor doi vectori este: %f \n",produsScalar(prim1,prim2));
33
34         return 0;
35     }
36
37     void afiseazaLista(nod *p)
38     {
39         printf("*****\n");
40         while(p)
41         {
42             printf("%d %f \n",p->poz, p->info);
43             p = p->urm;
44         }
45     }
46
47     void suma(nod *prim1, nod *prim2, nod *&prim3, nod *&ultim3)
48     {
49         if(prim1 == NULL && prim2 == NULL)
50         {
51             *prim3 = NULL;
52             *ultim3 = NULL;
53             return;
54         }
55
56         if(prim1 == NULL)
57         {
58             *prim3 = *prim2;
59             *ultim3 = *ultim2;
60             return;
61         }
62
63         if(prim2 == NULL)
64         {
65             *prim3 = *prim1;
66             *ultim3 = *ultim1;
67             return;
68         }
69
70         if(prim1->poz < prim2->poz)
71         {
72             *prim3 = *prim1;
73             *ultim3 = *ultim1;
74             prim1 = prim1->urm;
75             if(prim1 == NULL)
76             {
77                 *ultim3 = *prim3;
78                 return;
79             }
80             suma(prim1, prim2, &prim3, &ultim3);
81         }
82         else
83         {
84             *prim3 = *prim2;
85             *ultim3 = *ultim2;
86             prim2 = prim2->urm;
87             if(prim2 == NULL)
88             {
89                 *ultim3 = *prim3;
90                 return;
91             }
92             suma(prim1, prim2, &prim3, &ultim3);
93         }
94     }
95
96     float produsScalar(nod *prim1, nod *prim2)
97     {
98         float produs = 0;
99
100        if(prim1 == NULL && prim2 == NULL)
101        {
102            return produs;
103        }
104
105        if(prim1 == NULL)
106        {
107            return produs;
108        }
109
110        if(prim2 == NULL)
111        {
112            return produs;
113        }
114
115        if(prim1->poz < prim2->poz)
116        {
117            return produs;
118        }
119
120        produs += prim1->info * prim2->info;
121        prim1 = prim1->urm;
122        prim2 = prim2->urm;
123        produsScalar(prim1, prim2);
124    }
125
```

# Structuri autoreferite

- operații cu vectori rari: suma și produsul scalar a doi vectori rari

```
82
83     void construiesteLista(nod** p, nod** u, char* numeFisier)
84     {
85         FILE *f = fopen(numeFisier,"r");
86         if (f==NULL)
87         {
88             printf("Eroare la deschiderea de fisier \n");
89             exit(0);
90         }
91
92         int poz;
93         float val;
94         while(1)
95         {
96             if (fscanf(f,"%d %f",&poz,&val)==2)
97                 adaugare(p, u, val, poz);
98             if (feof(f))
99                 return;
100        }
101    }
102 }
```

# Structuri autoreferite

- operații cu vectori rari: suma și produsul scalar a doi vectori rari

```
105 void adaugare(nod** p, nod** u, float val, int poz)
106 {
107     if (*p == NULL)
108     {
109         *p = (nod *) malloc(sizeof(nod));
110         (*p)->info = val;
111         (*p)->poz = poz;
112         (*p)->urm == NULL;
113         *u = *p;
114     }
115     else
116     {
117         nod *c = (nod *) malloc(sizeof(nod));
118         c->info = val;
119         c->urm = NULL;
120         c->poz = poz;
121         (*u)->urm = c;
122         *u = c;
123     }
124 }
125 }
```

# Structuri autoreferite

- operații cu vectori rari: suma și produsul scalar a doi vectori rari

```
38 void suma(nod *prim1, nod *prim2, nod **prim3, nod **ultim3)
39 {
40
41     nod *p1, *p2;
42     p1 = prim1;
43     p2 = prim2;
44     while (p1 != NULL && p2 != NULL)
45     {
46         if (p1->poz < p2->poz)
47         {
48             adaugare(prim3, ultim3, p1 -> info, p1 -> poz);
49             p1 = p1 -> urm;
50         }
51     else
52         if (p1->poz > p2->poz)
53         {
54             adaugare(prim3, ultim3, p2 -> info, p2 -> poz);
55             p2 = p2 -> urm;
56         }
57
58     else // (p1->poz == p2->poz)
59     {
60         adaugare(prim3, ultim3, p1 -> info + p2 -> info, p2 -> poz);
61         p1 = p1 -> urm; p2 = p2 -> urm;
62     }
63 }
```

# Structuri autoreferite

- operații cu vectori rari: suma și produsul scalar a doi vectori rari

```
65 while(p1)
66 {
67     adaugare(prim3, ultim3, p1 -> info, p1 -> poz);
68     p1 = p1 -> urm;
69 }
70
71 while(p2)
72 {
73     adaugare(prim3, ultim3, p2 -> info, p2 -> poz);
74     p2 = p2 -> urm;
75 }
76
77 }
```

# Structuri autoreferite

- operații cu vectori rari: suma și produsul scalar a doi vectori rari

```
P:\curs11\vectoriRari
*****
1 10.000000
10 2.500000
100 -5.600000
1000 3.600000
*****
5 18.000000
10 3.100000
90 46.000000
100 5.200000
500 3.400000
*****
1 10.000000
5 18.000000
10 5.600000
90 46.000000
100 -0.400000
500 3.400000
1000 3.600000
*****
Produsul scalar a celor doi vectori este: -21.369999
```

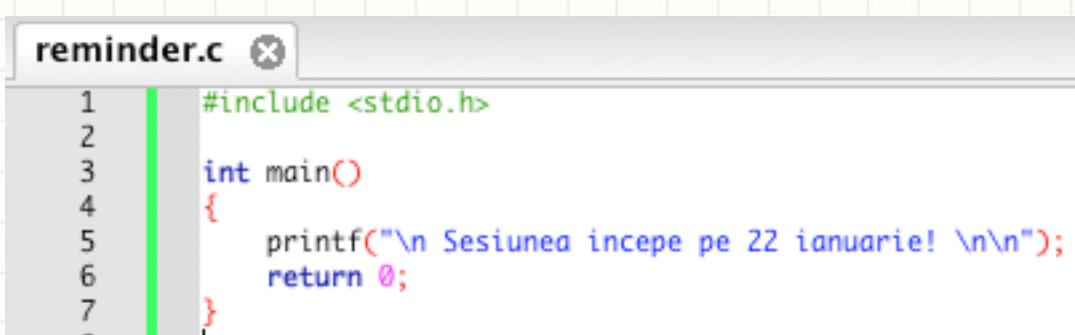
# Cursul de azi

1. Declarații complexe
2. Structuri de date complexe și autoreferite
3. Preluarea argumentelor funcției main din linia de comandă

# Preluarea argumentelor funcției main din linia de comandă

- un program executabil (comandă) poate fi lansat în execuție de către interpretorul de comenzi al sistemului de operare.
- de exemplu, programul **reminder** care afișează la terminal un mesaj:

```
Bogdan-Alexes-MacBook-Pro:curs11 bogdan$ ./reminder  
Sesiunea incepe pe 22 ianuarie!
```



```
reminder.c ✘  
1 #include <stdio.h>  
2  
3 int main()  
4 {  
5     printf("\n Sesiunea incepe pe 22 ianuarie! \n\n");  
6     return 0;  
7 }
```

- program fără parametri, lansat în execuție prin:  
**>./reminder**

# Preluarea argumentelor funcției main din linia de comandă

- un program poate avea și parametri, care apar după numele comenzi și sunt separați prin spații libere.
- de exemplu, programul **reminder** ar putea avea un parametru sir de caractere, fiind lansat în execuție, în acest caz prin:

> **./reminder parametru**

```
Bogdan-Alexes-MacBook-Pro:curs11 bogdan$ ./reminder 18
```

```
Sesiunea incepe pe 22 ianuarie! Au mai ramas 18 zile!
```

# Preluarea argumentelor funcției main din linia de comandă

- un program poate avea și parametri, care apar după numele comenzi și sunt separați prin spații libere.
- programul poate avea acces la parametrii liniei de comandă, dacă funcția **main()** prezintă argumente:

```
int main(int argc, char *argv[])
{ ... }
```

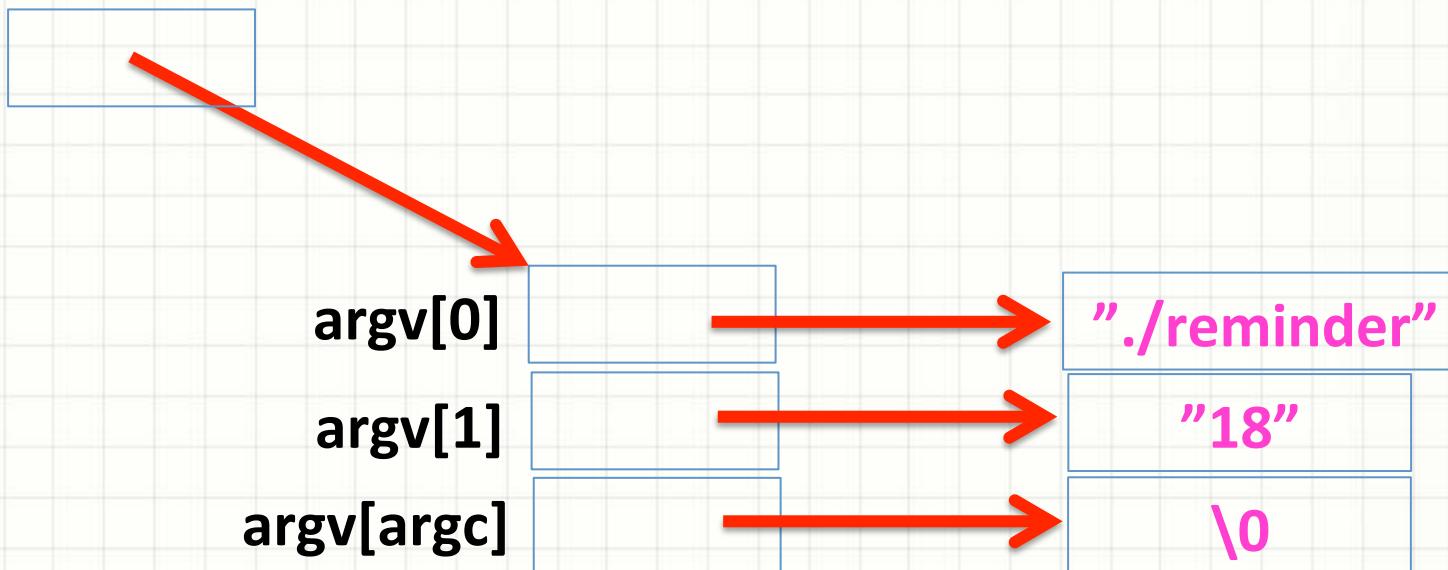
- primul argument, **argc** (*argumentul contor*) este o variabilă de tip întreg care reprezintă *numărul de parametri ai programului/comenzi*.
- al doilea argument, **argv** (*argumentul vector*) este un pointer la un tablou de pointeri la siruri de caractere, care conțin argumentele ( fiecare element al tabloului = un pointer la un sir de caractere = un argument).

# Preluarea argumentelor funcției main din linia de comandă

- primul argument, **argc** (*argumentul contor*)
- al doilea argument, **argv** (*argumentul vector*)
- **argv[0]** conține întotdeauna *numele programului*;
- **argv[1]** conține *primul parametru* ca sir de caractere;
- **argv[2]** conține *al doilea parametru* ca sir de caractere;
- **argv[argc-1]** conține *ultimul parametru* ca sir de caractere;
- **argv[argc]** va fi un pointer la un sir vid (**NULL**);
- pentru o comandă fără parametri **argc=1**, iar o comandă cu 2 parametri va avea **argc=3**.

# Preluarea argumentelor funcției main din linia de comandă

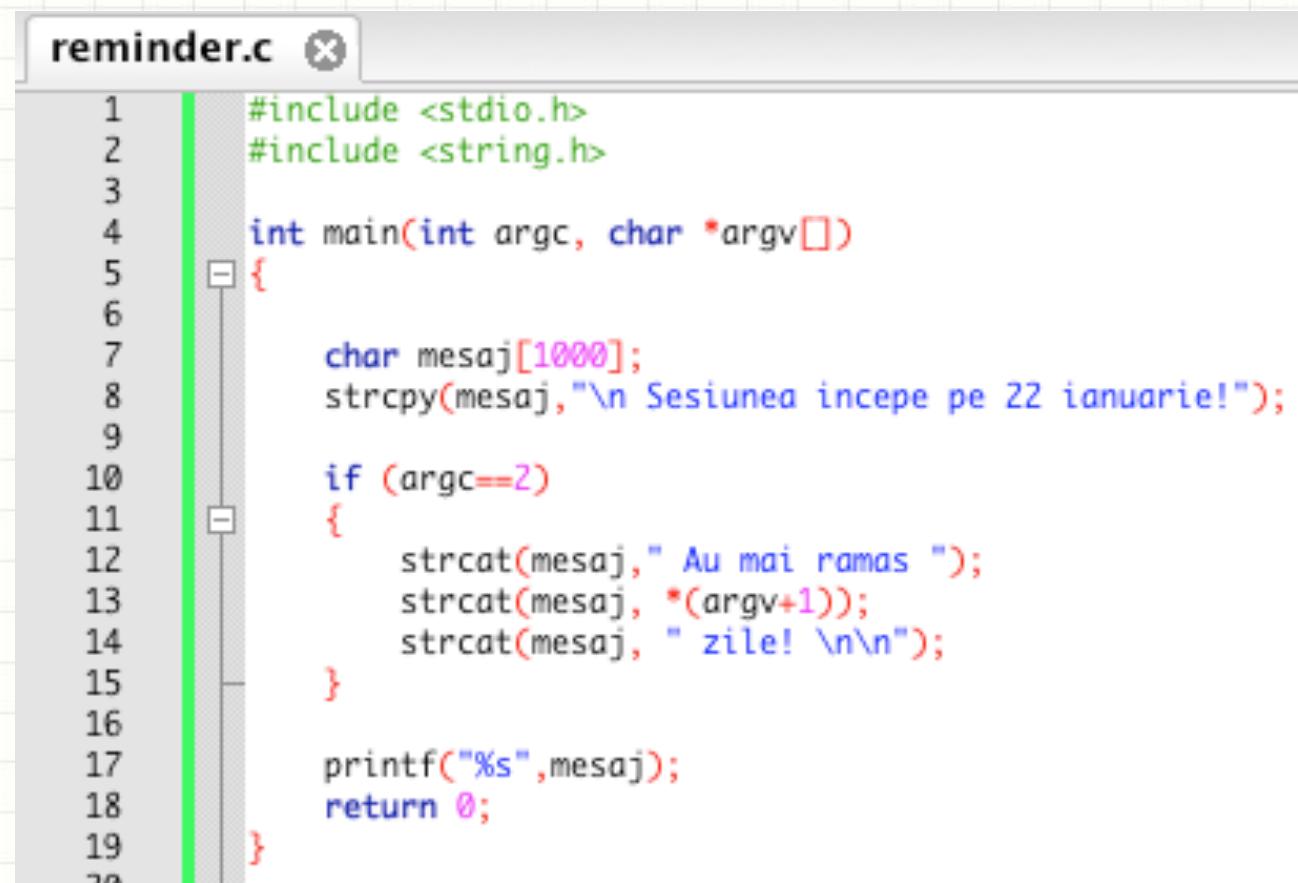
argv



```
Bogdan-Alexes-MacBook-Pro:curs11 bogdan$ ./reminder 18
```

Sesiunea incepe pe 22 ianuarie! Au mai ramas 18 zile!

# Preluarea argumentelor funcției main din linia de comandă



```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(int argc, char *argv[])
5 {
6
7     char mesaj[1000];
8     strcpy(mesaj, "\n Sesiunea incepe pe 22 ianuarie!");
9
10    if (argc==2)
11    {
12        strcat(mesaj, " Au mai ramas ");
13        strcat(mesaj, *(argv+1));
14        strcat(mesaj, " zile! \n\n");
15    }
16
17    printf("%s",mesaj);
18
19    return 0;
20 }
```

```
Bogdan-Alexes-MacBook-Pro:curs11 bogdan$ ./reminder 18
```

```
Sesiunea incepe pe 22 ianuarie! Au mai ramas 18 zile!
```

# Preluarea argumentelor funcției main din linia de comandă

```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5
6     char mesaj[1000];
7     strcpy(mesaj, "\n Sesiunea incepe pe 22 ianuarie!");
8     int i;
9
10    for (i=1;i<argc;i++)
11    {
12        strcat(mesaj, *(argv+i));
13        strcat(mesaj, " ");
14    }
15    strcat(mesaj, " \n\n");
16    printf("%s",mesaj);
17    return 0;
18 }
```

P/curs11/reminder2 "Serie 13 e pregatita?"

Sesiunea incepe pe 22 ianuarie! Serie 13 e pregatita?

Bogdan-Alexes-MacBook-Pro:~ bogdan\$

# Preluarea argumentelor funcției main din linia de comandă

- la problema cu triunghiuri dăm numele fișierului în main

```
-- 19 int main()
20 {
21     char numeFisier[] = "/Users/bogdan/Desktop/triunghi.txt";
22     triunghi *T = NULL;
23     int n = citesteTriunghiuri(numeFisier,&T);
24     afisareTriunghiuri(T,n);
25 }
```

- putem modifica astfel încât să citim numele fișierului din linia de comandă

```
triunghi_arg.c ✘
18
19 int main(int argc, char *argv[])
20 {
21
22     char numeFisier[1000];
23     if (argc==1)
24         strcpy(numeFisier,"/Users/bogdan/Desktop/triunghi.txt");
25     if (argc == 2)
26         strcpy(numeFisier,argv[1]);
27         |
28     triunghi *T = NULL;
29     int n = citesteTriunghiuri(numeFisier,&T);
30     afisareTriunghiuri(T,n);
31 }
```

# Preluarea argumentelor funcției main din linia de comandă

- putem modifica astfel încât să citim din linia de comandă

```
Bogdan-Alexes-MacBook-Pro:curs11 bogdan$ ./triunghi_arg /Users/bogdan/Desktop/triunghi.txt
*****
Punctul A are coordonatele (0.500000,0.500000)
Punctul B are coordonatele (1.500000,1.500000)
Punctul C are coordonatele (2.500000,2.500000)
Segmentul AB are lungimea 1.414214
Segmentul AC are lungimea 2.828427
Segmentul BC are lungimea 1.414214
Punctele sunt coliniare si nu formeaza un triunghi
*****
Triunghiul 1:
Punctul A are coordonatele (0.000000,-1.000000)
Punctul B are coordonatele (-1.000000,1.000000)
Punctul C are coordonatele (1.000000,1.000000)
Segmentul AB are lungimea 2.236068
Segmentul AC are lungimea 2.236068
Segmentul BC are lungimea 2.000000
Aria triunghiului este 2.000000
*****
Triunghiul 2:
Punctul A are coordonatele (2.000000,2.000000)
Punctul B are coordonatele (0.000000,0.000000)
Punctul C are coordonatele (0.500000,0.000000)
Segmentul AB are lungimea 2.828427
Segmentul AC are lungimea 2.500000
Segmentul BC are lungimea 0.500000
Aria triunghiului este 0.500000
Triunghiul de arie maxima are aria = 2.000000
Triunghiul contine punctele: (0.000000,-1.000000) (-1.000000,-1.000000) (1.000000,1.000000)
```

# Preluarea argumentelor funcției main din linia de comandă

- `char *argv[]` și `char **argv` sunt similare
- putem modifica programul în manieră similară astfel încât să citim numele fișierului din linia de comandă

```
18
19     int main(int argc, char **argv)
20 {
21
22     char numeFisier[1000];
23     if (argc==1)
24         strcpy(numeFisier, "/Users/bogdan/Desktop/triunghi.txt");
25     if (argc == 2)
26         strcpy(numeFisier, argv[1]);
27
28     triunghi *T = NULL;
29     int n = citesteTriunghiuri(numeFisier,&T);
30     afisareTriunghiuri(T,n);
31 }
```