

Programarea calculatoarelor

FMI

Secția Calculatoare și tehnologia informației, anul I

Cursul 11 / 11.12.2023

Programa cursului

□ Introducere

- Algoritmi
- Limbaje de programare.

□ Fundamentele limbajului C

- Introducere în limbajul C. Structura unui program C.
- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Tipuri derivate de date: pointeri, tablouri, șiruri de caractere, structuri, uniuni, câmpuri de biți, enumerări
- Instrucțiuni de control
- Directive de preprocesare. Macrodefiniții.
- Funcții de citire/scriere.
- Etapele realizării unui program C.

□ Fișiere text

- Funcții specifice de manipulare.

□ Funcții (1)

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

□ Tablouri și pointeri

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

□ Șiruri de caractere

- Funcții specifice de manipulare

□ Fișiere binare

- Funcții specifice de manipulare

□ Structuri de date complexe și autoreferite

- **Definire și utilizare**

□ Funcții (2)

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

Cuprinsul cursului de azi

- 1. Structuri de date compexe și autoreferite**
2. Funcții cu număr variabil de argumente

Structuri

❑ 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;
```

Obs. 1:

- ✓ 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.

Obs. 2: dacă <nume> este prezent → se pot declara noi variabile de tip structura:

```
struct <nume> <lista noilor identificatori>;
```

Obs. 3: 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 medie_admitere;  
} 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 medie_admitere;  
} 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 variabila **A** de tip student*/*

Structuri. Inițializare

main.c

```
1  #include <stdio.h>
2  typedef struct{
3      char nume[30];
4      char prenume[30];
5      float medie_admitere;
6  } student;
7
8  int main()
9  {
10     student A={"Popescu","Ion", 9.25};
11     student B={.medie_admitere=9.25,.prenume="Dana",.nume="Ionescu"};
12     student C={"Popescu"};
13
14     printf("%s %s %.2f\n", A.nume, A.prenume, A.medie_admitere);
15     printf("%s %s %.2f\n", B.nume, B.prenume, B.medie_admitere);
16     printf("%s %s %.2f\n", C.nume, C.prenume, C.medie_admitere);
17
18     return 0;
19 }
```

```
Popescu Ion 9.25
Ionescu Dana 9.25
Popescu 0.00
```

Transmiterea structurilor ca parametri

main.c

Popescu Ion 9.25

?

```
1  #include <stdio.h>
2
3  typedef struct{
4      char nume[30];
5      char prenume[30];
6      float medie_admitere;
7  } student;
8
9  void adauga_un_punct(student x)
10 {
11     x.medie_admitere++;
12     if(x.medie_admitere>10)
13         x.medie_admitere=10;
14 }
15
16 int main()
17 {
18     student A={"Popescu","Ion", 9.25};
19
20     adauga_un_punct(A);
21
22     printf("%s %s %.2f\n", A.nume, A.prenume, A.medie_admitere);
23
24     return 0;
25 }
```


Transmiterea structurilor ca parametri

- ❑ când o structura este transmisă ca parametru unei funcții 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
- ❑ Soluția: **pointeri la structuri**
- ❑ folosim operatorul `->` pentru a accesa câmpurile

Pointeri la structuri

main.c

Popescu Ion 10.00

```
1  #include <stdio.h>
2
3  typedef struct{
4      char nume[30];
5      char prenume[30];
6      float medie_admitere;
7  } student;
8
9  void adauga_un_punct(student *x)
10 {
11     x->medie_admitere++;
12     if(x->medie_admitere>10)
13         x->medie_admitere=10;
14 }
15
16 int main()
17 {
18     student A={"Popescu","Ion", 9.25};
19
20     adauga_un_punct(&A);
21
22     printf("%s %s %.2f\n", A.nume, A.prenume, A.medie_admitere);
23
24     return 0;
25 }
```

Pointeri la structuri

- ❑ folosim operatorul -> pentru a accesa câmpurile
- ❑ respectă aceleași reguli ca și ceilalți pointeri
- ❑ trebuie să fie inițializați
- ❑ trebuie să se facă conversii explicite când este cazul
- ❑ pointeri la structuri vs. structuri care conțin pointeri

Structuri imbricate și tablouri de structuri

- o structură imbricată (nested) = o structură care conține ca membru o altă structură
- tablou de structuri = tablou cu elemente de tip struct

```
struct student{  
    char nume[30];  
    char prenume[30];  
    float medie_admitere;  
    struct adresa; /*Structura adresa trebuie să fie definită în prealabil*/  
}  
struct student grupa[30]; /*tablou de structuri*/
```

Structuri imbricate și tablouri de structuri

□ Enunț:

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 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.

Structuri imbricate și tablouri de structuri

```
main.c | triunghiuri.txt |
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  typedef struct
6  {
7      float abscisa;
8      float ordonata;
9  } punct2d;
10
11  typedef struct
12  {
13      punct2d A, B, C;
14      float AB, AC, BC;
15      int triunghi_valid;
16      float perimetru;
17      float arie;
18  } triunghi;
19
20  int citeste_triunghiuri (char *, triunghi **);
21  int afisare_triunghiuri (triunghi *, int);
22  void afisare_arie_maxima(triunghi *pT, int n);
23
24  int main ()
25  {
26      char nume_fisier[]="triunghiuri.txt";
27      triunghi *T=NULL;
28      int n=citeste_triunghiuri(nume_fisier, &T);
29      afisare_triunghiuri(T,n);
30      afisare_arie_maxima(T,n);
31      return 0;
32  }
```

```
33
34 int citeste_triunghiuri (char *nume, triunghi **pT)
35 {
36     FILE *f = fopen (nume, "r");
37     if (f == NULL)
38         {printf ("Eroare la deschiderea fisierului\n"); exit (0);}
39     int i, n;
40     fscanf (f, "%d", &n);
41     triunghi *p = (triunghi *) malloc (n * sizeof (triunghi));
42     if (p == NULL)
43         {printf ("Eroare la alocare\n"); exit (0);}
44
45     punct2d A, B, C;
46     for (i = 0; i < n; i++)
47     {
48         fscanf (f, "%f %f %f %f %f %f", &A.abscisa, &A.ordonata, &B.abscisa, &B.ordonata,
49             &C.abscisa, &C.ordonata);
50         float AB = sqrt (pow (A.abscisa - B.abscisa, 2) + pow (A.ordonata - B.ordonata, 2));
51         float AC = sqrt (pow (A.abscisa - C.abscisa, 2) + pow (A.ordonata - C.ordonata, 2));
52         float BC = sqrt (pow (B.abscisa - C.abscisa, 2) + pow (B.ordonata - C.ordonata, 2));
53
54         p[i].A=A; p[i].B=B; p[i].C=C; p[i].AB=AB; p[i].AC=AC; p[i].BC=BC;
55
56         if (AB + BC == AC || AB + AC == BC || AC + BC == AC) p[i].triunghi_valid = 0;
57         else
58         {
59             p[i].triunghi_valid = 1;
60             p[i].perimetru = AB + AC + BC;
61             float sp = p[i].perimetru / 2;
62             p[i].arie = sqrt (abs(sp * (sp - p[i].AB) * (sp - p[i].AC) * (sp - p[i].BC)));
63         }
64     }
65     *pT = p;
66     fclose (f);
67     return n;
68 }
```

```
70 int afisare_triunghiuri (triunghi *pT, int n)
71 {
72     for (int i = 0; i < n; i++)
73     {
74         printf("\n");
75         if (pT[i].triunghi_valid)
76             printf("triunghiul %d: \n", i);
77
78         printf("punctul A are coordonatele (%.2f, %.2f)\n", pT[i].A.abscisa,
79             pT[i].A.ordonata);
80         printf("punctul B are coordonatele (%.2f, %.2f)\n", pT[i].B.abscisa,
81             pT[i].B.ordonata);
82         printf("punctul C are coordonatele (%.2f, %.2f)\n", pT[i].C.abscisa,
83             pT[i].C.ordonata);
84         printf("Segmentul AB are lungimea: %.2f\n", pT[i].AB);
85         printf("Segmentul AC are lungimea: %.2f\n", pT[i].AC);
86         printf("Segmentul BC are lungimea: %.2f\n", pT[i].BC);
87         if (pT[i].triunghi_valid)
88             printf("Aria triunghiului este: %.2f\n", pT[i].arie);
89         else
90             printf("Punctele sunt coliniare: nu formeaza triunghi.\n");
91     }
92     return n;
93 }
94
95 void afisare_arie_maxima(triunghi *pT, int n)
96 {
97     float arie_maxima=0; int nr=0,i,j=0;
98     for(i=0; i<n;i++)
99     if (pT[i].arie)
100     {
101         j++;
102         if(pT[i].arie>arie_maxima)
103             {arie_maxima=pT[i].arie;nr=j;}
104     }
105     printf("Triunghiul %d are aria maxima = %.2f",nr,arie_maxima);
106 }
```


main.c

triunghiuri.txt

```
1 5
2 0.5 2 2 3.5 2 5
3 1 1 2 2 3 3
4 0.5 0 3 3 5 0.5
5 12 10 3 4 2 6
6 2.5 3 2 4 2 6
```

```
punctul A are coordonatele (0.50, 2.00)
punctul B are coordonatele (2.00, 3.50)
punctul C are coordonatele (2.00, 5.00)
Segmentul AB are lungimea: 2.12
Segmentul AC are lungimea: 3.35
Segmentul BC are lungimea: 0.00
Punctele sunt coliniare: nu formeaza triunghi.
```

```
triunghiul 1:
punctul A are coordonatele (1.00, 1.00)
punctul B are coordonatele (2.00, 2.00)
punctul C are coordonatele (3.00, 3.00)
Segmentul AB are lungimea: 1.41
Segmentul AC are lungimea: 2.83
Segmentul BC are lungimea: 1.00
Aria triunghiului este: 1.00
```

```
triunghiul 2:
punctul A are coordonatele (0.50, 0.00)
punctul B are coordonatele (3.00, 3.00)
punctul C are coordonatele (5.00, 0.50)
Segmentul AB are lungimea: 3.91
Segmentul AC are lungimea: 4.53
Segmentul BC are lungimea: 2.00
Aria triunghiului este: 3.87
```

```
triunghiul 3:
punctul A are coordonatele (12.00, 10.00)
punctul B are coordonatele (3.00, 4.00)
punctul C are coordonatele (2.00, 6.00)
Segmentul AB are lungimea: 10.82
Segmentul AC are lungimea: 10.77
Segmentul BC are lungimea: 1.00
Aria triunghiului este: 5.29
```

```
punctul A are coordonatele (2.50, 3.00)
punctul B are coordonatele (2.00, 4.00)
punctul C are coordonatele (2.00, 6.00)
Segmentul AB are lungimea: 1.12
Segmentul AC are lungimea: 3.04
Segmentul BC are lungimea: 0.00
Punctele sunt coliniare: nu formeaza triunghi.
Triunghiul 3 are aria maxima = 5.29
```

Sortarea unui tablou de structuri

```
main.c | triunghiuri.txt
11 typedef struct
12 {
13     punct2d A, B, C;
14     float AB, AC, BC;
15     int triunghi_valid;
16     float perimetru;
17     float arie;
18 } triunghi;
19
20 int citeste_triunghiuri(char *, triunghi **);
21 int afisare_triunghiuri(triunghi *, int);
22 void afisare_arie_maxima(triunghi *, int);
23 void sorteaza_triunghiuri(triunghi *, int);
24
25 int main ()
26 {
27     char nume_fisier[]="triunghiuri.txt";
28     triunghi *T=NULL;
29     int n=citeste_triunghiuri(nume_fisier, &T);
30     afisare_triunghiuri(T,n);
31     //afisare_arie_maxima(T,n);
32     sorteaza_triunghiuri(T,n);
33     afisare_triunghiuri(T,n);
34     return 0;
35 }
```

```
main.c | triunghiuri.txt
111 void sorteaza_triunghiuri(triunghi *pT, int n)
112 {
113     int i,j;
114     triunghi aux;
115
116     for(i=0; i<n; i++)
117         if(!pT[i].triunghi_valid)
118             pT[i].arie=0;
119
120     for(i=0; i<n-1; i++)
121         for(j=i; j<n; j++)
122             if(pT[i].arie>pT[j].arie)
123             {
124                 aux=pT[i];
125                 pT[i]=pT[j];
126                 pT[j]=aux;
127             }
128 }
129
```

Structuri autoreferite

- ❑ structuri autoreferite = structuri care conțin o declarație recursivă pentru anumiți membri de tip pointer

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

declarație validă

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

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

declarație validă: structurile S și T se invocă reciproc

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

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

Declarație invalidă

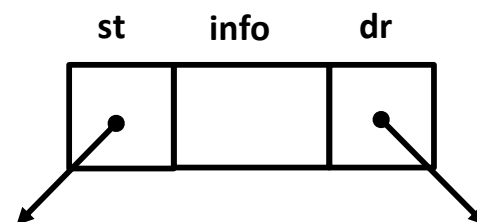
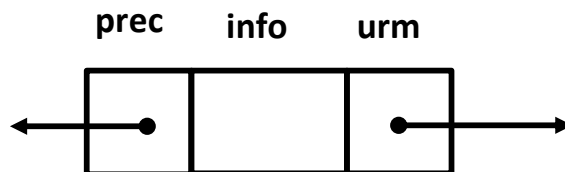
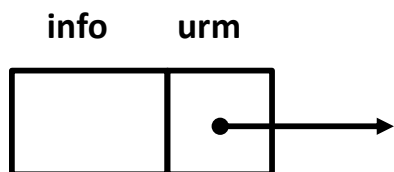
Structuri autoreferite

- aplicații pentru structuri de date în alocare dinamică

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

```
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/predecesor, fiuStang, fiuDrept

Structuri autoreferite

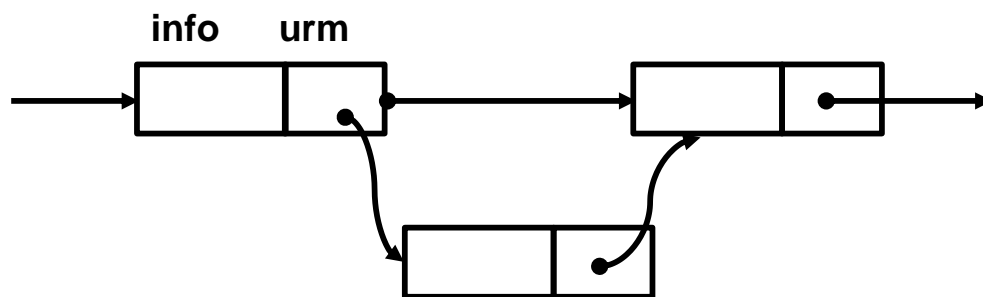
- ❑ aplicații pentru structuri de date în alocare dinamică
- ❑ **liste, stive, cozi, arbori**
 - ❑ **avantaj** față de implementarea statică
 - ❑ operațiile de adăugare sau ștergere sunt foarte rapide
 - ❑ **dezavantaj** față de implementarea statică :
 - ❑ accesul la un nod se face prin parcurgerea nodurilor precedente
 - ❑ adresa nodurilor vecine ocupă memorie suplimentară

Structuri autoreferite

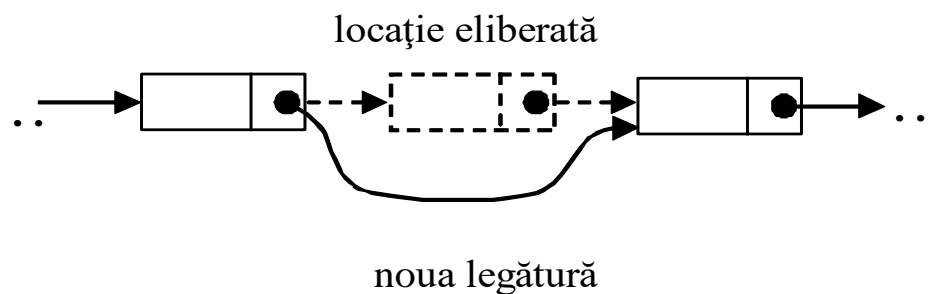
□ aplicații pentru structuri de date în alocare dinamică

□ **Operații specifice:**

□ adăugare

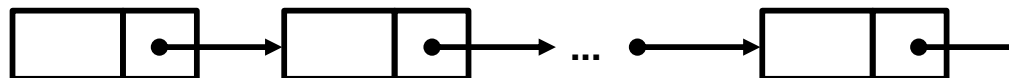


□ ștergere



□ traversare

□ căutare



Exemple: adăugare, ștergere, 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
 - ❑ multe dintre elementele vectorului egale cu 0
 - ❑ reprezentare eficientă: liste simplu înlănțuite alocate dinamic
 - ❑ fiecare nod din lista reține:
 - ❑ valoarea
 - ❑ poziția 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

```
main.c  vector1.txt  vector2.txt
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  typedef struct{
8      float info;
9      int poz;
10     struct nod *urm;
11 }nod;
12
13 void adaugare(nod**, nod**, float, int);
14 void construire_lista(nod**, nod**, char*);
15 void afisare_lista(nod*);
16 void suma(nod*, nod*, nod**, nod**);
17 float produs_sclar(nod*, nod*);
18
19 int main()
20 {
21     char* nume_fisier1="vector1.txt";
22     char* nume_fisier2="vector2.txt";
23     /*atentie: pt un spatiu in plus la finalul fisierului
24     dubleaza ultimul nod*/
25
```


Structuri autoreferite

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

```
main.c  vector1.txt  vector2.txt
19  int main()
20  {
21      char* nume_fisier1="vector1.txt";
22      char* nume_fisier2="vector2.txt";
23      /*atentie: pt un spatiu in plus la finalul fisierului
24      dubleaza ultimul nod*/
25
26      nod* prim1=NULL; nod* ultim1=NULL;
27      construire_lista(&prim1,&ultim1,nume_fisier1);
28      afisare_lista(prim1);
29
30      nod* prim2=NULL; nod* ultim2=NULL;
31      construire_lista(&prim2,&ultim2,nume_fisier2);
32      afisare_lista(prim2);
33
34      nod* prim3=NULL; nod* ultim3=NULL;
35      suma(prim1,prim2, &prim3, &ultim3);
36      afisare_lista(prim3);
37
38      printf("\n%.2f",produs_scalar(prim1,prim2));
39
40      return 0;
41  }
```

Structuri autoreferite

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

```
main.c  vector1.txt  vector2.txt
43 void construire_lista(nod** p, nod** u, char* nume_fisier)
44 {
45     FILE* f=fopen(nume_fisier, "r");
46     if (f==NULL)
47         {printf("Eroare la deschidere fisier\n");exit(0);}
48     int poz; float val;
49     while(!feof(f))
50     {
51         fscanf(f,"%d %f",&poz, &val);
52         adaugare(p, u, val, poz);
53     }
54 }
55
56 void afisare_lista(nod* p)
57 {
58     printf("\n");
59     while(p)
60     {
61         printf("%d %.2f\n", p->poz, p->info);
62         p=p->urm;
63     }
64 }
65
66 void adaugare(nod** p, nod** u, float val, int poz)
```

Structuri autoreferite

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

```
main.c  vector1.txt  vector2.txt
66 void adaugare(nod** p, nod** u, float val, int poz)
67 {
68     if(*p==NULL)
69     {
70         *p=(nod*)malloc(sizeof(nod));
71         (*p)->info=val;
72         (*p)->urm=NULL;
73         (*p)->poz=poz;
74         *u=*p;
75     }
76     else
77     {
78         nod* c=(nod*)malloc(sizeof(nod));
79         c->info=val;
80         c->urm=NULL;
81         c->poz=poz;
82         (*u)->urm=c;
83         *u=c;
84     }
85 }
86
```

Structuri autoreferite

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

```
main.c  vector1.txt  vector2.txt
87 void suma(nod* prim1, nod* prim2, nod** prim3, nod** ultim3)
88 {
89     nod *p1,*p2; p1=prim1; p2=prim2;
90     while(p1!=NULL && p2!=NULL)
91     {
92         if(p1->poz < p2->poz)
93         {   adaugare(prim3,ultim3,p1->info,p1->poz); p1=p1->urm;
94         }
95         else
96             if(p1->poz > p2->poz)
97             {   adaugare(prim3,ultim3,p2->info,p2->poz); p2=p2->urm;
98             }
99         else //if(p1->poz == p2->poz)
100             {   adaugare(prim3,ultim3,p1->info + p2->info,p2->poz);
101                 p1=p1->urm;p2=p2->urm;
102             }
103     }
104     while(p1!=NULL)
105     {   adaugare(prim3,ultim3,p1->info,p1->poz); p1=p1->urm;
106     }
107     while(p2!=NULL)
108     {   adaugare(prim3,ultim3,p2->info,p2->poz); p2=p2->urm;
109     }
110 }
```

Structuri autoreferite

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

```
main.c  vector1.txt  ⋮  vector2.txt  ⋮
112  float produs_scalar(nod* prim1,nod* prim2)
113  {
114      float r=0;
115      nod *p1,*p2; p1=prim1; p2=prim2;
116      while(p1!=NULL && p2!=NULL)
117      {
118          if(p1->poz == p2->poz)
119          {
120              r+=(p1->info)*(p2->info);
121              p1=p1->urm;
122              p2=p2->urm;
123          }
124          else
125              if(p1->poz < p2->poz)
126                  p1=p1->urm;
127              else
128                  if(p1->poz > p2->poz)
129                      p2=p2->urm;
130      }
131      return r;
132  }
133
```

Structuri autoreferite

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

vector1.txt

1 5

10 20

40 50

80 23.45

vector2.txt

1 8.1

15 5.3

36 10

```
1 5.00
10 20.00
40 50.00
80 23.45
```

```
1 8.10
15 5.30
36 10.00
```

```
1 13.10
10 20.00
15 5.30
36 10.00
40 50.00
80 23.45
```

```
40.50
```

Cuprinsul cursului de azi

1. Structuri de date compexe și autoreferite
2. Funcții cu număr variabil de argumente

Funcții cu număr variabil de argumente

□ funcții cu număr variabil de argumente utilizate de către voi până acum:

□ printf, fprintf

□ scanf, fscanf

```
int a, b, c;  
scanf("%d %d",&a,&b);  
printf("a=%d\nb=%d\n",a,b);  
scanf("%d",&c);  
printf("c=%d\n",c);  
printf("\n\n");
```


Funcții cu număr variabil de argumente

❑ probleme:

- ❑ nu se cunoaște numărul parametrilor funcției
- ❑ nu se cunoaște tipul parametrilor

❑ header-ul `stdarg.h` cuprinde:

- ❑ definiția unui tip de date specializat (`va_list`) dedicat manipulării listelor cu număr variabil de parametri
- ❑ macro-uri (`va_start`, `va_arg`, `va_end`) care realizează operații pe acest tip de date
- ❑ macrou = fragment de cod căruia i se asociază un nume, la preprocesare se înlocuiește numele cu fragmentul de cod.

Cursul 4: Constante simbolice și macro-uri

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

```
//constante simbolice
```

```
#define ALPHA 30
```

```
#define BETA ALPHA+10
```

```
#define GAMMA (ALPHA+10)
```

```
//macro-uri
```

```
#define MIN(a,b) (((a)<(b))?(a):(b))
```

```
#define ABS1(x) (x<0)?-x:x
```

```
#define ABS2(x) (((x)<0)?-(x):(x))
```

```
#define INTER(tip,a,b) \  
    {tip c; c=a; a=b; b=c;}
```

```
int main()
```

```
{
```

```
    int x=2*BETA;
```

```
    int y=2*GAMMA;
```

```
    printf("%d %d\n",x,y); //70 80
```

```
    int m=MIN(x,y);
```

```
    printf("%d\n",m); //70
```

```
    int a=ABS1(x-y);
```

```
    int b=ABS2(x-y);
```

```
    printf("%d %d\n",a,b); //-150 10
```

```
    INTER(int,a,b);
```

```
    printf("%d %d\n",a,b); //10 -150
```

```
    INTER(int,a,b);
```

```
    printf("%d %d\n",a,b); //-150 10
```

```
    return 0;
```

```
}
```

Funcții cu număr variabil de argumente

□sintaxa:

tip_rezultat nume_functie(tip_argument nume_argument, ...)

unde (*cel puțin*) primul argument este întotdeauna fix, vizibil,
restul argumentelor fiind declarate prin *cele trei puncte*, (...) –
mecanismul elipsă

□**exemplu:** funcție ce calculează suma a n numere întregi

int suma(int n,...);

suma(4,1,2,1,1) -> 5; suma(5,1,2,1,1,3) ->8

Funcții cu număr variabil de argumente

□ macro-urile din `stdarg.h`:

□ `va_list`: tip de date dedicat manipulării listelor cu număr variabil de parametri (de obicei e `unsigned char*`)

□ `va_start(va_list lp, numeArgument)` : extrage în lista `lp` parametri funcției care urmează după ultimul parametru fix specificat de `numeArgument`;

□ `va_arg(va_list lp, tip_de_date)`: extrage la fiecare apel câte o valoare din lista `lp` – valoarea se consideră de tipul `tip_de_date` indicat ca parametru (poate fi `int` sau `double`);

□ `va_end(va_list lp)`: obligatoriu când se încheie operațiile pe lista `lp`

Funcții cu număr variabil de argumente

❑ **exemplul 1:** funcție ce calculează suma a n numere întregi

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdarg.h>
4
5  int suma(int n, ...)
6  {
7      int i,s;
8      va_list listaParametri;
9      va_start(listaParametri,n);
10     s = 0;
11     for(i=0;i<n;i++)
12         s = s + va_arg(listaParametri,int);
13     va_end(listaParametri);
14     return s;
15 }
16
17
18 int main()
19 {
20     int a;
21     a = suma(4,1,2,1,1);
22     printf("a=%d\n",a);
23     a = suma(5,1,2,1,1,3);
24     printf("a=%d\n",a);
25     return 0;
26 }
```

a=5

a=8

Funcții cu număr variabil de argumente

```
void f(int x, ...)
{
    va_list lp; //declara lista de parametri
    va_start(lp, x); /*initializeaza lista de parametri, trebuie sa stiu unde incepe, dupa x */
    for( ; ; ){
        tip_de_date t = va_arg(lp, tip_de_date); //extrage parametrul curent
        ... }
    va_end(args);
}
```

- ❑ apelăm funcția **f** dintr-o altă funcție **g**;
- ❑ **f** trebuie să știe ce parametri primește;
- ❑ folosim **va_start** care apelează ultimul parametru formal cunoscut(**x**) transmis și reținut în stivă;
- ❑ transmiterea parametrilor se face de la stânga la dreapta într-o stivă.

Funcții cu număr variabil de argumente

```
void f(int x, ...)
```

```
{
```

```
    va_list lp; //declara lista de parametri
```

```
    va_start(lp, x); //initializeaza lista de parametri, trebuie sa stiu unde incepe, dupa x
```

```
    for( ; ; ){
```

```
        tip_de_date t = va_arg(lp, tip_de_date); //extrage parametrul curent
```

```
        ... }
```

```
    va_end(args);
```

```
}
```

- ❑ **va_start** găsește adresa lui x din stivă (e macrou și nu funcție, are acces la stiva bună!) și apoi din stivă ia fiecare argument transmis cu ajutorul lui **va_arg**;
- ❑ **va_arg** trebuie să știe ce dimensiune în octeți are parametrul pe care trebuie să îl extragă din stivă;
- ❑ **va_end** este obligatoriu, altfel rezultatul e “undefined”;
- ❑ funcția **f** trebuie să știe unde se oprește cu citirea parametrilor.

Funcții cu număr variabil de argumente

□ posibilă definire a macrou-rilor `va_list`, `va_start`, `va_arg`:

```
typedef unsigned char * va_list;
```

```
#define va_start(lp,param) (lp = (((va_list)&param) + sizeof(param)))
```

```
#define va_arg(lp,type) (*(type *)(((lp += sizeof(type)) - sizeof(type)))
```

□ `va_list` e un pointer la char (adresă de variabilă stocată pe un octet);

□ `va_start` inițializează lista de argumente ca un pointer ce reține adresa imediată după ultimul parametru formal transmis în stivă. De la această adresa încep parametri în număr variabil;

□ `va_arg` realizează două lucruri:

- updatează lista de argumente la următorul argument mutând pointerul (aritmetica pointerilor);

- returnează valoarea argumentului actual (se întoarce);

Funcții cu număr variabil de argumente

□ **exemplul 1:** funcție ce calculează suma a n numere întregi

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdarg.h>
4
5  int suma(int n, ...)
6  {
7      int i,s;
8      va_list listaParametri;
9      va_start(listaParametri,n);
10     s = 0;
11     for(i=0;i<n;i++)
12         s = s + va_arg(listaParametri,int);
13     va_end(listaParametri);
14     return s;
15 }
16
17
18 int main()
19 {
20     int a;
21     a = suma(4,1,2,1,1);
22     printf("a=%d\n",a);
23     a = suma(5,1,2,1,1,3);
24     printf("a=%d\n",a);
25     return 0;
26 }
```

a=5

a=8

Funcții cu număr variabil de argumente

Exemplul 1: funcție ce calculează suma a n numere întregi

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdarg.h>
4
5  int suma(int n, ...)
6  {
7      int i,s;
8      va_list listaParametri;
9      va_start(listaParametri,n);
10     s = 0;
11     for(i=0;i<n;i++)
12         s = s + va_arg(listaParametri,int);
13     va_end(listaParametri);
14     return s;
15 }
16
17 int main()
18 {
19     int a = suma(4,1,2,1,1);
20     printf("a=%d\n",a);
21     a = suma(5,1,2,1,1,3);
22     printf("a=%d\n",a);
23     a = suma(2,1,2,1,1,3);
24     printf("a=%d\n",a);
25     a = suma(7,1,2,1,1,3);
26     printf("a=%d\n",a);
27     return 0;
28 }
```

a=5

a=8

a=3

a=-1082128046

Funcții cu număr variabil de argumente

□ **exemplul 2:** suma unui șir de numere întregi ce se termină cu 0

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdarg.h>
4
5  int suma(int x, ...)
6  {
7      int t,s;
8      va_list listaParametri;
9      va_start(listaParametri,x);
10     s = x;
11     do
12     {
13         t = va_arg(listaParametri,int);
14         s = s+t;
15     }
16     while(t!=0);
17     va_end(listaParametri);
18     return s;
19 }
20
21 int main()
22 {
23     int a = suma(4,1,2,1,1,0);
24     printf("a=%d\n",a);
25     return 0;
26 }
```

a=9

Process returned 0 (0x0) execution time : 0.012 s
Press ENTER to continue.

Funcții cu număr variabil de argumente

□ exemplul 3: maximul a n numere întregi

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdarg.h>
```

```
4
5 int maxim(int n,...)
```

```
6 {
7     int max,i,aux;
8     va_list lp;
9     va_start(lp,n);
10    max = va_arg(lp,int);
11    for(i=2;i<=n;i++)
12    {
13        aux = va_arg(lp,int);
14        if(max<aux)
15            max = aux;
16    }
17    va_end(lp);
18    return max;
19 }
```

```
20
21 int main()
```

```
22 {
```

```
23     int a = maxim(7,1,2,10,5,7,4,3);
```

```
24     printf("a=%d\n",a);
```

```
25     return 0;
```

```
26 }
```

a=10

Process returned 0 (0x0)

execution time : 0.005 s

Funcții cu număr variabil de argumente

□ **exemplul 4:** concatenarea unui număr variabil de șiruri de caractere într-un singur șir alocat dinamic. Marcăm sfârșitul șirurilor printr-un șir vid.

```
char *concateneazaSiruri(const char *primulSir, ...)
{
    va_list listaParametri;
    char *p, *q;

    if(primulSir == NULL) return NULL;
    int lungimeSir = strlen(primulSir);
    va_start(listaParametri, primulSir);
    while((p = va_arg(listaParametri, char *)) != NULL)
        lungimeSir += strlen(p);
    va_end(listaParametri);
    q = (char *) malloc(lungimeSir + 1);
    if(q == NULL) return NULL;
    strcpy(q, primulSir);
    va_start(listaParametri, primulSir);
    while((p = va_arg(listaParametri, char *)) != NULL)
        strcat(q, p);
    va_end(listaParametri);
    return q;
}
```

Funcții cu număr variabil de argumente

vaListExemplu4.c

```
1  #include <stdlib.h>
2  #include <stdarg.h>
3  #include <string.h>
4
5  char *concateneazaSiruri(const char *primulSir, ...)
6  {
7      va_list listaParametri;
8      char *p,*q;
9
10     if(primulSir == NULL) return NULL;
11     int lungimeSir = strlen(primulSir);
12     va_start(listaParametri, primulSir);
13     while((p = va_arg(listaParametri, char *)) != NULL)
14         lungimeSir += strlen(p);
15     va_end(listaParametri);
16     q = (char *) malloc(lungimeSir + 1);
17     if(q == NULL) return NULL;
18     strcpy(q, primulSir);
19     va_start(listaParametri, primulSir);
20     while((p = va_arg(listaParametri, char *)) != NULL)
21         strcat(q, p);
22     va_end(listaParametri);
23     return q;
24 }
25
26 int main()
27 {
28     char *str = concateneazaSiruri("Funcțiile cu numar variabil ", "de argumente sunt ", "foarte simple!!!", (char *)'\0');
29     printf("%s \n",str);
30     return 0;
31 }
```

P/curs12/vaListExemplu4

Funcțiile cu număr variabil de argumente sunt foarte simple!!!

Redden Alvin MacBook Pro, 13-inch, Retina Display

Cursul 11

1. Structuri de date complexe și autoreferite
2. Funcții cu număr variabil de argumente

Cursul 12

1. Preluarea argumentelor funcției main din linia de comandă
2. Programare generică