

PROGRAMAREA CALCULATOARELOR

Andrei Patrascu

andrei.patrascu@fmi.unibuc.ro

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

Cursul 11

EXAMINARE

1. Examenul: sambata, 19 ianuarie 2019, orele 9:00 - 11:30.
2. Test de laborator 2: in locul laboratorului 13 (prima saptamana dupa vacanta). Exceptie: grupa 154 (foarte probabil marti de la 8)

PROGRAMA CURSULUI

□ Introducere

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



□ Fundamentele limbajului 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)

- 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 text și fișiere binare

- Funcții specifice de manipulare.

□ Structuri de date complexe și autoreferite

- Definire și utilizare

□ Funcții (2)

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

CURSUL DE AZI

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

CLASE DE ALOCARE/MEMORARE

- ❑ Într-un program C felul în care declarăm variabilele definește modul de alocare al acestora. Clasa de alocare a unei variabile definește următoarele caracteristici:
 - ❑ locul în memorie unde se rezervă spațiu pentru variabilă;
 - ❑ durata de viață;
 - ❑ vizibilitatea;
 - ❑ modalitatea de inițializare.
- ❑ clase de alocare:
 - ❑ **auto(matic)**
 - ❑ **register**
 - ❑ **static (intern)**
 - ❑ **static extern**

CLASA DE ALOCARE AUTO

- ❑ este implicită (variabile locale). Am discutat despre variabile locale în cursurile trecute;
- ❑ se specifică prin cuvântul cheie **auto**;
- ❑ în mod implicit toate variabilele locale sunt memorate în stivă;
- ❑ spațiul de memorie se alocă la execuție;
- ❑ variabilele automate sunt vizibile numai în corpul funcțiilor/instrucțiunilor compuse în care au fost declarate; la revenirea din execuția funcțiilor/instrucțiunilor compuse variabilele se elimină și stiva revine la starea dinaintea apelului;
- ❑ nu sunt inițializate;
- ❑ parametrii funcțiilor sunt implicit de clasă auto. Ei sunt transmiși, de asemenea, prin stivă (**de la dreapta la stânga!**).

```
int a,b,c;  
auto int d;
```

```
void f(int *x, int *y)  
{   auto int t; t=*x; *x=*y;*y=t; }
```

CLASA DE ALOCARE REGISTER

- se specifică prin cuvântul cheie **register**: se cere un acces rapid (registrul procesorului) la o variabilă. Nu se garantează că cererea va fi satisfăcută.
- numai parametri și variabilele automate de tipul int, char și pointer pot fi declarate ca variabile registru;
- **nu există adresă de memorie asociată**;
- nu puteți să manipulați tablouri de registre (aveți nevoie la dereferențiere de adresa elementului de început al tabloului)
- număr limitat de variabile în registre (compilatorul le trece pe cele pe care nu le poate alocă în clasa auto);
- parametri formali pot fi declarați în clasa register.

Exemplu:

```
register int i,s;  
for(i=0;i<n;i++)  
    s = s + i;
```

compilatoarele moderne nu au nevoie de asemenea declarații, ele reușesc să optimize codul mai bine decât am putea noi prin declararea variabilelor de tip register

CLASA DE ALOCARE STATIC INTERN

- se specifică prin cuvântul cheie **static**;
- desemnează variabile cu adrese de memorie constantă, adresa e fixă pe durata executării programului;
- se alocă de compilator în zone speciale (zona de date a programului);
- variabilele globale sunt implicit din clasa static;
- variabilele din clasa static se inițializează numai la primul apel (prin codul generat!).

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void adunare(int x){
5      int static y=25;
6      y+=x;
7      printf("y=%d\n",y);
8  }
9
10 int main(){
11     adunare(1);
12     adunare(2);
13     adunare(3);
14     return 0;
15 }
```

```
y=26
y=28
y=31
```

```
Process returned 0 (0x0)    execution time : 0.004 s
Press ENTER to continue.
```


CLASA DE ALOCARE STATIC INTERN

- variabilele din clasa static nu sunt globale, sunt vizibile numai în funcțiile/ blocurile de funcții în care au fost declarate
- se folosesc când scriem funcții recursive să numărăm câte apeluri generează o funcție

```
fibonacci.c x
1  #include<stdio.h>
2
3  int fib(int x)
4  {
5      int static nrApeluri = 0;
6      nrApeluri = nrApeluri + 1;
7      printf("Apelul nr %d \n",nrApeluri);
8      if (x==0 || x== 1)
9          return x;
10     return fib(x-1) + fib(x-2);
11 }
12
13 int main()
14 {
15     fib(20);
16     return 0;
17 }
```

```
Apelul nr 21878
Apelul nr 21879
Apelul nr 21880
Apelul nr 21881
Apelul nr 21882
Apelul nr 21883
Apelul nr 21884
Apelul nr 21885
Apelul nr 21886
Apelul nr 21887
Apelul nr 21888
Apelul nr 21889
Apelul nr 21890
Apelul nr 21891
```

CLASA DE ALOCARE STATIC EXTERN

- se specifică prin cuvântul cheie **extern**;
- o variabilă de tip extern este o variabilă definită într-un alt fișier sursă (extern);
- se alocă în funcție de modul de declarare din fișierul sursă.

Exemplu:

```
//fisier1.c
```

```
extern int i; //declara variabila i ca fiind definită in alt  
fisier
```

```
//fisier2.c
```

```
int i = 5; //variabila i este definită aici
```

O variabilă poate fi declarată în mai multe fișiere (cu extern), dar trebuie definită într-un singur fișier!

CLASA DE ALOCARE STATIC EXTERN

exempluExtern.c

```
1  #include<stdio.h>
2
3  int x = 1;
4
5  void f()
6  {
7      int x = 7;
8      printf("%d \n",x);
9  }
10
11 int main()
12 {
13     f();
14     return 0;
15 }
```

Afiseaza 7 !

exempluExtern.c

```
1  #include<stdio.h>
2
3  int x = 1;
4
5  void f()
6  {
7      int x = 7;
8      {
9          extern int x;
10         printf("%d \n",x);
11     }
12 }
13
14 int main()
15 {
16     f();
17     return 0;
18 }
```

Se acceseaza variabila globala x = 1!

CURSUL DE AZI

1. Clase de memorare
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

```
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

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

student.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 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 }
```

```
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 }
```

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

Reader: Alexei, Marius, Popescu, Dan

TRANSMITEREA STRUCTURILOR CA PARAMETRI

- ❑ când o structura este transmisă ca parametru unei functii se face o copie a zonei de memorie respective
- ❑ transmitere prin valoare
- ❑ la ieșirea din funcție se distruge 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 x
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 aceleași 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 mediiIntrare;  
    struct adresa;  
}
```

Structura adresa trebuie să fie definită în prealabil

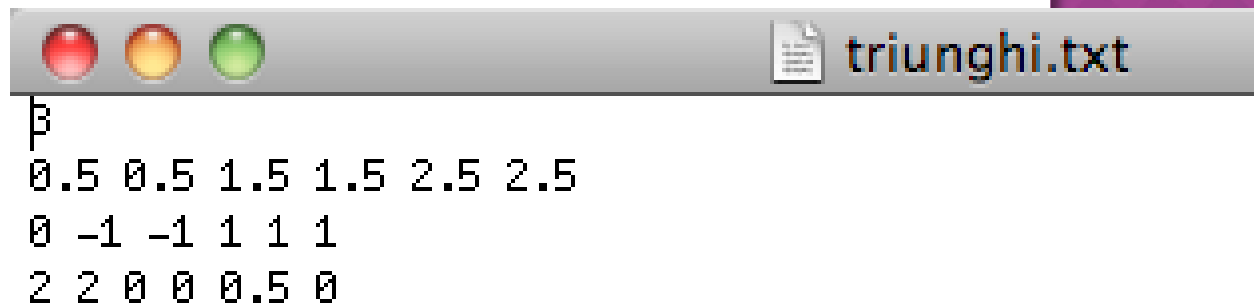
- ❑ tablou de structuri: tablou cu elemente de tip struct
`struct student grupa[30];`

STRUCTURI IMBRICATE ȘI TABLOURI DE STRUCTURI

- fișierul text *triunghi.txt* conține 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 ordonata 2 abscisa 3
ordonata 3

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.



```
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* conține 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 ordonata 2 abscisa 3 ordonata 3
(x_1 , y_1) (x_2 , y_2) (x_3 , y_3)

Cerinta: 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.

```
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

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);
18
```

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 %f", &A.abscisa, &A.ordonata, &B.abscisa,
61             &B.ordonata, &C.abscisa, &C.ordonata);
62         float AB = sqrt(pow(A.abscisa - B.abscisa, 2) + pow(A.ordonata - B.ordonata, 2));
63         float AC = sqrt(pow(A.abscisa - C.abscisa, 2) + pow(A.ordonata - C.ordonata, 2));
64         float BC = sqrt(pow(B.abscisa - C.abscisa, 2) + pow(B.ordonata - C.ordonata, 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].A.ordonata);
90         printf("Punctul B are coordonatele (%f,%f) \n",pT[i].B.abscisa,pT[i].B.ordonata);
91         printf("Punctul C are coordonatele (%f,%f) \n",pT[i].C.abscisa,pT[i].C.ordonata);
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
96         if(!pT[i].triunghiValid)
97             printf("Punctele sunt coliniare si nu formeaza un triunghi\n");
98         if(pT[i].triunghiValid)
99             printf("Aria triunghiului este %f \n");
100     }
101 }
```

STRUCTURI IMBRICATE ȘI TABLOURI DE STRUCTURI

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

int main()
{
    char nume_fisier[]="arie_triunghi.txt";
    triunghi *T = NULL;
    int n = citesteTriunghiuri(nume_fisier,&T);
    afisareTriunghiuri(T,n);

    int i; float arieMaxima = 0; int indiceTriunghi = -1;

    for (i=0;i<n;i++)
    {
        if (T[i].triunghiValid)
            if (arieMaxima < T[i].arie)
                {arieMaxima = T[i].arie;indiceTriunghi = i;}
    }
    if (arieMaxima)
    {
        printf("Triunghi de arie maxima are aria = %f \n",T[indiceTriunghi].arie);
        printf("Triunghiul contine punctele: (%f,%f) (%f,%f) (%f,%f)",
            T[indiceTriunghi].A.abscisa, T[indiceTriunghi].A.ordonata,
            T[indiceTriunghi].B.abscisa, T[indiceTriunghi].B.ordonata,
            T[indiceTriunghi].C.abscisa, T[indiceTriunghi].B.ordonata);
    }
    else printf("Nu exista! \n");
    return 0;
}
```

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;  
}
```

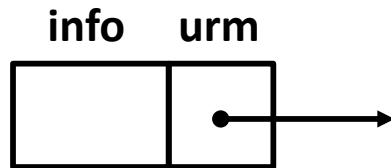
declaratie invalida

STRUCTURI AUTOREFERITE

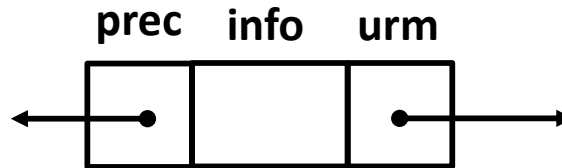
- aplicații pentru structuri de date în alocare dinamică

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

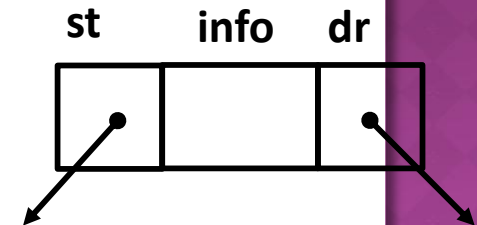
declarație unui
structuri nod



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



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



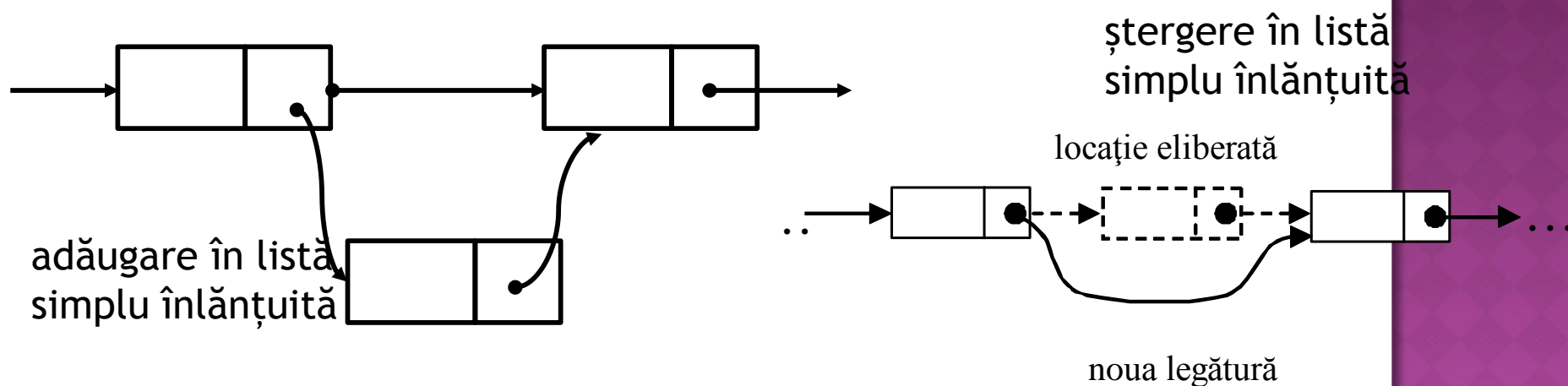
- fiecare nod conține:
 - un câmp/mai multe câmpuri cu informația nodului - info
 - un pointer/mai mulți pointeri către nodurile vecine:
următor, precedent, fiuStang, fiuDrept

STRUCTURI AUTOREFERITE

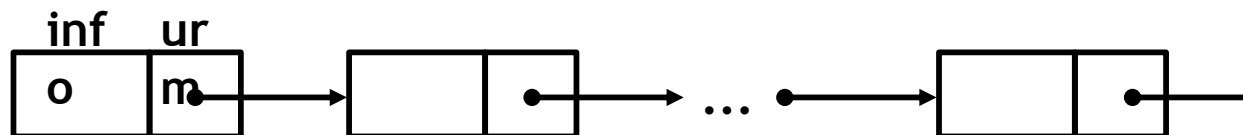
- ❑ aplicații pentru structuri de date în alocare dinamică
- ❑ liste, stive, cozi, arbori
 - ❑ avantaj față de implementarea statică
 - ❑ operațiile de adaugare sau stergere sunt foarte rapide
 - ❑ dezavantaj față de implementarea statică :
 - ❑ accesul la un nod se face prin parcurgerea nodurilor precedente
 - ❑ 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 înlănțuite 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 nenule din ambii vectori

```
struct nod{  
    float info;  
    int poz;  
    struct nod *urm;  
}
```

declarație a structurii nod
folosită la reprezentarea listei

STRUCTURI AUTOREFERITE

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

```
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 construieșteLista(nod**, nod**, char*);
13 void suma(nod*, nod*, nod**, nod**);
14 float produsScalar(nod*, nod*);
15
```

vector1.txt

1 5
10 20
40 50

vector2.txt

1 8.1
15 5.3
36 10

STRUCTURI AUTOREFERITE

- operații cu vectori rari: suma și produsul scalar

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char* nume_fisiere = { "vector1.txt", "vector2.txt" };

    nod* prim1 = NULL; nod* ultim = NULL;
    construiesLista(&prim1, &ultim1, nume_fisiere[0]);
    afiseazaLista(prim1);

    nod* prim2 = NULL; nod* ultim2 = NULL;
    construiesLista(&prim2, &ultim2, nume_fisiere[1]);
    afiseazaLista(prim2);

    nod* prim3 = NULL; nod* ultim3 = NULL;
    suma(prim1, prim2, &prim3, &ultim3);
    afiseazaLista(prim3);
    printf("*****\n");
    printf("Produsul scalar a celor 2 vectori: %f \n", produsScalar(prim1, prim2));

    return 0;
}
```

STRUCTURI AUTOREFERITE

- operații cu vectori rari: suma și produsul scalar

```
82
83 void construiesLista(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     return;
102 }
```

```
126 void afiseazaLista(nod *p)
127 {
128     printf("*****\n");
129     while(p)
130     {
131         printf("%d %f \n", p->poz, p->info);
132         p = p->urm;
133     }
134 }
```

STRUCTURI AUTOREFERITE

- operații cu vectori rari: suma și produsul scalar

```
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 }
```


STRUCTURI AUTOREFERITE

- operații cu vectori rari: suma și produsul scalar

```
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         else //(p1->poz == p2->poz)
58         {
59             adaugare(prim3, ultim3, p1 -> info + p2 -> info, p2 -> poz);
60             p1 = p1 -> urm; p2 = p2 -> urm;
61         }
62     }
63 }
```

STRUCTURI AUTOREFERITE

- operații cu vectori rari: suma și produsul scalar

```
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

```
*****
```

```
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. Clase de memorare
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 **afisare** care afișează la terminal un mesaj:
- ❑ program fără parametri, lansat în execuție prin:
>./afisare

PRELUAREA ARGUMENTELOR FUNCȚIEI MAIN DIN LINIA DE COMANDĂ

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

PRELUAREA ARGUMENTELOR FUNCȚIEI MAIN DIN LINIA DE COMANDĂ

- un program poate avea și parametri, care apar după numele comenzii 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[])  
{ ... }
```

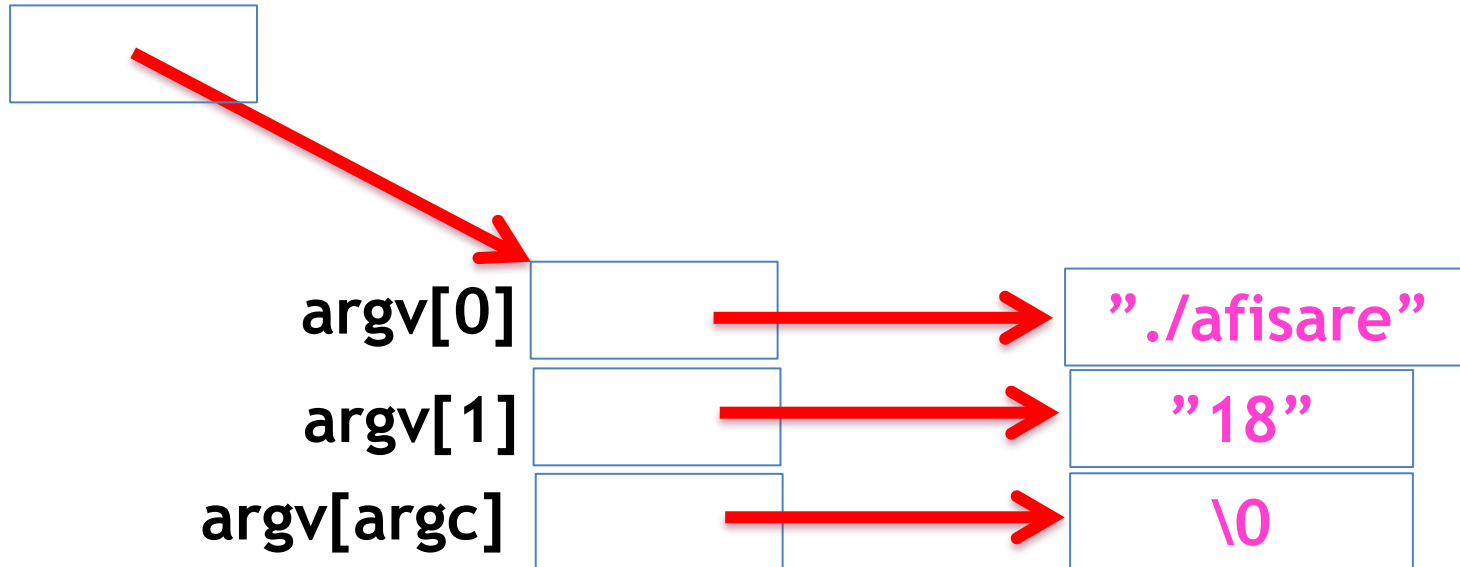
- primul argument, **argc** (*argumentul contor*) este o variabilă de tip întreg care reprezintă *numărul de parametri ai programului/comenzii*.
- al doilea argument, **argv** (*argumentul vector*) este un pointer la un tablou de pointeri la șiruri de caractere, care conțin argumentele (fiecare element al tabloului = un pointer la un șir 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 șir de caractere;
- ❑ **argv[2]** conține *al doilea parametru* ca șir de caractere;
- ❑ **argv[argc-1]** conține *ultimul parametru* ca șir de caractere;
- ❑ **argv[argc]** va fi un pointer la un șir 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



PRELUAREA ARGUMENTELOR FUNCȚIEI MAIN DIN LINIA DE COMANDĂ

reminder.c

```
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     return 0;
19 }
```

FUNCTIEI 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 "Seria 13 e pregatita?"

Sesiunea incepe pe 22 ianuarie!Seria 13 e pregatita?

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

FUNCTIEI 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);
```

- 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
```

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)
```

FUNCTIEI MAIN DIN LINIA DE COMANDĂ

- ❑ `char *argv[]` si `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
32     . . .
```