



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,

2016-2017

Cursul 12

Câți dintre voi au completat chestionarele

- 525 de evaluări anul trecut pe semestrul 1
- 949 de evaluări pe semestrul acesta
- VĂ MULTUMIM!!!
- evaluările ajung la cadrele didactice după sesiune ☺

Grupa	Numar Studenti		
101	23		
102	13		
103	19		
104	22		
105	17	Mate	94
131	20		
132	21		
133	20		
134	22		
135	20	INFO13	103
141	22		
142	19		
143	12		
144	23	INFO14	76
151	17		
152	20		
153	14		
154	21	CTI	72

Organizare

1. Examenul scris în sesiune: luni, 6 februarie 2017, ora 13:00. (confirmat)
2. Test de laborator: sâmbătă, 28 ianuarie, între orele 10 – 12 seria 13, orele 12:30 – 14:30 seria 14 (repartizarea pe laboratoarele de la etajul 2 si 3 o vom pune in ziua testării)

Examen

**TE DUCI LA EXAMEN SPERÂND
CĂ TE VOR AJUTA COLEGII.**

Examen

**TE DUCI LA EXAMEN SPERÂND
CĂ TE VOR AJUTA COLEGII.
COINCIDENTĂ: ȘI EI SE GÂNDEAU
LA ACELAȘI LUCRU.**

InfO'Clock - rezultate

- studenții din tabelul alăturat își pot echivala laboratorul + seminarul (maxim 6 puncte) cu punctajul echivalent obținut la InfO'Clock

- laboratorul + seminarul nu se cumulează cu InfO'Clock

Rezultate Evaluare InfO'Clock		Grupa	Punctaj echivalat (maxim 6 puncte)
	Nume Student		
1	Gutu Dan	131	5.4p
2	Pacioianu David	131	5.4p
3	Jelauc Valerian	132	4.2p
4	Paun Bogdan Gabriel	133	5.4p
5	Dranca Constantin	134	5.4p
6	Iordache Stefan	134	5.4p
7	Poesina Eduard	134	3p
8	Anghelache Bogdan	135	3p
9	Banu Alexandru	135	5.4p
10	Burdusa Andrei	135	5.4p
11	Calinescu Valentin Gelu	135	6p
12	Costan Miriam	135	6p
13	Constantin Teodor Claudiu	135	5.4p
14	Dospra Cristian	135	6p
15	Dinu Radu	135	6p
16	Gorneanu Andrei	135	5.4p
17	Ionescu teodor Stelian	135	6p
18	Floreacă Andrei	135	5.4p
19	Lupascu Marian	135	5.4p
20	Marusic Diana	135	4.2p
21	Muntean Radu	135	6p
22	Pandele Maria	135	6p
23	Rîclea Andrei	135	4.2p
24	Slevoacă Stefan	135	5.4p
25	Tarniceru Vlad	135	6p
26	Zugravu Andrei	135	4.2p

Cursul trecut

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

Programa cursului

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

Cursul de azi

1. Funcții cu număr variabil de argumente
2. Programare generică (mai mult la seminar)
3. Recursivitate
4. Subiect de examen

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

- **sintaxă:**

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 cand se încheie operațiile pe lista lp

Funcții cu număr variabil de argumente

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

The screenshot shows a code editor window titled "main.c". The code defines a function "suma" that calculates the sum of a variable number of arguments. It uses the `<stdarg.h>` header and the `va_start`, `va_arg`, and `va_end` macros. The main function demonstrates calling `suma` with 4, 5, 1, 2, 1, 1 and 5, 1, 2, 1, 1, 3 arguments respectively, and printing the results.

```
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;
20     a = suma(4,1,2,1,1);
21     printf("a=%d\n",a);
22     a = suma(5,1,2,1,1,3);
23     printf("a=%d\n",a);
24     return 0;
25 }
```

.
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);  
}
```

- 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 stângă 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` 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

The screenshot shows a code editor window titled "main.c". The code defines a function "suma" that calculates the sum of its arguments using the `va_start`, `va_arg`, and `va_end` macros from the `<stdarg.h>` header. The main function demonstrates calling `suma` with 5 and 8 arguments respectively.

```
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 }
```

Output:

```
a=5
a=8
```

Funcții cu număr variabil de argumente

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

The screenshot shows a code editor window titled "main.c". The code defines a function "suma" that calculates the sum of its arguments using the `va_start`, `va_arg`, and `va_end` macros from the `stdarg.h` header. The main function demonstrates calling `suma` with different argument lists and printing the results.

```
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 sir de numere întregi ce se temină 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:

Funcții cu număr variabil de argumente

❑ exemplul 3: maximul a n numere întregi

```
main.c X
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

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.

```
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;  
}
```

Functii 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("Functiile cu numar variabil ", "de argumente sunt ", "foarte simple!!!", (char *)'\0');
29     printf("%s \n",str);
30     return 0;
31 }
```

P/curs12/vaListExemplu4

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

----- *----- M----- n----- L-----+ ■

Cursul de azi

1. Funcții cu număr variabil de argumente
2. Programare generică (mai mult la seminar)
3. Recursivitate
4. Subiect de examen

Pointeri generici

- pointeri la tipul **void** (pointer generic)
- pot stoca adresa de memorie a unui obiect oarecare
 - dimensiunea zonei de memorie indicate și interpretarea informației conținute, nu sunt definite;
 - nu poate fi dereferențiat

```
voidPointer.C  ✘
1  #include<stdio.h>
2
3  int main()
4  {
5      int i = 6;
6      void *p;
7
8      p = &i;
9      printf("p pointeaza catre o valoare intreaga %d\n", * p);
10
11     return 0;
12 }
```

```
In function 'int main()':
error: 'void*' is not a pointer-to-object type
```

Pointeri generici

- pointeri la tipul **void (pointer generic)**
- pot stoca adresa de memorie a unui obiect oarecare
 - dimensiunea zonei de memorie indicate și interpretarea informației conținute, nu sunt definite;
 - nu poate fi dereferențiat
 - pentru a-l folosi (la derefențiere) trebuie convertit la un alt tip de pointer prin operatorul cast (**tip ***)
 - un singur pointer poate pointa la diferite tipuri de date la momente iferite pe parcursul execuției unui program

Pointeri generici

- pointeri la tipul **void** (pointer generic)
- pot stoca adresa de memorie a unui obiect oarecare
 - pentru a-l folosi (la dereferențiere) trebuie convertit la un alt tip de pointer prin operatorul cast (**tip ***)
 - un singur pointer poate pointa la diferite tipuri de date la momente diferite pe parcursul execuției unui program

The screenshot shows a code editor window titled "voidPointer1.C". The code is as follows:

```
#include<stdio.h>
int main()
{
    int i = 6;
    char c = 'a';
    void *p;

    p = &i;
    printf("p pointeaza catre o valoare intreaga %d\n", *(int*) p);

    p = &c;
    printf("p pointeaza catre un char %c\n", *(char*) p);

    return 0;
}
```

The output of the program is shown in the terminal below the code editor:

```
p pointeaza catre o valoare intreaga 6
p pointeaza catre un char a
```

Programarea generică

- paradigmă de programare în care codăm diferiți algoritmi pentru tipuri de date generice (neinstantțiate).
- exemplu: un algoritm de sortare al unui tablou de numere arată la fel și pentru date de tipul **char, int, float, double**. Putem particulariza funcția de comparare să lucreze cu date de tipul **char, int, float, double**.
- În programare generică folosim **pointeri generici**

Programarea generică

- exemplul 1: scrieți o funcție generică de căutare care să returneze un pointer generic către prima apariție a valorii x în tabloul unidimensional t format din n elemente, fiecare având dimensiunea d octeți, sau pointerul NULL dacă valoarea x nu se găsește în tablou.

```
void* cauta(const void *x,const void *t, int n,int d)
{
    char *p = (char*)t;
    for(int i=0; i<n; i++)
        if(!memcmp(p+i*d,x,d))
    {
        return p+i*d;
    }
    return NULL;
}
```

Programarea generică

□ exemplul 1

The screenshot shows a code editor window titled "cautareGenerica.C". The code implements a generic search function and a main function demonstrating its use.

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

void* cauta(const void *x,const void *t, int n,int d)
{
    char *p = (char*)t;
    for(int i=0; i<n; i++)
        if(!memcmp(x,p+i*d,d))
    {
        return p+i*d;
    }
    return NULL;
}

int main()
{
    int v[] = {10,1,12,3,14,5};
    int x;
    printf("x = ");scanf("%d",&x);
    int *p;
    p = (int*)cauta(&x,v,sizeof(v)/sizeof(int),sizeof(int));
    if(p)
        printf("%d se afla in tabloul v pe pozitia %d \n",x,p-v);

    char w[] = "Orice student intelege programarea generica";
    char ch = 's';
    char *q;
    q = (char*)cauta(&ch,w,sizeof(w)/sizeof(char),sizeof(char));
    if(q)
        printf("%c se afla in tabloul w pe pozitia %d \n",ch,q-w);
    return 0;
}
```

The output window shows the results of the program execution:

```
x = 5
5 se afla in tabloul v pe pozitia 5
s se afla in tabloul w pe pozitia 6
```

Cursul 6: Utilitatea pointerilor la funcții

- se folosesc în **programarea generică**, realizăm apeluri de tip **callback**;
- o funcție C transmisă, printr-un pointer, ca argument unei alte funcții F se numește și funcție **“callback”**, pentru că ea va fi apelată “înapoi” de funcția F
- **exemple:**
 1. int suma(int n, int (*expresie)(int)); **(sumă generică de n numere)**
 2. void qsort(void *adresa,int nr_elemente, int dimensiune_element, int (*cmp)(const void *, const void *)); **(funcția qsort din stdlib.h)**

Cursul 6: Utilitatea pointerilor la funcții

- exemplul 2: vreau să calculez suma

$$S_k(n) = \sum_{i=1}^n i^k$$

$$S_1(n) = 1 + 2 + \dots + n$$

$$S_2(n) = 1^2 + 2^2 + \dots + n^2$$

$$S_k(n) = \sum_{i=1}^n \text{expresie}(i)$$

Folosind pointeri la funcții pot să văd funcția ca o variabilă

Cursul 6: Utilitatea pointerilor la funcții

- exemplul 2: vreau să calculez suma
- implementare elegantă:

$$S_k(n) = \sum_{i=1}^n i^k$$

```
int suma(int n, int (*expresie)(int))
{
    int i,s=0;
    for(i=1;i<=n;i++)
        s = s + expresie(i);
    return s;
}
```

```
int expresie1(int x)
{
    return x;
}
```

```
int expresie2(int x)
{
    return x*x;
}
```

Cursul 6: Utilitatea pointerilor la funcții

□ exemplul 2: vreau să calculez suma

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int suma(int n, int (*expresie)(int))
5 {
6     int i,s=0;
7     for(i=1;i<=n;i++)
8         s = s + expresie(i);
9     return s;
10}
11
12 int expresie1(int x)
13 {
14     return x;
15 }
16
17 int expresie2(int x)
18 {
19     return x*x;
20 }
21
22 int main()
23 {
24
25     int S1 = suma(5,expresie1);
26     printf("S1 = %d\n",S1);
27     int S2 = suma(5,expresie2);
28     printf("S2 = %d\n",S2);
29     return 0;
30 }
```

$$S_k(n) = \sum_{i=1}^n i^k$$

S1 = 15
S2 = 55

Process returned 0 (0x0) execution time : 0.0

Cursul 6: Utilitatea pointerilor la funcții

- **funcția qsort din stdlib.h folosită pentru sortarea unui vector/tablou.** Antetul lui qsort este:

```
void qsort (void *adresa, int nr_elemente, int dimensiune_element,  
int (*cmp) (const void *, const void *))
```

- adresa = pointer la adresa primului element al tabloului ce urmeaza a fi sortat
(pointer generic – nu are o aritmetică inclusă)
- nr_elemente = numarul de elemente al vectorului
- dimensiune_element = dimensiunea in octeți a fiecărui element al tabloului
(char = 1 octet, int = 4 octeți, etc)
- **cmp – funcția de comparare a două elemente**

Programare generică - qsort

- funcția qsort din stdlib.h folosită pentru sortarea unui vector/tablou. Antetul lui qsort este:

```
void qsort (void *adresa, int nr_elemente, int dimensiune_element,  
int (*cmp) (const void *, const void *))
```

```
int cmp(const *a, const *b)
```

adresele a două elemente din tablou

Cmp este o funcție generică comparator, compară 2 elemente de orice tip. Întoarce:

un număr < 0 dacă vrem a la stânga lui b

un număr >0 dacă vrem a la dreapta lui b

0, dacă nu contează

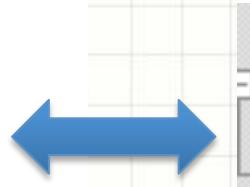
Programare generică - qsort

- funcția qsort din stdlib.h folosită pentru sortarea unui vector/tablou. Antetul lui qsort este:

```
void qsort (void *adresa, int nr_elemente, int dimensiune_element,  
int (*cmp) (const void *, const void *))
```

Exemplu de funcție cmp pentru sortarea unui vector de numere întregi:

```
int cmp(const void *a, const void *b)  
{  
    int va, vb;  
    va = *(int*)a;  
    vb = *(int*)b;  
    if(va < vb) return -1;  
    if(va > vb) return 1;  
    return 0;  
}
```



```
int cmp(const void *a, const void *b)  
{  
    return *(int*)a - *(int*)b;  
}
```

Programare generică - qsort

- funcția qsort din stdlib.h folosită pentru sortarea unui vector/tablou. Antetul lui qsort este:

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int cmp(const void *a, const void *b)
5 {
6     return *(int*)a - *(int*)b;
7 }
8
9 int main()
10 {
11     int v[] = {0,5,-6,9, 7, 12, 8, 7, 4};
12     qsort(v,9,sizeof(int),cmp);
13     int i;
14     for(i=0;i<9;i++)
15         printf("%d ",v[i]);
16     printf("\n");
17
18     return 0;
19 }
```

```
-6 0 4 5 7 7 8 9 12
```

```
Process returned 0 (0x0)   execution time : 0.00 secs ...
```

Programarea generică

- exemplul 4: scrieți o funcție generică de căutare care să returneze un pointer generic către prima apariție a valorii x în tabloul unidimensional t format din n elemente, fiecare având dimensiunea d octeți, sau pointerul NULL dacă valoarea x nu se găsește în tablou. Folosiți o funcție comparator cmp cu antetul

```
int (cmpValori*)(const void*, const void *)
```

care returnează valoarea 1 dacă valorile aflate la adresele primite ca parametri sunt egale sau 0 în caz contrar.
Explicitarea funcției concrete de comparație se va face în apel.

Vom considera două cazuri: compararea a două valori întregi, respectiv compararea a două siruri de caractere.

Programarea generică

□ exemplul 4:

```
void* cauta(const void *x,const void *t, int n,int d, int (*f)(const void*,const void*))  
{  
    char *p = (char*)t;  
    for(int i=0; i<n; i++)  
        if(f(x,p+i*d))  
        {  
            return p+i*d;  
        }  
    return NULL;  
  
int cmpInt(const void* a, const void* b)  
{  
    return *((int*)a) == *((int*)b) ;  
}  
  
int cmpChar(const void* a, const void* b)  
{  
    return (*((char*)b) == *((char*)a));  
}
```

cautareGenerica1.C

```
1 #include <stdio.h>
2 #include <memory.h>
3
4 void* cauta(const void *x,const void *t, int n,int d, int (*f)(const void*,const void*))
5 {
6     char *p = (char*)t;
7     for(int i=0; i<n; i++)
8         if(f(x,p+i*d))
9         {
10             return p+i*d;
11         }
12     return NULL;
13 }
14
15 int cmpInt(const void* a, const void* b)
16 {
17     return *((int*)a) == *((int*)b) ;
18 }
19
20 int cmpChar(const void* a, const void* b)
21 {
22     return (*((char*)b) == *((char*)a));
23 }
24
25
26 int main()
27 {
28     int v[] = {10,1,12,3,14,5};
29     int x;
30     printf("x = ");scanf("%d",&x);
31     int *p;
32     p = (int*)cauta(&x,v,sizeof(v)/sizeof(int),sizeof(int),cmpInt);
33     if(p)
34         printf("%d se afla in tabloul v pe pozitia %d \n",x,p-v);
35
36     char w[] = "Orice student intelege programarea generica";
37     char ch = 's';
38     char *q;
39     q = (char*)cauta(&ch,w,sizeof(w)/sizeof(char),sizeof(char),cmpChar);
40     if(q)
41         printf("%c se afla in tabloul w pe pozitia %d \n",ch,q-w);
42
43     return 0;
44 }
```

```
x = 12
12 se afla in tabloul v pe pozitia 2
s se afla in tabloul w pe pozitia 6
```

Cursul de azi

1. Funcții cu număr variabil de argumente
2. Programare generică (mai mult la seminar)
3. Recursivitate
4. Subiect de examen

Recursivitate

- recursivitate = capacitatea unei funcții de a se auto-apela
- studiem mecanismul recursivității, ce se întâmplă când o funcție se auto-apelează (nu studiem recursivitatea ca tehnică de programare)
- corespondentul din matematică al recursivității este recurența

Recursivitate - exemplu

- Definiția factorialului unui număr natural n

$$n! = \begin{cases} 1, & \text{dacă } n=0 \\ n*(n-1)!, & \text{dacă } n>=1 \end{cases}$$

$$\begin{aligned} 4! &= 4 * 3! = 4 * 6 = 24 \\ 3! &= 3 * 2! = 3 * 2 = 6 \\ 2! &= 2 * 1! = 2 * 1 = 2 \\ 1! &= 1 * 0! = 1 * 1 = 1 \\ 0! &= 1 \end{aligned}$$

Adâncimea recursivității

Altă definiție:

$$n! = \frac{(n+1)!}{n+1}$$

Definiție inutilă, nu se oprește relația de recurență.

Recursivitate - exemplu

- Definiția factorialului unui număr natural n

$$n! = \begin{cases} 1, & \text{dacă } n=0 \\ n*(n-1)!, & \text{dacă } n>=1 \end{cases}$$

```
int factorial(int n)
{
    if (n==0) return 1; //conditia de oprire
    return n*factorial(n-1); //recursivitate
}
```

- ce se întâmplă în stivă pentru apelul `t = factorial(4)` ?
- în stivă, fiecare apel se aşează deasupra apelului precedent.
- se salvează un context de apel.

Recursivitate - stivă

- ce se întâmplă în stivă pentru apelul $t = \text{factorial}(4)$?
- în stivă, fiecare apel se aşează deasupra apelului precedent.
- se salvează un context de apel:
 1. adresa de revenire
 2. copii ale valorile parametrilor efectivi
 3. valorile variabilelor locale
 4. copii ale regiștrilor
 5. valoarea returnată

$t = \text{factorial}(4);$

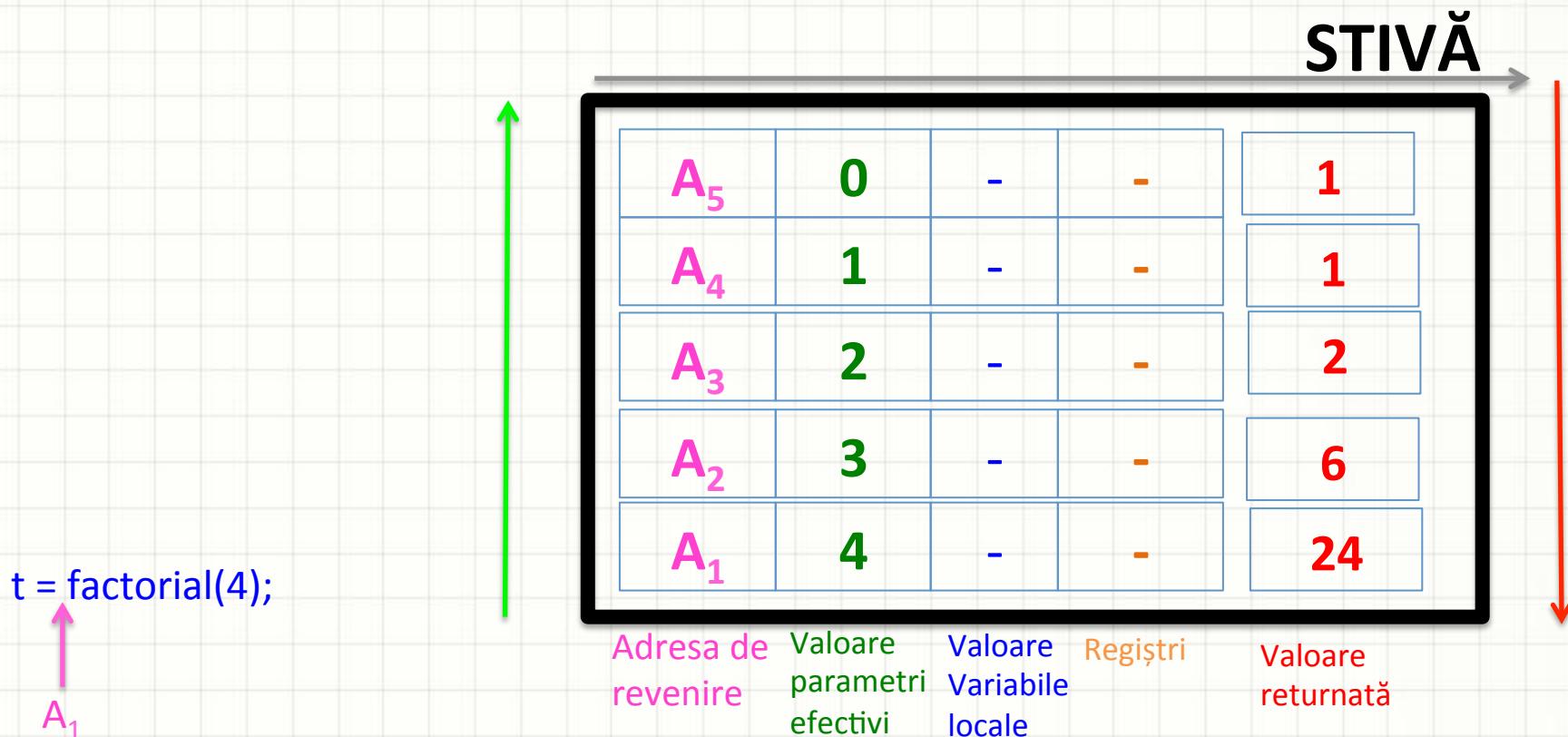
A₁

STIVĂ				
Adresa de revenire	Valoare parametri efectivi	Valoare Variabile locale	Regiștri	Valoare returnată
A ₅	0	-	-	1
A ₄	1	-	-	1
A ₃	2	-	-	2
A ₂	3	-	-	6
A ₁	4	-	-	24

Recursivitate - exemplu

Trei faze în execuție:

1. expandarea recursivității (crește stiva)
2. condiția de oprire a recursivității
3. restaurarea stivei (a contextului e apel)



Recursivitate - exemplu

- Citesc numere întregi până la întâlnirea lui 0 și le afișez în ordine inversă.

```
main.c ✘
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void citesteAfiseaza()
5 {
6     int x;
7     scanf("%d",&x);
8     if (x!=0)
9         citesteAfiseaza();
10    printf("%d ",x);
11 }
12
13 int main()
14 {
15     citesteAfiseaza();
16     return 0;
17 }
```

```
10
20
30
40
0
0 40 30 20 10
Process returned 0 (0x0)    execution time : 5.713
Press ENTER to continue.
```

Ce se întâmplă dacă în loc de `int x` am `static int x` (linia 6)?

Recursivitate - exemplu

2. Citesc numere întregi până la întâlnirea lui 0 și le afișez în ordine inversă.

```
main.c X
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void citesteAfiseaza()
5 {
6     static int x;
7     scanf("%d",&x);
8     if (x!=0)
9         citesteAfiseaza();
10    printf("%d ",x);
11 }
12
13 int main()
14 {
15     citesteAfiseaza();
16     return 0;
17 }
```

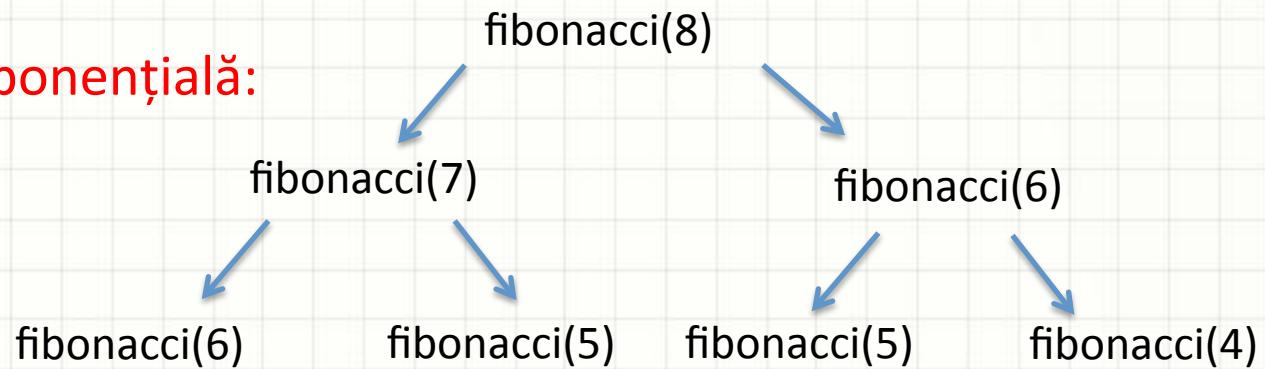
```
10
20
30
40
0
0 0 0 0 0
Process returned 0 (0x0)    execution time : 5.964
Press ENTER to continue.
```

Recursivitate - exemplu

- ❑ sirul lui Fibonacci: $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$, dacă $n \geq 2$

```
int fibonacci(int n)
{
    if (n<=1)
        return n;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

Complexitate exponențială:



Recursivitate - exemplu

3. Sirul lui Fibonacci: $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$, dacă $n \geq 2$

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

F_20 = 6765

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

Recursivitate - exemplu

3. Sirul lui Fibonacci: $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$, dacă $n \geq 2$

```
main.c ✘
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int fibonacci(int n)
5 {
6     static int nrApeluri = 0;
7     printf("Am ajuns la apelul %d\n", ++nrApeluri);
8
9     if(n<=1)
10        return n;
11    return fibonacci(n-2) + fibonacci(n-1);
12 }
13
14 int main()
15 {
16     int n=20;
17     printf("F_%d = %d\n", n, fibonacci(n));
18     return 0;
19 }
```

```
Am ajuns la apelul 21885
Am ajuns la apelul 21886
Am ajuns la apelul 21887
Am ajuns la apelul 21888
Am ajuns la apelul 21889
Am ajuns la apelul 21890
Am ajuns la apelul 21891
F_20 = 6765

Process returned 0 (0x0)   execution time...
```

Putem număra câte apeluri se efectuează.

Recursivitate - exemplu

3. Sirul lui Fibonacci: $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$, dacă $n \geq 2$

Implementare iterativă

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int fibonacci(int n)
5 {
6     if(n<=1)
7         return n;
8     int a=0,b=1,c,i;
9     for(i=2;i<=n;i++)
10    {
11        c = a+b;
12        a = b;
13        b = c;
14    }
15    return b;
16 }
17
18 int main()
19 {
20     int n=20;
21     printf("F_%d = %d\n",n,fibonacci(n));
22     return 0;
23 }
```

```
F_20 = 6765
Process returned 0 (0x0)  execution time ...
Press ENTER to continue.
```

Cursul de azi

1. Funcții cu număr variabil de argumente
2. Programare generică (mai mult la seminar)
3. Recursivitate
4. Subiect de examen

Subiect de la seria de Mate

Subiectul 1. (1 punct)

Ce afișează instrucțiunea de mai jos? Justificați.

```
printf("%d %d\n", (11 && 22), (11 & 22));
```

Subiectul 2. (1 punct)

Funcția *verificaPrim* de mai jos este scrisă pentru a verifica dacă un număr natural n este prim. Funcția trebuie să returneze 1 dacă n este prim sau 0 altfel. Funcția *verificaPrim* de mai jos este scrisă greșit.

```
int verificaPrim(int n)
{
    int i;
    if (n<=1)
        return 1;
    if(n%2==0)
        return 0;
    for(i=3;i<n;i=i+2)
        if(n%i == 0)
            return 0;
    return 1;
}
```

- (a) dați exemplu de o intrare n pentru care funcția scrisă returnează corect rezultatul. Justificați. (0.25 puncte)
- (b) dați exemplu de o intrare n pentru care funcția scrisă returnează incorect rezultatul. Justificați. (0.25 puncte)
- (c) scrieți varianta corectă a funcției *verificaPrim* (puteți porni de la actuala variantă sau puteți rescrie în totalitate funcția). (0.5 puncte)

Subiectul 3. (1 punct)

Ordonați crescător funcțiile de mai jos pe baza creșterii lor asimptotice, precizând de fiecare dată funcțiile cu același ordin de creștere:

$$f_1(n) = 2^n, f_2(n) = n^3 + 3, f_3(n) = \sqrt{n}, f_4(n) = 3^n + 5, f_5(n) = 4^n$$

$$f_6(n) = 10 \cdot \log_2(n), f_7(n) = n \cdot \sqrt{n}, f_8(n) = 2 \cdot n \cdot \log_2(n), f_9(n) = \ln(n), f_{10}(n) = 4 \cdot n + 5.$$

Subiect de la seria de Mate

Subiectul 4. (1 punct)

```
#include <stdio.h>
#include <stdlib.h>
int f1(int a) {return 5*a;}
int f2(int b) {return b-2;}
int f3(int *c) {return f2(f1(*c));}
void main()
{
    int a=0,b=1,c=2;
    printf("%d %d %d \n",f1(b),f2(c),f3(&a));
}
```

Analizați programul de alături. Dacă programul este corect spuneți ce va afișa argumentând răspunsul. Dacă programul nu este corect explicați de unde provine eroarea.

Subiectul 5. (2 puncte)

Scrieți o funcție care primește ca parametri numele unui tablou unidimensional de numere întregi și numărul de elemente al acestuia și returnează suma maximă a unei secvențe (șir de numere consecutive) formate doar din valori strict pozitive ale tabloului primit ca parametru. Precizați și argumentați complexitatea funcției.

Exemplu: Pentru vectorul (1, 2, 3, -1, 0, 4, 5, -2) suma maximă a unei secvențe este 9.

Subiect de la seria de Mate

Subiectul 6. (2 puncte)

Scrieți o funcție care primește ca parametru un număr natural nenul n și returnează un tablou bidimensional pătratic, alocat dinamic, având următoarea formă:

pentru $n = 3$

0	0	0
0	1	1
0	1	2

pentru $n = 4$

0	0	0	0
0	1	1	1
0	1	2	2
0	1	2	3

Subiectul 7. (2 puncte)

Fișierul *cuvinte.in* conține pe prima linie un număr natural n nenul iar pe următoarele linii un text în care cuvintele sunt despărțite prin spații. Realizați un program care să scrie în fișierul text *cuvinte.out* toate cuvintele de lungime n din fișierul *cuvinte.in* sau mesajul "Imposibil" dacă în fișierul de intrare nu există niciun cuvânt de lungime n .

Subiect de la seria de Info

Subiectul nr. 1 (2 puncte)

- a) Scrieți o funcție care să citească numărul de elemente ale unui tablou unidimensional, să aloce dinamic tabloul respectiv și apoi să citească valorile elementelor sale. (0.5 puncte)
- b) Scrieți o funcție care să afișeze elementele unui tablou unidimensional de numere reale. (0.5 puncte)
- c) Scrieți o funcție care să calculeze poziția pe care se află valoarea minimă a unei secvențe cuprinse între doi indici i și j ($0 \leq i \leq j < n$) dintr-un tablou unidimensional format din n numere reale. (0.5 puncte)
- d) Scrieți un program care, folosind apeluri utile ale funcțiilor definite anterior, citește de la tastatură un tablou unidimensional format din n numere reale, îl sortează și apoi îl afișează. Precizați și argumentați complexitatea programului obținut. (0.5 puncte)

Observație: Nu este permisă utilizarea unor variabile globale!

Subiectul nr. 2 (1 punct)

Scrieți o funcție care primește ca parametru un număr natural nenul n și returnează un tablou bidimensional pătratic, alocat dinamic, având următoarea formă (pentru $n = 4$):

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 2 & 2 \\ 0 & 1 & 2 & 3 \end{matrix}$$

Subiect de la seria de Info

Subiectul nr. 3 (2 puncte)

Fișierul *cuvinte.in* conține pe prima linie un număr natural nenul *n*, iar pe următoarele linii un text în care cuvintele sunt despărțite prin spații și semnele de punctuație uzuale. Scrieți un program care să scrie în fișierul text *cuvinte.out* toate cuvintele de lungime *n* din fișierul *cuvinte.in* sau mesajul "Imposibil!" dacă în fișierul de intrare nu există nici un cuvânt cu proprietatea cerută.

Subiectul nr. 4 (2 puncte)

- a) Scrieți o funcție generică de căutare cu următorul antet:

```
void * cautare(const void * x, const void * t, int n, int d, int (* cmpValori)(const void *, const void *))
```

Funcția trebuie să returneze un pointer generic către prima apariție a valorii *x* în tabloul unidimensional *t* format din *n* elemente, fiecare având dimensiunea *d* octeți sau pointerul *NULL* dacă valoarea *x* nu se găsește în tablou. Funcția comparator *cmpValori* se consideră că returnează 1 dacă valorile aflate la adresele primite ca parametrii sunt egale sau 0 în caz contrar. (1 punct)

- b) Scrieți un program care citește de la tastatură un număr întreg *r*, un tablou unidimensional *v* format din *n* numere întregi și afișează, folosind apeluri utile ale funcției *cautare*, toate pozițiile pe care apare numărul *x* în tabloul *v* sau mesajul "Valoare inexistentă!" dacă numărul *x* nu se găsește în tablou. (1 punct)

Subiect de la seria de Info

Subiectul nr. 5 (2 puncte)

- a) Definiți o structură *Student* care să permită memorarea numelui, notelor, mediei și grupei corespunzătoare unui student. Scrieți o funcție care să calculeze mediile celor n studenți ale căror date sunt memorate într-un tablou unidimensional t cu elemente de tip *Student*. (1 punct)
- b) Folosind funcția *qsort* din biblioteca *stdlib.h*, sortați elementele unui tablou unidimensional t format din n elemente de tip *Student* în ordinea descrescătoare a mediilor, iar în cazul unor medii egale studenții respectivi se vor ordona alfabetic. Implementați funcția comparator corespunzătoare și scrieți apelul funcției *qsort*. (0.5 puncte)
- c) Scrieți o funcție care să scrie într-un fișier binar informațiile despre studenții dintr-o grupă. Funcția va avea ca parametrii un tablou unidimensional t cu n elemente de tip *Student* și un număr natural nenul g reprezentând numărul unei grupe. Numele fișierului binar va fi *grupa_g.bin*. (0.5 puncte)