

PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,

2018-2019

Cursul 12

Test de laborator – enunț

Subiectul I – 3 puncte

Fie n un număr natural nenul. Definim scrierea sa binară scurtă ca fiind scrierea sa binară începând cu poziția celui mai din stânga bit setat (care are valoarea = 1). Definim *greutatea* numărului n ca fiind numărul de biți setați din scrierea sa binară scurtă. Spunem că numărul n este *rotund* dacă numărul de biți setați este egal cu numărul de biți nesetați (care au valoarea = 0) din scrierea sa binară scurtă.

Exemplu: Numărul 7 are scrierea binară scurtă 111, deci are greutatea 3 și nu e rotund. Numărul 202 are scrierea binară scurtă 11001010, deci are greutatea 4 și e rotund.

- (1 punct) Scrieți o funcție care să citească de la tastatură un tablou unidimensional alocat dinamic cu elemente numere naturale nenule. Numărul de elemente al tabloului și valorile acestora se vor citi în cadrul funcție. Funcția va furniza accesul la tabloul alocat dinamic precum și numărul de elemente al acestuia.
- (1 punct) Scrieți o funcție care pentru un număr natural nenul n primit ca parametru determină și furnizează numărul de biți setați (cu valoarea 1) și de biți nesetați (cu valoarea 0) din scrierea sa binară scurtă.
- (1 punct) Scrieți un program care, folosind apeluri utile ale funcțiilor definite anterior, citește de la tastatură un tablou unidimensional de numere naturale nenule alocat dinamic și afișează la ecran numerele rotunde de greutate minimă (dintre numerele rotunde) din tabloul citit. Dacă nu există niciun număr rotund în tabloul citit se va afișa mesajul *Imposibil*. La sfârșitul executării programului se va elibera memoria alocată dinamic folosită.

Exemplu: se citește un tablou cu 4 elemente: 10 9 7 202. Programul va afișa 10 9.

Explicație: În scriere binară scurtă avem $10_{(2)} = 1010$, $9_{(2)} = 1001$, $7_{(2)} = 111$, $202_{(2)} = 11001010$. Numerele 10, 9, 202 sunt rotunde, numărul 7 nu este rotund. Numerele 9 și 10 au greutatea 2 și vor fi afișate, numărul 202 are greutatea 4 și nu va fi afișat.

Test de laborator – enunț

Subiectul II – 6 puncte

Fișierul text *triunghiuri_dreptunghice.txt* conține pe prima linie un număr natural n ($1 \leq n \leq 1000$) și apoi n linii unde fiecare linie conține 2 numere reale separate prin spațiu ce codifică un triunghi dreptunghic prin dimensiunile celor două catete, *cateta1* și *cateta2*.

- (0.5 puncte) Definiți o structură cu numele *triunghiDreptunghic* care să vă permită citirea triunghiurilor din fișier precum și calculul ariilor și a perimetrelor lor.
- (1.5 puncte) Scrieți o funcție care primește ca parametru numele unui fișier text (cu o structură similară cu cea a fișierului *triunghiuri_dreptunghice.txt*) și citește într-un tablou unidimensional alocat dinamic cu elemente de tip *triunghiDreptunghic* toate triunghiurile din fișierul respectiv. Funcția va furniza accesul la tabloul alocat dinamic precum și numărul de elemente al acestuia.
- (1 punct) Scrieți o funcție care primește ca parametru un tablou unidimensional de elemente de tip *triunghiDreptunghic*, lungimea acestuia și un număr real $x > 0$ și adună la dimensiunea catetelor valoarea reală x actualizând pentru fiecare triunghi în parte catetele, aria și perimetru.

Test de laborator – enunț

- d) **(3 puncte)** Scrieți un program care pe baza apelurilor funcțiilor anterioare realizează:
- i. citește triunghiurile din fișierul *triunghiuri_dreptunghice.txt* într-un tablou *v* unidimensional alocat dinamic de elemente de tip *triunghiDreptunghic*.
 - ii. redimensionează tabloul unidimensional inițial *v* la un tablou *w* cu de două ori mai multe elemente copiind elementele din prima parte a tabloului (tabloul inițial *v*) în a doua parte a tabloului. Tabloul *w* nou obținut conține tabloul inițial *v* de două ori (prin abuz de notatie am putea scrie *w* = [*v v*]);
 - iii. pentru toate elementele din a doua jumătate a tabloului *w*, adună la dimensiunea catetelor fiecărui triunghi valoarea 1 și actualizează catetele, perimetrul și aria acestor trinhiuri folosind funcția de la punctul c;
 - iv. folosind apeluri ale funcției *qsort*, sortează tabloul *w* astfel: în prima jumătate se sortează triunghiurile în ordinea crescătoare a ariei, iar pentru arii egale în ordinea crescătoare a perimetrului; în a doua jumătate a lui *w* se sortează triunghiurile în ordinea descrescătoare a ariei, iar pentru arii egale în ordinea descrescătoare a perimetrului.
 - v. scrieți în fișierul text *triunghiuri_sortate.txt* ordinea astfel obținută pentru tabloul *w*, câte un triunghi pe fiecare linie, reprezentat prin dimensiunile celor trei laturi.
 - vi. eliberati memoria locată dinamic.

Test de laborator - barem

Subiectul I – 3 puncte

a) 1 punct

- antet corect – 0.25 puncte
- alocare dinamică corectă – 0.4 puncte
- citire corectă a numărului de elemente și a valorilor tabloului – 0.1 puncte
- furnizarea corecta a rezultatelor – 0.25 puncte

Observație: Se va puncta complet orice variantă corectă (funcția returnează o structură care conține adresa de început a tabloului și lungimea lui, funcția returnează un parametru iar celalalt e trimis ca pointer, etc.).

b) 1 punct

- antet corect – 0.25 puncte
- determinare scriere binară scurtă – 0.5 puncte
- calcul corect biți setați și nesetați – 0.25 puncte

Observație: Se va puncta complet orice variantă corectă (se folosesc operații pe biți, se folosesc operații aritmice / și %, et.).

c) 1 punct

- apeluri corecte ale funcțiilor – 0.5 puncte
- determinare număr rotund – 0.25 puncte
- determinare număr rotund de greutate minimă 0.20 puncte + mesaj Imposibil 0.05 puncte – 0.25 puncte

Observație: Se va puncta apelarea corectă a funcțiilor, chiar dacă nu se rezolvă corect cerința.

Test de laborator - barem

Subiectul II – 6 puncte

a) 0.5 puncte

- definire structură – 0.5 puncte

b) 1.5 puncte

- antet corect – 0.25 puncte
- deschidere fisier – 0.25 puncte
- citire corecta din fisier si asignarea elementelor in tabloul de structuri 0.75 puncte
- furnizare corectă a tabloului 1D și a lungimii acestuia – 0.25 puncte

c) 1 punct

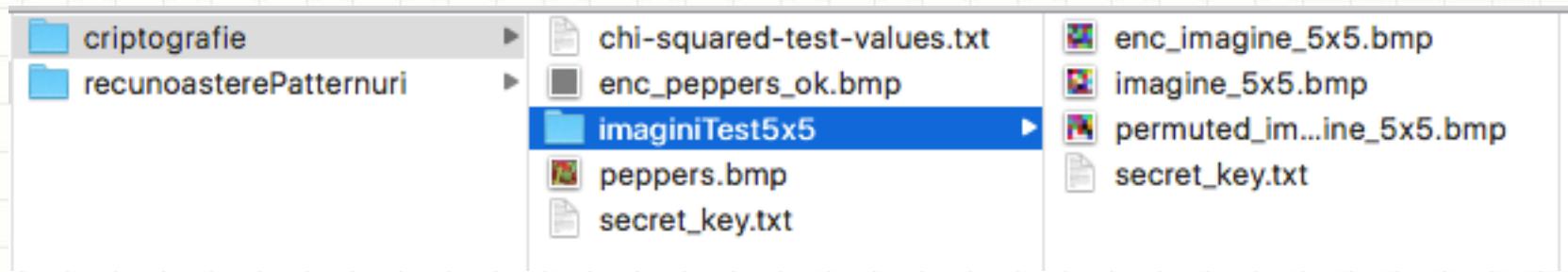
- antet corect – 0.25 puncte
- adunare corectă cu x a catetelor – 0.25 puncte
- actualizare celelalte câmpuri - 0.5 puncte

d) 3 puncte

- i. apel corect pentru citire - 0.25 puncte
- ii. Redimensionare cu realloc – 0.4 puncte + copiere conținut 0.1 puncte = 0.5 puncte
- iii. apel corect a funcției de la punctul c) – 0.25 puncte
- iv. scriere funcție comparator 2×0.5 puncte = 1 punct + apel qsort in main – $2 \times 0.25 = 0.5$ puncte, total = 1.5 puncte
- v. deschidere și închidere de fișier 0.1 puncte , scrierea in fisier pe fiecare linie în ordinea corectă a triunghiurilor – 0.3 puncte, total 0.4 puncte
- vi.eliberare memorie a lui w cu free – 0.1 puncte

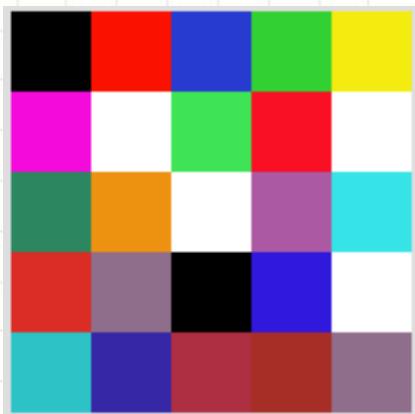
Proiect – criptare + decriptare

- ❑ am verificat sursa noastră pentru criptare, nu este nici greșală.
- ❑ pentru a vă verifica mai ușor am adăugat în arhivă un exemplu de criptare+decriptare pentru o imagine 5 x5 pixeli.

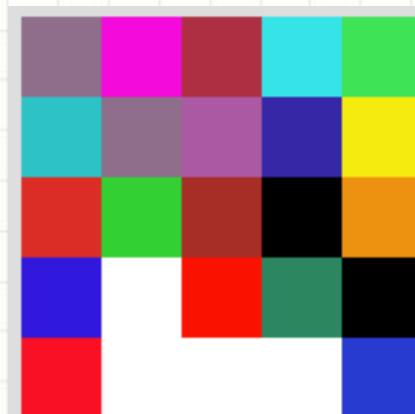


- ❑ *imagine_5x5.bmp* – imagine 5x5 pixeli
- ❑ *permuted_imagine_5x5.bmp* – imaginea cu pixelii permutați
- ❑ *enc_imagine_5x5.bmp* – imaginea criptată

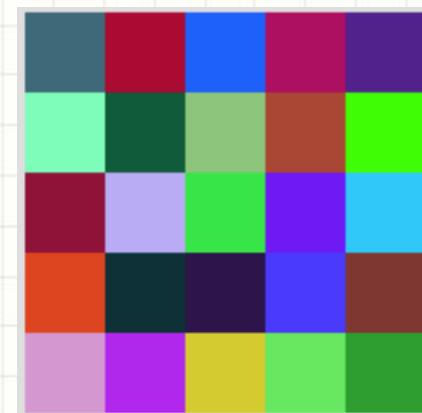
Imagine test 5x5



imagine_5x5.bmp



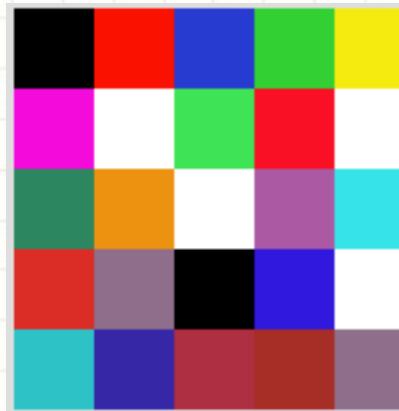
permuted_imagine_5x5.bmp



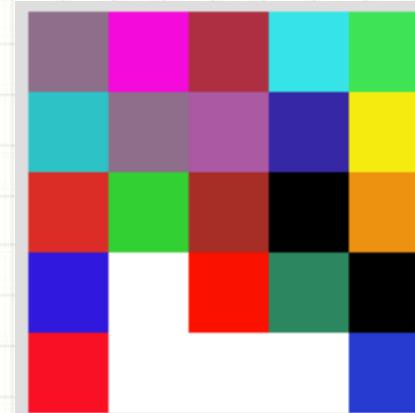
enc_imagine_5x5.bmp

Permutarea obținută

```
i = 0, permutation[0] = 13
i = 1, permutation[1] = 17
i = 2, permutation[2] = 24
i = 3, permutation[3] = 11
i = 4, permutation[4] = 9
i = 5, permutation[5] = 1
i = 6, permutation[6] = 22
i = 7, permutation[7] = 4
i = 8, permutation[8] = 20
i = 9, permutation[9] = 21
i = 10, permutation[10] = 18
i = 11, permutation[11] = 14
i = 12, permutation[12] = 16
i = 13, permutation[13] = 7
i = 14, permutation[14] = 3
i = 15, permutation[15] = 10
i = 16, permutation[16] = 0
i = 17, permutation[17] = 19
i = 18, permutation[18] = 15
i = 19, permutation[19] = 23
i = 20, permutation[20] = 5
i = 21, permutation[21] = 8
i = 22, permutation[22] = 2
i = 23, permutation[23] = 12
i = 24, permutation[24] = 6
```



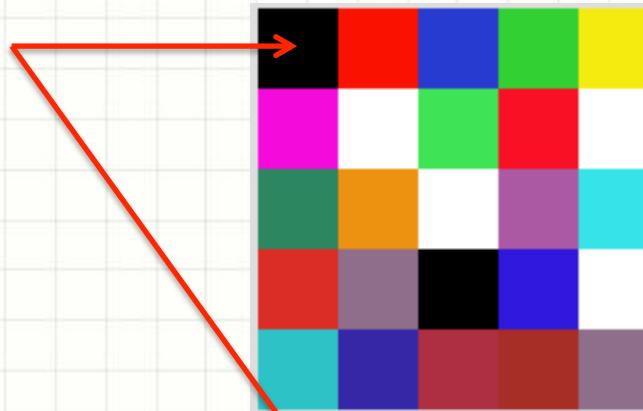
imagine_5x5.bmp



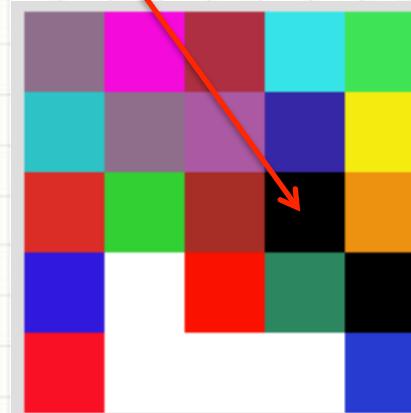
permuted_imagine_5x5.bmp

Permutarea obținută

```
i = 0, permutation[0] = 13
i = 1, permutation[1] = 17
i = 2, permutation[2] = 24
i = 3, permutation[3] = 11
i = 4, permutation[4] = 9
i = 5, permutation[5] = 1
i = 6, permutation[6] = 22
i = 7, permutation[7] = 4
i = 8, permutation[8] = 20
i = 9, permutation[9] = 21
i = 10, permutation[10] = 18
i = 11, permutation[11] = 14
i = 12, permutation[12] = 16
i = 13, permutation[13] = 7
i = 14, permutation[14] = 3
i = 15, permutation[15] = 10
i = 16, permutation[16] = 0
i = 17, permutation[17] = 19
i = 18, permutation[18] = 15
i = 19, permutation[19] = 23
i = 20, permutation[20] = 5
i = 21, permutation[21] = 8
i = 22, permutation[22] = 2
i = 23, permutation[23] = 12
i = 24, permutation[24] = 6
```



imagine_5x5.bmp



permuted_imagine_5x5.bmp

Verificare cu Matlab

```
Trial>> img = imread('imagine_5x5.bmp')
```

```
5x5x3 uint8 array
```

```
img(:,:,1) =
```

0	255	39	39	244
244	255	63	255	255
44	236	255	171	17
219	143	0	48	255
39	54	174	167	143

```
img(:,:,2) =
```

0	0	59	208	236
11	255	227	0	255
135	145	255	89	228
45	110	0	25	255
196	39	47	45	110

```
img(:,:,3) =
```

0	0	208	51	11
219	255	86	36	255
98	15	255	162	231
37	139	0	220	255
198	167	65	39	139

Verificare cu Matlab

```
Trial>> img_permutata = imread('permuted_imagine_5x5.bmp')
```

5x5x3 uint8 array

```
img_permutata(:,:,1) =
```

143	244	174	17	63
39	143	171	54	244
219	39	167	0	236
48	255	255	44	0
255	255	255	255	39

```
img_permutata(:,:,2) =
```

110	11	47	228	227
196	110	89	39	236
45	208	45	0	145
25	255	0	135	0
0	255	255	255	59

```
img_permutata(:,:,3) =
```

139	219	65	231	86
198	139	162	167	11
37	51	39	0	15
220	255	0	98	0
36	255	255	255	208

Verificare cu Matlab

img(:,:,1) =

0	255	39	39	244
244	255	63	255	255
44	236	255	171	17
219	143	0	48	255
39	54	174	167	143

img_permutata(:,:1) =

143	244	174	17	63
39	143	171	54	244
219	39	167	0	236
48	255	255	44	0
255	255	255	255	39

img(:,:,2) =

0	0	59	208	236
11	255	227	0	255
135	145	255	89	228
45	110	0	25	255
196	39	47	45	110

img_permutata(:,:2) =

110	11	47	228	227
196	110	89	39	236
45	208	45	0	145
25	255	0	135	0
0	255	255	255	59

img(:,:,3) =

0	0	208	51	11
219	255	86	36	255
98	15	255	162	231
37	139	0	220	255
198	167	65	39	139

img_permutata(:,:3) =

139	219	65	231	86
198	139	162	167	11
37	51	39	0	15
220	255	0	98	0
36	255	255	255	208

Organizare În 2019

- Curs: cursurile 13 și 14 – programare generică, recursivitate, recapitulare
- Laborator: primiți notele de la test în prima săptămână din ianuarie, prezentați proiectul + exerciții cu funcții cu număr variabil de argumente, programare generică, recursivitate
- Seminar: ultimul seminar cu funcții cu număr variabil de argumente, programare generică, rercursivitate
- Examen final pe 4 februarie 2018

Recapitulare – cursul trecut

1. Tablouri de siruri de caractere
2. Preluarea argumentelor functiei main din linia de comanda
3. Functii cu numar variabil de argumente
4. Declaratii complexe

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: tablouri, siruri de caractere, structuri, uniuni, câmpuri de biți, enumerări, pointeri
 - Instrucțiuni de control
 - Directive de preprocessare. Macrodefiniții.
 - Funcții de citire/scriere.
 - Etapele realizării unui program C.
- Fișiere text**
 - Funcții specifice de manipulare.
- Fișiere binare**
 - Funcții specifice de manipulare.
- Funcții (1)**
 - Declarare și definire. Apel. Metode de transmitere a parametrilor. Pointeri la funcții.
- Tablouri și pointeri**
 - Aritmetică pointerilor
 - Legătura dintre tablouri și pointeri
 - Alocarea dinamică a memoriei
 - Clase de memorare
- Siruri de caractere**
 - 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. Funcții cu număr variabil de argumente
2. Declarații complexe
3. Structuri de date complexe și autoreferite

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 parametrii 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 cand se încheie operațiile pe lista lp

Funcții cu număr variabil de argumente

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



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

int suma(int n,...)
{
    int i, s;
    va_list listaParametri;
    va_start(listaParametri,n);
    s = 0;
    for(i = 0; i < n; i++)
        s = s + va_arg(listaParametri, int);
    va_end(listaParametri);
    return s;
}

int main()
{
    int a;
    a = suma(4,1,2,1,1);
    printf("a = %d\n", a);
    a = suma(5,1,2,1,1,3);
    printf("a = %d \n", a);
    return 0;
}
```

Funcții cu număr variabil de argumente

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

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

int suma(int n,...)
{
    int i, s;
    va_list listaParametri;
    va_start(listaParametri,n);
    s = 0;
    for(i = 0; i < n; i++)
        s = s + va_arg(listaParametri, int);
    va_end(listaParametri);
    return s;
}

int main()
{
    int a;
    a = suma(4,1,2,1,1);
    printf("a = %d\n", a);
    a = suma(5,1,2,1,1,3);
    printf("a = %d \n", a);
    return 0;
}
```

```
[Bogdan-Alexes-MacBook-Pro:curs11 bogdan$ gcc exemplu1.c
[Bogdan-Alexes-MacBook-Pro:curs11 bogdan$ ./a.out
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(lp); //elibereaza memoria  
}
```

- apelam functia **f** dintr-o alta functie **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 dreapta la stânga 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(lp); //elibereaza memoria  
}
```

- **va_start** găsește adresa lui x din stivă (e macro 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 macro-urilor `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` initializează 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:
 - updateaza lista de argumente la urmatorul argument mutând pointerul (aritmetică pointerilor);
 - returneaza valoarea argumentului actual (se întoarce);

Funcții cu număr variabil de argumente

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

```
00 exemplu1.c
01
02 1 #include <stdio.h>
03 2 #include <stdarg.h>
04 3 #include <string.h>
05
06 4
07 5 int suma(int n,...)
08 6 {
09 7     int i, s;
10 8     va_list listaParametri;
11 9     va_start(listaParametri,n);
12 10    s = 0;
13 11    for(i = 0; i < n; i++)
14 12        s = s + va_arg(listaParametri, int);
15 13    va_end(listaParametri);
16 14    return s;
17 15 }
18
19 16
20 17 int main()
21 18 {
22 19     int a;
23 20     a = suma(4,1,2,1,1);
24 21     printf("a = %d\n", a);
25 22     a = suma(5,1,2,1,1,3);
26 23     printf("a = %d \n", a);
27 24     a = suma(2,1,2,1,1,3);
28 25     printf("a = %d \n", a);
29 26     a = suma(7,1,2,1,1);
30 27     printf("a = %d \n", a);
31 28     return 0;
32 }
```

```
Bogdan-Alexes-MacBook-Pro:curs11 bogdan$ gcc exemplu1.c
Bogdan-Alexes-MacBook-Pro:curs11 bogdan$ ./a.out
a = 5
a = 8
a = 3
a = -515301515
```

Funcții cu număr variabil de argumente

- exemplul 2: suma unui sir de numere întregi ce se temină

cu 0

```
exemplu2.c
1 #include <stdio.h>
2 #include <stdarg.h>
3 #include <string.h>
4
5 int suma(int x,...)
6 {
7     int t, s;
8     va_list lp;
9     va_start(lp,x);
10    s = x;
11    do
12    {
13        t = va_arg(lp,int);
14        s = s + t;
15    }
16    while(t);
17    va_end(lp);
18    return s;
19 }
20
21 int main()
22 {
23     int a;
24     a = suma(4,1,2,1,1,0);
25     printf("a = %d\n", a);
26
27     return 0;
28 }
```

```
Bogdan-Alexes-MacBook-Pro:curs11 bogdan$ gcc exemplu2.c
Bogdan-Alexes-MacBook-Pro:curs11 bogdan$ ./a.out
a = 9
```

Functii cu număr variabil de argumente

□ exemplul 3: maximul a n numere întregi

```
exemplu3.c
01 #include <stdio.h>
02 #include <stdarg.h>
03 #include <string.h>
04
05 int maxim(int n,...)
06 {
07     int max, i, aux;
08     va_list lp;
09     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;
24     a = maxim(7,1,2,19,5,7,4,3);
25     printf("a = %d\n", a);
26
27     return 0;
28 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs11 bogdan$ gcc exemplu3.c
[Bogdan-Alexes-MacBook-Pro:curs11 bogdan$ ./a.out
a = 19
```

Funcții cu număr variabil de argumente

- **exemplul 4:** concatenarea unui număr variabil de siruri de caractere într-un singur sir alocat dinamic. Marcăm sfârșitul sirurilor printr-un sir vid.

```
6  char *concateneazaSiruri(const char *primulSir, ...)
7  {
8      va_list listaParametri;
9      char *p,*q;
10
11     if(primulSir == NULL) return NULL;
12     int lungimeSir = strlen(primulSir);
13     va_start(listaParametri, primulSir);
14     while((p = va_arg(listaParametri, char *)) != NULL)
15         lungimeSir += strlen(p);
16     va_end(listaParametri);
17     q = (char *) malloc(lungimeSir + 1);
18     if(q == NULL) return NULL;
19     strcpy(q, primulSir);
20     va_start(listaParametri, primulSir);
21     while((p = va_arg(listaParametri, char *)) != NULL)
22         strcat(q, p);
23     va_end(listaParametri);
24     return q;
25 }
```

Functii cu număr variabil de argumente

```
exemplu4.c
1 #include <stdio.h>
2 #include <stdarg.h>
3 #include <string.h>
4 #include <stdlib.h>
5
6 char *concateneazaSiruri(const char *primulSir, ...)
7 {
8     va_list listaParametri;
9     char *p,*q;
10
11    if(primulSir == NULL) return NULL;
12    int lungimeSir = strlen(primulSir);
13    va_start(listaParametri, primulSir);
14    while((p = va_arg(listaParametri, char *)) != NULL)
15        lungimeSir += strlen(p);
16    va_end(listaParametri);
17    q = (char *) malloc(lungimeSir + 1);
18    if(q == NULL) return NULL;
19    strcpy(q, primulSir);
20    va_start(listaParametri, primulSir);
21    while((p = va_arg(listaParametri, char *)) != NULL)
22        strcat(q, p);
23    va_end(listaParametri);
24    return q;
25 }
26
27 int main()
28 {
29     char *str = concateneazaSiruri("Functiile cu numar variabil ", "de argumente sunt ", "foarte simple!!!", (char *)'\0');
30     printf("%s \n",str);
31     return 0;
32 }
```

[Bogdan-Alexes-MacBook-Pro:curs11 bogdan\$ gcc exemplu4.c

[Bogdan-Alexes-MacBook-Pro:curs11 bogdan\$./a.out

Functiile cu numar variabil de argumente sunt foarte simple!!!

Funcții cu număr variabil de argumente

□ probleme la seminar:

Probleme propuse

1. Scrieți o funcție cu număr variabil de parametri care să concateneze mai multe șiruri de caractere, parcurgând o singură listă parametrilor variabili.
2. Scrieți o funcție cu număr variabil de parametri care să determine prefixul comun de lungime maximă al mai multor șiruri de caractere.
3. Scrieți o funcție cu număr variabil de parametri care să returneze numărul de apariții ale unui număr întreg dat într-un șir de numere întregi. Folosind apele utile ale funcției definite anterior, scrieți o funcție care verifică dacă 4 numere întregi sunt distințe sau nu.
4. Scrieți o funcție cu număr variabil de parametri care să creeze un șir de caractere din mai multe caractere date.
5. Scrieți o funcție cu număr variabil de parametri care să creeze un tablou de numere întregi din mai multe numere întregi. Atenție, funcția trebuie să furnizeze și lungimea tabloului!

Cuprinsul cursului de azi

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

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

Cuprinsul cursului de azi

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

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.

Transmiterea structurilor ca parametri

```
student1.c      x
1 #include<stdio.h>
2
3
4 typedef struct {
5     char nume[30];
6     char prenume[30];
7     float notaExamen;
8 } student;
9
10 void adaugaUnPunct(student x)
11 {
12     x.notaExamen++;
13     if (x.notaExamen > 10)
14         x.notaExamen = 10;
15 }
16
17 int main()
18 {
19     student A = {"Popescu","Maria",4.75};
20     adaugaUnPunct(A);
21     printf("%s %s %f \n", A.nume,A.prenume,A.notaExamen);
22     return 0;
23 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs12 bogdan$ gcc -o student1 student1.c
[Bogdan-Alexes-MacBook-Pro:curs12 bogdan$ ./student1
Popescu Maria 4.750000
```

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      x
001 #include<stdio.h>
002
003
004 typedef struct {
005     char nume[30];
006     char prenume[30];
007     float notaExamen;
008 } student;
009
010 void adaugaUnPunct(student* x)
011 {
012     x->notaExamen++;
013     if (x->notaExamen > 10)
014         x->notaExamen = 10;
015 }
016
017 int main()
018 {
019     student A = {"Popescu","Maria",4.75};
020     adaugaUnPunct(&A);
021     printf("%s %s %f \n", A.nume,A.prenume,A.notaExamen);
022     return 0;
023 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs12 bogdan$ gcc -o student2 student2.c
[Bogdan-Alexes-MacBook-Pro:curs12 bogdan$ ./student2
Popescu Maria 5.750000]
```

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 notaExamen;  
    struct adresa;  
}
```

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

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

Exemplu

- ❑ 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.



```
B
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
```

Exemplu

- ❑ 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.

```
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;
```

Exemplu

- 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 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.

Exemplu

- ❑ 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

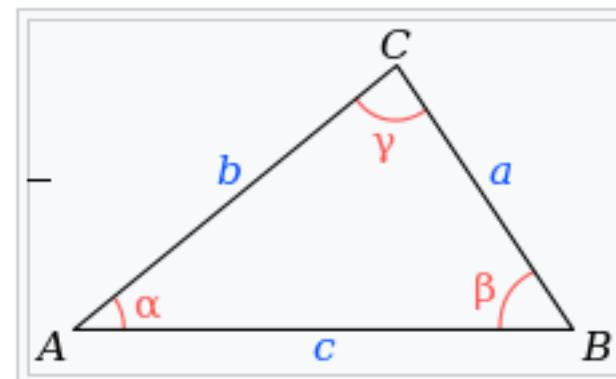
De la Wikipedia, enciclopedia liberă

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

Exemplu

```
triunghi_arg.c      x

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <string.h>
5
6 typedef struct {
7     float abscisa;
8     float ordonata;
9 } punct2D;
10
11 typedef struct{
12     punct2D A, B, C;
13     float AB, AC, BC;
14     int triunghiValid;
15     float perimetru;
16     float arie;
17 } triunghi;
18
19 int citesteTriunghiuri(char*, triunghi**);
20 void afisareTriunghiuri(triunghi*, int);
```

Exemplu

```
64 int citesteTriunghiuri(char *nume, triunghi **pT)
65 {
66     FILE* f = fopen(nume,"r");
67     if(f==NULL) { printf("Eroare la citirea fisierului \n"); exit(0); }
68     int i,n;
69     fscanf(f,"%d",&n);
70     triunghi* p = (triunghi *) malloc(n*sizeof(triunghi));
71     if (p==NULL) { printf("Eroare la alocare"); exit(0); }
72     punct2D A,B,C;
73     for(i=0;i<n;i++)
74     {
75         fscanf(f,"%f %f %f %f %f",&A.abscisa,&Aordonata,&B.abscisa,
76                 &Bordonata,&C.abscisa,&Cordonata);
77         float AB = sqrt(pow(A.abscisa - B.abscisa,2) + pow(Aordonata - Bordonata,2));
78         float AC = sqrt(pow(A.abscisa - C.abscisa,2) + pow(Aordonata - Cordonata,2));
79         float BC = sqrt(pow(B.abscisa - C.abscisa,2) + pow(Bordonata - Cordonata,2));
80         p[i].A = A; p[i].B = B; p[i].C = C; p[i].AB = AB; p[i].AC = AC; p[i].BC = BC;
81         //validare triunghi
82         if ((AB + BC == AC) || (AB + AC == BC) || (AC + BC == AB))
83             p[i].triunghiValid = 0;
84         else
85             p[i].triunghiValid = 1;
86         if(p[i].triunghiValid)
87         {
88             p[i].perimetru = AB + AC + BC;
89             float sp = p[i].perimetru/2;
90             p[i].arie = sqrt(sp * (sp - AB) * (sp - BC) * (sp - AC));
91         }
92     *pT = p;
93     fclose(f);
94     return n;
95 }
```

Exemplu

```
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96 void afisareTriunghiuri(triunghi* pT,int n)
97 {
98     int i;
99     for(i=0;i<n;i++)
100    {
101        printf("*****\n");
102        if(pT[i].triunghiValid)
103            printf("Triunghiul %d: \n",i);
104        printf("Punctul A are coordonatele (%f,%f) \n",pT[i].A.abscisa,pT[i].Aordonata);
105        printf("Punctul B are coordonatele (%f,%f) \n",pT[i].B.abscisa,pT[i].Bordonata);
106        printf("Punctul C are coordonatele (%f,%f) \n",pT[i].C.abscisa,pT[i].Cordonata);
107        printf("Segmentul AB are lungimea %f \n",pT[i].AB);
108        printf("Segmentul AC are lungimea %f \n",pT[i].AC);
109        printf("Segmentul BC are lungimea %f \n",pT[i].BC);
110
111        if(!pT[i].triunghiValid)
112            printf("Punctele sunt coliniare si nu formeaza un triunghi\n");
113        if(pT[i].triunghiValid)
114            printf("Aria triunghiului este %f \n", pT[i].arie);
115    }
116    return;
117 }
```

Exemplu

```
22 int main(int argc, char **argv)
23 {
24     char numeFisier[1000];
25     if (argc == 2)
26         strcpy(numeFisier,argv[1]);
27     else
28         {printf("Programul executabil asteapta un argument \n");exit(0);}
29
30     triunghi *T = NULL;
31     int n = citesteTriunghiuri(numeFisier,&T);
32     afisareTriunghiuri(T,n);
33
34     float arieMaxima = 0;
35     int i,indiceTriunghi = -1;
36     for (i=0;i<n;i++)
37     {
38         if (T[i].triunghiValid)
39             if (arieMaxima < T[i].arie)
40                 {arieMaxima = T[i].arie; indiceTriunghi = i;}
41     }
42     if (arieMaxima)
43     {
44         printf("Triunghiul de arie maxima are aria = %f \n",T[indiceTriunghi].arie);
45         printf("Triunghiul contine punctele: (%f,%f) (%f,%f) (%f,%f) \n",
46                T[indiceTriunghi].A.abscisa,T[indiceTriunghi].A.ordonata,
47                T[indiceTriunghi].B.abscisa, T[indiceTriunghi].A.ordonata,
48                T[indiceTriunghi].C.abscisa, T[indiceTriunghi].C.ordonata);
49     }
50     else
51         printf("Nu exista \n");
52     return 0;
53 }
```

Exemplu

```
Bogdan-Alexes-MacBook-Pro:curs12 bogdan$ gcc -o triunghi triunghi_arg.c
Bogdan-Alexes-MacBook-Pro:curs12 bogdan$ ./triunghi 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)
```

Sortarea unui tablou de structuri

```
void sorteazaTriunghiuri(triunghi* p, int n)
{
    triunghi aux;

    int i,j;
    for(i=0;i<n;i++)
    {
        if(!p[i].triunghiValid)
            p[i].arie = 0;
    }
    for(i=0;i<n;i++)
        for(j=i+1;j<n;j++)
            if(p[i].arie < p[j].arie)
            {
                aux = p[i];
                p[i] = p[j];
                p[j] = aux;
            }
    return;
}
```

```
sorteazaTriunghiuri(T,n);
afisareTriunghiuri(T,n);
return 0;
```

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
```

Sortarea unui tablou de structuri cu qsort

```
141 int cmpTriunghiuri(const void *a, const void *b)
142 {
143     triunghi *pa = (triunghi *) a;
144     triunghi *pb = (triunghi *) b;
145     if(pa->triunghiValid == pb->triunghiValid)
146     {
147         if (pa->triunghiValid ==0)
148             return 0;
149         return pb->arie - pa->arie;
150     }
151     if (pa->triunghiValid)
152         return -1;
153     return 1;
154 }
```

Posibil incoprect, fac
diferența a două
float-uri care apoi e
convertită la int

```
54
55     //sorteazaTriunghiuri(T,n);
56     qsort(T,n,sizeof(triunghi),cmpTriunghiuri);
57     afisareTriunghiuri(T,n);
58     return 0;
```

Sortarea unui tablou de structuri cu qsort

```
140
141     int cmpTriunghiuri(const void *a, const void *b)
142     {
143         triunghi *pa = (triunghi *) a;
144         triunghi *pb = (triunghi *) b;
145         if(pa->triunghiValid == pb->triunghiValid)
146         {
147             if (pa->triunghiValid ==0)
148                 return 0;
149             if (pb->arie < pa->arie)
150                 return -1;
151             if (pb->arie > pa->arie)
152                 return 1;
153             if (pb->arie == pa->arie)
154                 return 0;
155         }
156         if (_pa->triunghiValid)
157             return -1;
158         return 1;
159     }
160 }
```

```
54
55     //sorteazaTriunghiuri(T,n);
56     qsort(T,n,sizeof(triunghi),cmpTriunghiuri);
57     afisareTriunghiuri(T,n);
58     return 0;
```

Sortarea unui tablou de structuri cu qsort

```
*****
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

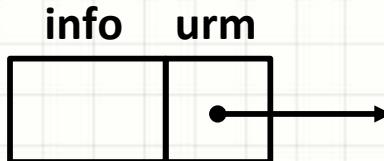
```
struct T{  
    char ch;  
    int i;  
    struct T t;  
}
```

declaratie invalida

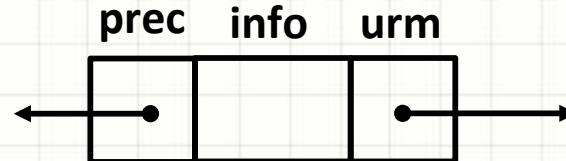
Structuri autoreferite

- ❑ aplicații pentru structuri de date în alocare dinamică

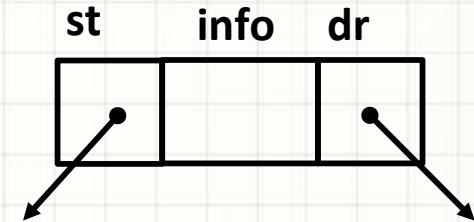
```
struct nod{  
    int info;  
    struct nod *urm;  
}  
  
declaratie unui  
structuri nod
```



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



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



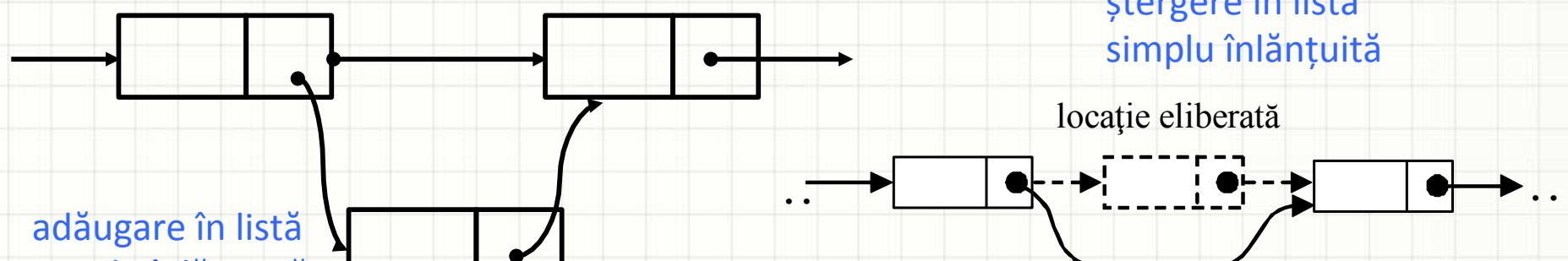
- ❑ fiecare nod conține:
 - ❑ un câmp/mai multe câmpuri cu informația nodului - **info**
 - ❑ un pointer/mai mulți pointeri către nodurile vecine: următor, precedent, fiuStang, fiuDrept

Structuri autoreferite

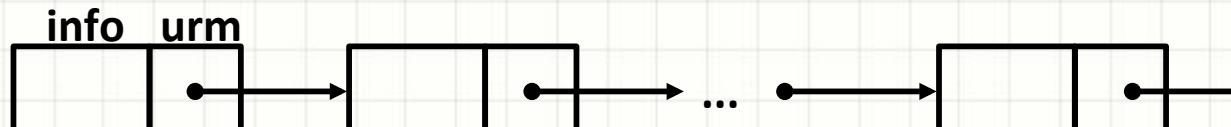
- aplicații pentru structuri de date în alocare dinamică
- liste, stive, cozi, arbori
 - avantaj față de implementarea statică
 - operațiile de adăugare sau ștergere sunt foarte rapide
 - dezavantaj față de implementarea statică :
 - accesul la un nod se face prin parcurgerea nodurilor precedente
 -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

Structuri autoreferite

- ❑ aplicații pentru structuri de date în alocare dinamică
- ❑ 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

The screenshot shows two windows side-by-side. The left window is titled 'vector1' and contains the following data:
1 10
10 2.5
100 -5.6
1000 3.6|

The right window is titled 'vector2' and contains the following data:
5 18
10 3.1
90 46
100 5.2
500 3.4|

The code editor window shows the file 'vectoriRari.c' with the following content:

```
vectoriRari.c

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

Structuri autoreferite

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

```
00 18 int main(int argc, char **argv)
00 19 {
00 20     char numeFisier1[1000],numeFisier2[1000];
00 21     if (argc == 3)
00 22         {strcpy(numeFisier1,argv[1]);strcpy(numeFisier2,argv[2]);}
00 23     else
00 24         {printf("Programul executabil asteapta doua argumente fisiere text \n");exit(0);}
00 25
00 26     struct nod *prim1 = NULL, *ultim1 = NULL;
00 27     construiesteLista(&prim1,&ultim1,numeFisier1);
00 28     afiseazaLista(prim1);
00 29
00 30     struct nod *prim2 = NULL, *ultim2 = NULL;
00 31     construiesteLista(&prim2,&ultim2,numeFisier2);
00 32     afiseazaLista(prim2);
00 33
00 34     struct nod *prim3 = NULL, *ultim3 = NULL;
00 35     suma(prim1,prim2,&prim3,&ultim3);
00 36     printf("*****\n");
00 37     printf("Suma celor doi vectori este: \n");
00 38     afiseazaLista(prim3);
00 39
00 40     printf("*****\n");
00 41     printf("Produsul scalar a celor doi vectori este: %f \n",produsScalar(prim1,prim2));
00 42
00 43     return 0;
00 44 }
```

Structuri autoreferite

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

```
127 void afiseazaLista(struct nod *p)
128 {
129     printf("*****\n");
130     while(p)
131     {
132         printf("%d %f \n", p->poz, p->info);
133         p = p->urm;
134     }
135 }
```

Structuri autoreferite

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

```
01 void construiesteLista(struct nod** p, struct nod** u, char* numeFisier)
02 {
03     FILE *f = fopen(numeFisier,"r");
04     if (f==NULL)
05     {
06         printf("Eroare la deschiderea de fisier \n");
07         exit(0);
08     }
09
10     int poz;
11     float val;
12     while(1)
13     {
14         if (fscanf(f,"%d %f",&poz,&val)==2)
15             adaugare(p, u, val, poz);
16         if (feof(f))
17             return;
18     }
19     return;
20 }
```

Structuri autoreferite

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

```
00 113 void adaugare(struct nod** p,struct nod** u, float val, int poz)
01 114 {
02 115     if (*p == NULL)
03 116     {
04 117         *p = (struct nod *) malloc(sizeof(struct nod));
05 118         (*p)->info = val;
06 119         (*p)->poz = poz;
07 120         (*p)->urm = NULL;
08 121         *u = *p;
09 122     }
10 123     else
11 124     {
12 125         struct nod *c = (struct nod *) malloc(sizeof(struct nod));
13 126         c->info = val;
14 127         c->urm = NULL;
15 128         c->poz = poz;
16 129         (*u)->urm = c;
17 130         *u = c;
18 131     }
19 132 }
```

Structuri autoreferite

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

```
48 void suma(struct nod *prim1,struct nod *prim2,struct nod **prim3,struct nod **ultim3)
49 {
50
51     struct nod *p1, *p2;
52     p1 = prim1;
53     p2 = prim2;
54     while (p1 != NULL && p2 != NULL)
55     {
56         if (p1->poz < p2->poz)
57             { adaugare(prim3, ultim3, p1 -> info, p1 -> poz);
58             p1 = p1 -> urm;}
59         else
60             if (p1->poz > p2->poz)
61             { adaugare(prim3, ultim3, p2 -> info, p2 -> poz);
62             p2 = p2 -> urm;}
63
64         else // (p1->poz == p2->poz)
65             { adaugare(prim3, ultim3, p1 -> info + p2 -> info, p2 -> poz);
66             p1 = p1 -> urm; p2 = p2 -> urm;}
67     }
68
69     while(p1)
70     {   adaugare(prim3, ultim3, p1 -> info, p1 -> poz);
71         p1 = p1 -> urm;
72     }
73
74     while(p2)
75     {
76         adaugare(prim3, ultim3, p2 -> info, p2 -> poz);
77         p2 = p2 -> urm;
78     }
79 }
```

Structuri autoreferite

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

```
137 float produsScalar(struct nod* prim1, struct nod* prim2)
138 {
139     float prodScalar = 0;
140     struct nod *p1, *p2;
141     p1 = prim1;
142     p2 = prim2;
143     while (p1 != NULL && p2 != NULL)
144     {
145         if (p1->poz < p2->poz)
146             p1 = p1 -> urm;
147         else
148             if (p1->poz > p2->poz)
149                 p2 = p2 -> urm;
150             else // (p1->poz == p2->poz)
151             {
152                 prodScalar += (p1 -> info) * (p2 -> info);
153                 p1 = p1 -> urm; p2 = p2 -> urm;
154             }
155     }
156     return prodScalar;
157 }
```

Structuri autoreferite

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

```
*****
1 10.000000
10 2.500000
100 -5.600000
1000 3.600000
*****
5 18.000000
10 3.100000
90 4.600000
100 5.200000
500 3.400000
*****
Suma celor doi vectori este:
*****
1 10.000000
5 18.000000
10 5.600000
90 4.600000
100 -0.400000
500 3.400000
1000 3.600000
*****
Produsul scalar a celor doi vectori este: -21.369999
```