



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,

2018-2019

Cursul 13

Evaluarea activităților didactice pe semestrul 1

- studenții au posibilitatea să evaluateze activitățile cadrelor didactice

- evaluare = completarea unui **chestionar** pentru fiecare disciplină (curs/seminar/laborator) ce conține **11 întrebări = 3-5 minute**

- evaluările au caracter anonim, fiecare student se va loga folosind un token nenominal de unică folosință primit de la secretariat de către șeful de grupă și distribuit apoi membrilor grupei.

- evaluarea se desfășoară în **perioada 9 – 20 ianuarie**

- concluziile din evaluarea de anul trecut vor fi făcute publice într-un raport (cel mai probabil în ianuarie)

Chestionarul pentru curs

1. Aprecieri asupra materiei predate la curs

1.1. Dificultatea materiei a fost corespunzatoare nivelului meu de intelegerere

- a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

1.2. Resursele de invatare puse la dispozitie au fost accesibile si utile

- a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

1.3. Am inteles utilitatea materiei in pregatirea mea profesionala

- a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

2. Aprecieri asupra profesorului de curs

2.1 Ne-a prezentat de la inceputul semestrului obiectivele cursului si cerințele de evaluare

- a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

2.2 A respectat pe parcursul semestrului obiectivele anunțate

- a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

2.3 Expunerea sa a fost clara, sistematica si coerenta

- a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

2.4 A fost entuziasmat si a prezentat materia într-o manieră atractivă

- a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

2.5 A prezentat materia într-o manieră interactivă si a fost disponibil pentru intrebări

- a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

2.6 A fost punctual și a utilizat eficient timpul programat activității de curs

- a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

2.7 A cultivat respectul reciproc intre profesor si studenti

- a. Dezacord total; b. Dezacord parțial; c. Neutru; d. Acord parțial; e. Acord total.

3. Ce mi-a plăcut / ce nu mi-a plăcut / ce aş îmbunătăți

.....

FMI NO STRESS – duminică, 10⁰⁰ - 15⁰⁰

[Despre...](#) [Probleme](#) [Clasament](#)

Concursul *FMI No Stress* este organizat de membri ai [Facultății de Matematică și Informatică](#) din cadrul Universității din București.

Despre concurs

Te invităm să iei parte la un concurs organizat de membrii ai [Facultatati de Matematica si Informatica, Universitatea Bucuresti](#). **FMI No Stress** este un concurs cu dificultate usoara spre medie si este destinat in primul rand incepatorilor in domeniul algoritmicii. Concurrentii vor avea de rezolvat 8-10 probleme de natură algoritmică, cu grad variat de dificultate. Proba va avea loc **Duminica, 13 ianuarie, ora 10⁰⁰** și se va desfășura pe durata a 5h. Vei primi feedback complet la toate submisiile proprii pe durata concursului, astfel încât vei putea ști în orice moment ce punctaj ai obținut. Vă așteptăm în număr cât mai mare, succes tuturor!

Organizatori

Subiectele vor fi propuse si pregatite de:

-  Andi Arnautu • [andrei.arnautu](#)
-  Chiriac Andrei • [chiriacandrei25](#)
-  Constantinescu Andrei-Costin • [Andrei1998](#)
-  Popa Andrei • [andreiiii](#)
-  Bogdan Ciobanu • [bciobanu](#)
-  Dobre Bogdan Mihai • [dobrebogdan](#)
-  Iordache Ioan-Bogdan • [iordache.bogdan](#)
-  Eugenie Daniel Posdarascu • [eudanip](#)
-  Livia Magureanu • [livlivli](#)
-  Lucian Bicsi • [retrograd](#)
-  Maria Pandele • [Smaranda](#)
-  Radu Muntean • [heracle](#)
-  Ionescu Teodor • [teoionescu](#)

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, recursivitate
- Vrem să afișăm situația voastră (notă laborator + bonus seminar) până duminică, 27 ianuarie, cu o săptămână înainte de examenul final din 4 februarie 2019

Cod asemănător?

Regulament de etică

Tema de proiect este individuală. Prezentarea care însoțește proiectul (expusă la curs) detaliază toți pașii implementării. Toate întrebările legate de proiect le adresați laboranților sau celor care țin cursul. Fișierele sursă primite vor fi verificate cu un instrument care detectează cod asemănător. Proiectele care seamănă între ele vor fi notate cu nota 1.

- proiectele cu cod evident asemănător vor fi notate cu nota 1
- ce înseamnă evident asemănător?
- porțiuni lungi de cod foarte asemănătoare, cod specific foarte asemănător

Cod evident asemănător?

142-153	83-92
92-98	64-69
319-330	174-182
55-63	97-104
112-115	126-129

<pre>pcrt = img.pixels[(img.height - i - 1) * img.width + j]; fwrite(pcrt.bytesPixel, 3, 1, fimg); } for(j = 0; j < img.padding; j++) fwrite(&aux, 1, 1, fimg); } fclose(fimg); } //transforma o image rgb in grayscale ImageRGB rgb2gray(ImageRGB img) { ImageRGB gray = initImage(img); Pixel pcrt; int i,j; unsigned int aux; for(i = 0; i < gray.height; i++) { for(j = 0; j < gray.width; j++) { pcrt = img.pixels[(img.height - i - 1) * img.width + j]; aux = 0.299*pcrt.PixelRGB.red + 0.587*pcrt.PixelRGB.green + 0.114*pcrt.PixelRGB.blue; pcrt.bytesPixel[0] = pcrt.bytesPixel[1] = pcrt.bytesPixel[2] = aux; gray.pixels[(gray.height - i - 1) * gray.width + j] = pcrt; } } }</pre>	<pre>{ fread(&(img.vector[img.W * i + j].b), 1, 1, fin); fread(&(img.vector[img.W * i + j].g), 1, 1, fin); fread(&(img.vector[img.W * i + j].r), 1, 1, fin); } for(k = 0; k < img.dim_padding; k++) fread(&c, 1, 1, fin); } fclose(fin); return img; } unsigned int xorshift (unsigned int numar_generat_anterior) { unsigned int numar_curent = numar_generat_anterior; numar_curent = numar_curent ^ numar_curent << 13; numar_curent = numar_curent ^ numar_curent >> 17; numar_curent = numar_curent ^ numar_curent << 5; return numar_curent; } pixel pixel_xor_constanta (pixel pixell, unsigned int x)</pre>
--	--

NU

Recapitulare – cursul trecut (2018)

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

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. Programare generică
2. Recursivitate

Seminar 7 + Laborator 12

14



Laborator12 → x2 X
Seminar7 → x2 X

SEMINAR 7 FUNCȚII CU NUMĂR VARIABIL DE ARGUMENTE. PROGRAMARE GENERICĂ. RECURSIVITATE

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 elemente (numere de tipul **char**, **int**, **float**, **double**, structuri) este invariant la tipurile de date.
- funcția **qsort** din **stdlib.h**
- putem particulariza funcția de comparare să lucreze cu date de tipul **char**, **int**, **float**, **double** sau tipuri de date derivate(structuri).
- În programare generică folosim **pointeri generici (void *)**
- asigură o mai mare generalitate în raport cu tipul de date a variabilelor procesate în cadrul functiilor

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 întrucât nu are aritmetică indusă

```
pointeriGenerici1.c      x
1 #include<stdio.h>
2
3 int main()
4 {
5     int i = 6;
6     void *p;
7
8     p = &i;
9
10    printf("p pointeaza catre un int cu valoare = %d\n", *p);
11
12    return 0;
13 }
```

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

```
pointeriGenerici2.c
1 #include<stdio.h>
2
3 int main()
4 {
5     int i = 6;
6     char c = 'a';
7     void *p;
8
9     p = &i;
10    printf("p pointeaza catre un int cu valoarea = %d\n", *(int*)p);
11
12    p = &c;
13    printf("p pointeaza catre un char cu valoarea = %c\n", *(char*)p);
14
15    return 0;
16 }
```

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

```
pointeriGenerici2.c
1 #include<stdio.h>
2
3 int main()
4 {
5     int i = 6;
6     char c = 'a';
7     void *p;
8
9     p = &i;
10    printf("p pointeaza catre un int cu valoarea = %d\n", *(int*)p);
11
12    p = &c;
13    printf("p pointeaza catre un char cu valoarea = %c\n", *(char*)p);
14
15    return 0;
16 }
```

Bogdan-Alexes-MacBook-Pro:curs13 bogdan\$ gcc pointeriGenerici2.c
Bogdan-Alexes-MacBook-Pro:curs13 bogdan\$./a.out
p pointeaza catre un int cu valoarea = 6
p pointeaza catre un char cu valoarea = a

Cursul 7: 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. `void qsort(void *adresa,int nr_elemente, int dimensiune_element, int (*cmp)(const void *, const void *)) ;`
 2. `void* bsearch(const void *cheie, const void* adresa, int nr_elemente, int dimensiune_element, int (*cmp) (const void *, const void *))`

Funcția generică de sortare qsort

- **funcția qsort din stdlib.h folosită pentru sortarea unui 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 urmează a fi sortat
(pointer generic – nu are o aritmetică inclusă)
- nr_elemente = numărul de elemente al vectorului
- dimensiune_element = dimensiunea în octeți a fiecărui element al tabloului
(char = 1 octet, int = 4 octeți, etc)
- **cmp – funcția de comparare a două elemente**

Funcția generică de sortare qsort

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

adresele a două elemente din tablou

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

- un număr < 0 dacă vrem elementul de la adresa **a** la stânga (înaintea) elementului de la adresa **b**
- un număr >0 dacă vrem elementul de la adresa **a** la dreapta (după) elementului de la adresa **b**
- 0, dacă nu contează

Functia comparator pentru nr. intregi

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

Exemplu de functie cmp pentru sortarea in ordine crescatoare a unui vector de numere intregi:

```
int cmp1(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 cmp2(const void* a, const void* b)  
{  
    return *(int *)a - *(int *)b;  
}
```

exempluqsort1.c

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<limits.h>
4
5 int cmp1(const void* a, const void* b)
6 {
7     int va, vb;
8     va = *(int *)a;
9     vb = *(int *)b;
10    if(va < vb) return -1;
11    if(va > vb) return 1;
12    return 0;
13 }
14
15 int cmp2(const void* a, const void* b)
16 {
17     return *(int *)a - *(int *)b;
18 }
19
20 int main()
21 {
22     int i;
23     int v[5] = {0, 5, -6, 9, 7 };
24     qsort(v,5,sizeof(int),cmp1);
25     for(i = 0; i< 5; i++)
26         printf("%d ", v[i]);
27     printf("\n");
28
29     int w[5] = {0, 5, -6, 9, 7 };
30     qsort(w,5,sizeof(int),cmp2);
31     for(i = 0; i < 5; i++)
32         printf("%d ", w[i]);
33     printf("\n");
34
35     return 0;
36 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ gcc exempluqsort1.c
[Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ ./a.out
-6 0 5 7 9
-6 0 5 7 9
```

exempluqsort2.c

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<limits.h>
4
5 int cmp1(const void* a, const void* b)
6 {
7     int va, vb;
8     va = *(int *)a;
9     vb = *(int *)b;
10    if(va < vb) return -1;
11    if(va > vb) return 1;
12    return 0;
13 }
14
15 int cmp2(const void* a, const void* b)
16 {
17     return *(int *)a - *(int *)b;
18 }
19
20 int main()
21 {
22     int i;
23     int v[5] = {INT_MIN+5, INT_MIN, INT_MAX-4, INT_MAX-1, INT_MAX};
24     qsort(v,5,sizeof(int),cmp1);
25     for(i = 0; i < 5; i++)
26     {
27         printf("%d ", v[i]);
28     }
29     printf("\n");
30
31     int w[5] = {INT_MIN+5, INT_MIN, INT_MAX-4, INT_MAX-1, INT_MAX};
32     qsort(w,5,sizeof(int),cmp2);
33     for(i = 0; i < 5; i++)
34     {
35         printf("%d ", w[i]);
36     }
37     printf("\n");
38
39     return 0;
40 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ gcc exempluqsort2.c
[Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ ./a.out
-2147483648 -2147483643 2147483643 2147483646 2147483647
2147483643 2147483646 2147483647 -2147483648 -2147483643
```

exempluqsort2.c

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<limits.h>
4
5 int cmp1(const void* a, const void* b)
6 {
7     int va, vb;
8     va = *(int *)a;
9     vb = *(int *)b;
10    if(va < vb) return -1;
11    if(va > vb) return 1;
12    return 0;
13 }
14
15 int cmp2(const void* a, const void* b)
16 {
17     return *(int *)a - *(int *)b;
18 }
19
20 int main()
21 {
22     int i;
23     int v[5] = {INT_MIN+5, INT_MIN, INT_MAX-4, INT_MAX-1, INT_MAX};
24     qsort(v,5,sizeof(int),cmp1);
25     for(i = 0; i < 5; i++)
26     {
27         printf("%d ", v[i]);
28     }
29     printf("\n");
30
31     int w[5] = {INT_MIN+5, INT_MIN, INT_MAX-4, INT_MAX-1, INT_MAX};
32     qsort(w,5,sizeof(int),cmp2);
33     for(i = 0; i < 5; i++)
34     {
35         printf("%d ", w[i]);
36     }
37     printf("\n");
38
39     return 0;
40 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ gcc exempluqsort2.c
[Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ ./a.out
2147483643 2147483643 2147483643 2147483643 2147483643
2147483643 2147483646 2147483647 -2147483648 -2147483643
```

Functia comparator pentru nr. intregi

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

Exemplu de funcție cmp pentru sortarea în ordine crescătoare a unui vector de numere întregi:

```
int cmp1(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 cmp2(const void* a, const void* b)  
{  
    return *(int *)a - *(int *)b;  
}
```

Functia cmp2 poate face underflow sau overflow

Funcția comparator pentru nr. reale

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

Exemplu de funcție cmp pentru sortarea în ordine crescătoare a unui vector de numere reale:

```
int cmp1(const void* a, const void* b)  
{  
    double va, vb;  
    va = *(double *)a;  
    vb = *(double *)b;  
    if(va < vb) return -1;  
    if(va > vb) return 1;  
    return 0;  
}
```

```
int cmp2(const void* a, const void* b)  
{  
    return *(double *)a - *(double *)b;  
}
```

Functia cmp2 poate face underflow sau overflow

Alt bug?

exempluqsort3.c

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<limits.h>
4
5 int cmp1(const void* a, const void* b)
6 {
7     double va, vb;
8     va = *(double *)a;
9     vb = *(double *)b;
10    if(va < vb) return -1;
11    if(va > vb) return 1;
12    return 0;
13 }
14
15 int cmp2(const void* a, const void* b)
16 {
17     return *(double *)a - *(double *)b;
18 }
19
20 int main()
21 {
22     int i;
23     double v[5] = {0.2, 0.1, 0.5, -0.2,-0.3};
24     qsort(v,5,sizeof(double),cmp1);
25     for(i = 0; i< 5; i++)
26         printf("%f ", v[i]);
27     printf("\n");
28
29     double w[5] = {0.2, 0.1, 0.5, -0.2,-0.3};
30     qsort(w,5,sizeof(double),cmp2);
31     for(i = 0; i < 5; i++)
32         printf("%f ", w[i]);
33     printf("\n");
34
35     return 0;
36 }
```

```
Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ gcc exempluqsort3.c
Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ ./a.out
-0.300000 -0.200000 0.100000 0.200000 0.500000
0.200000 0.100000 0.500000 -0.200000 -0.300000
```

exempluqsort3.c

```
00 #include<stdio.h>
01 #include<stdlib.h>
02 #include<limits.h>
03
04 int cmp1(const void* a, const void* b)
05 {
06     double va, vb;
07     va = *(double *)a;
08     vb = *(double *)b;
09     if(va < vb) return -1;
10     if(va > vb) return 1;
11     return 0;
12 }
13
14 int cmp2(const void* a, const void* b)
15 {
16     return *(double *)a - *(double *)b;
17 }
18
19 int main()
20 {
21     int i;
22     double v[5] = {0.2, 0.1, 0.5, -0.2,-0.3};
23     qsort(v,5,sizeof(double),cmp1);
24     for(i = 0; i< 5; i++)
25     {
26         printf("%f ", v[i]);
27     }
28     printf("\n");
29
30     double w[5] = {0.2, 0.1, 0.5, -0.2,-0.3};
31     qsort(w,5,sizeof(double),cmp2);
32     for(i = 0; i < 5; i++)
33     {
34         printf("%f ", w[i]);
35     }
36 }
```

```
Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ gcc exempluqsort3.c
Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ ./a.out
-0.300000 -0.200000 0.100000 0.200000 0.500000
0.200000 0.100000 0.500000 -0.200000 -0.300000
```

Funcția comparator pentru nr. reale

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

Exemplu de funcție cmp pentru sortarea în ordine crescătoare a unui vector de numere reale:

```
int cmp1(const void* a, const void* b)  
{  
    double va, vb;  
    va = *(double *)a;  
    vb = *(double *)b;  
    if(va < vb) return -1;  
    if(va > vb) return 1;  
    return 0;  
}
```

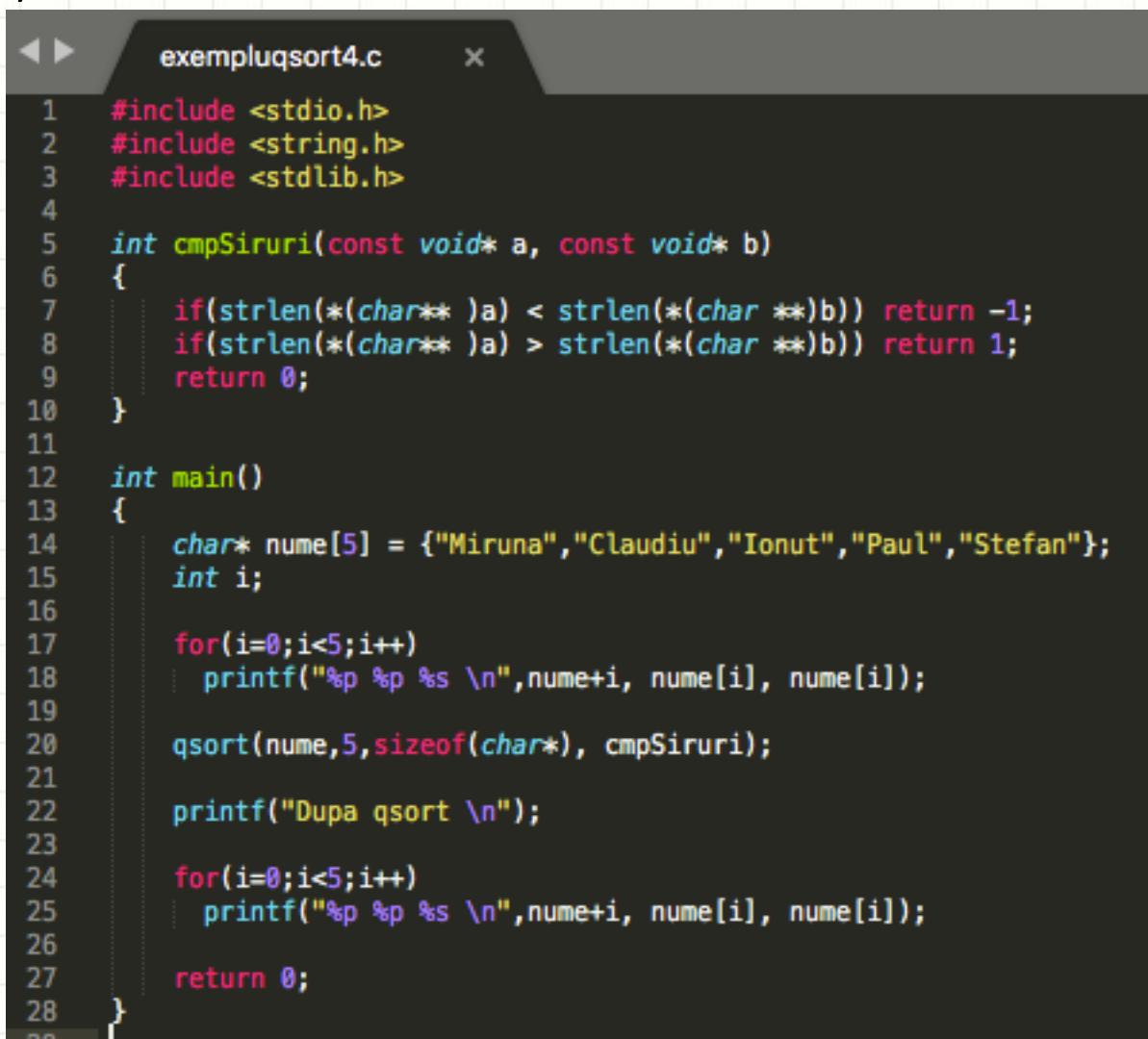
```
int cmp2(const void* a, const void* b)  
{  
    return *(double *)a - *(double *)b;  
}
```

Functia cmp2 poate face underflow sau overflow

Face conversie implicită la int

Funcția comparator pentru lungimea sirurilor

Vrea să sorteze un tablou de siruri de caractere crescător după lungimea sirurilor



```
001 //include <stdio.h>
002 //include <string.h>
003 //include <stdlib.h>
004
005 int cmpSiruri(const void* a, const void* b)
006 {
007     if(strlen(*(char** )a) < strlen(*(char ** )b)) return -1;
008     if(strlen(*(char** )a) > strlen(*(char ** )b)) return 1;
009     return 0;
010 }
011
012 int main()
013 {
014     char* nume[5] = {"Miruna","Claudiu","Ionut","Paul","Stefan"};
015     int i;
016
017     for(i=0;i<5;i++)
018         printf("%p %p %s \n",nume+i, nume[i], nume[i]);
019
020     qsort(nume,5,sizeof(char*), cmpSiruri);
021
022     printf("Dupa qsort \n");
023
024     for(i=0;i<5;i++)
025         printf("%p %p %s \n",nume+i, nume[i], nume[i]);
026
027     return 0;
028 }
```

Funcția comparator pentru lungimea sirurilor

Vrea să sorteze un tablou de siruri de caractere crescător după lungimea sirurilor

```
001 exempluqsort4.c      x
002
003 1 #include <stdio.h>
004 2 #include <string.h>
005 3 #include <stdlib.h>
006
007 4
008 5 int cmpSiruri(const void* a, const void* b)
009 6 {
010 7     if(strlen(*(char** )a) < strlen(*(char ** )b)) return -1;
011 8     if(strlen(*(char** )a) > strlen(*(char ** )b)) return 1;
012 9     return 0;
013 10 }
014
015 11
016 12 int main()
017 13 {
018 14     char* nume[5] = {"Miruna","Claudiu","Ionut","Paul","Stefan"};
019 15     int i;
020 16
021 17     for(i=0;i<5;i++)
022 18         printf("%p %p %s \n",nume+i,
023 19             qsort(nume,5,sizeof(char*), cmpSiruri);
024 20
025 21     printf("Dupa qsort \n");
026 22
027 23     for(i=0;i<5;i++)
028 24         printf("%p %p %s \n",nume+i,
029 25
030 26
031 27
032 28 }
```

```
Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ gcc exempluqsort4.c
Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ ./a.out
0x7fff51838ae0 0x10e3c7f6a Miruna
0x7fff51838ae8 0x10e3c7f71 Claudiu
0x7fff51838af0 0x10e3c7f79 Ionut
0x7fff51838af8 0x10e3c7f7f Paul
0x7fff51838b00 0x10e3c7f84 Stefan
Dupa qsort
0x7fff51838ae0 0x10e3c7f7f Paul
0x7fff51838ae8 0x10e3c7f79 Ionut
0x7fff51838af0 0x10e3c7f6a Miruna
0x7fff51838af8 0x10e3c7f84 Stefan
0x7fff51838b00 0x10e3c7f71 Claudiu
```

Funcția comparator pentru structuri

Am o structură care reține numele, prețul, cantitatea pentru fiecare produs dintr-un magazin. Se dă un vector de asemenea produse pe care vreau să îl sorteze descrescător după preț și în caz de prețuri egale în ordinea alfabetică a numelor produselor.

```
produs.c

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 struct Produs
6 {
7     char nume[25];
8     double pret;
9     double cantitate;
10 };
11
12
13 int cmpProduse(const void *a, const void* b)
14 {
15     if (((struct Produs *)a)->pret == ((struct Produs *)b)->pret)
16         return strcmp(((struct Produs *) a)->nume,((struct Produs *)b)->nume);
17
18     if (((struct Produs *)a)->pret > ((struct Produs *)b)->pret)
19         return -1;
20     return +1;
21 }
22 }
```

Funcția generică de căutare binară bsearch

- **funcția bsearch (binary search) din stdlib.h folosită pentru căutare binară într-un *tablou SORTAT*.** Antetul lui bsearch este:

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

- cheie = pointer cu adresa elementului pe care îl caut în tablou
- adresa = pointer cu adresa primului element al tabloului sortat în care căutăm
- nr_elemente = numarul de elemente al tabloului;
- dimensiune_element = dimensiunea in octeți a fiecărui element al tabloului (char = 1 octet, int = 4 octeți, etc)
- **cmp – funcția de comparare a două elemente**
- **returnează adresa către primul element găsit sau NULL**

Funcția generică de căutare binară bsearch

```
void* bsearch (const void *cheie, const void* adresa, int nr_elemente,  
int dimensiune_element, int (*cmp) (const void *, const void *))  
  
int cmp(const void *a, const void *b)
```



cmp este o funcție generică comparator, compară 2 elemente (primul este cheia pe care o caut). Întoarce:

- un număr < 0 dacă vrem să căutăm cheia în stânga lui b
- un număr >0 dacă vrem să căutăm cheia în dreapta lui b
- 0, dacă s-a găsit cheia

Functia bsearch - exemplu

```
exemplubsearch1.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int cmp1(const void* a, const void* b)
5 {
6     int va, vb;
7     va = *(int *)a;
8     vb = *(int *)b;
9     if(va < vb) return -1;
10    if(va > vb) return 1;
11    return 0;
12 }
13
14 int main()
15 {
16     int i, j, v[6] = {10,1,12,3,14,5};
17     qsort(v,6,sizeof(int),cmp1);
18     printf("Vectorul sortat este \n");
19     for(i=0; i< 6;i++)
20         printf("%d ",v[i]);
21     printf("\n");
22
23     for(j=0;j<=5;j++)
24     {
25         int *p = (int *) bsearch(&j,v,6,sizeof(int),cmp1);
26         if(p)
27             printf("Elementul %d se gaseste in vector pe pozitia %ld \n",j,p-v);
28         else
29             printf("Elementul %d nu se gaseste in vector \n",j);
30     }
31     return 0;
32 }
```

aceeași funcție comparator folosită de qsort și bsearch

Functia bsearch - exemplu

```
exemplubsearch1.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int cmp1(const void* a, const void* b)
5 {
6     int va, vb;
7     va = *(int *)a;
8     vb = *(int *)b;
9     if(va < vb) return -1;
10    if(va > vb) return 1;
11    return 0;
12 }
13
14 int main()
15 {
16     int i, j, v[6] = {10,1,12,3,14,5};
17     qsort(v,6,sizeof(int),cmp1);
18     printf("Vectorul sortat este \n");
19     for(i=0; i< 6;i++)
20         printf("%d ",v[i]);
21     printf("\n");
22
23     for(j=0;j<=5;j++)          Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ gcc exemplubsearch1.c
24     {
25         int *p = (int *) bsearch( Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ ./a.out
26         if(p)                      Vectorul sortat este
27             printf("Elementul %d 1 3 5 10 12 14
28         else                         Elementul 0 nu se gaseste in vector
29             printf("Elementul %d  Elementul 1 se gaseste in vector pe pozitia 0
30         }                           Elementul 2 nu se gaseste in vector
31     return 0;                      Elementul 3 se gaseste in vector pe pozitia 1
32 }                                Elementul 4 nu se gaseste in vector
                                         Elementul 5 se gaseste in vector pe pozitia 2
```

Functia bsearch - exemplu

```
exemplubsearch2.c  x

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 int cmpSiruri(const void* a, const void* b)
6 {
7     if(strlen(*(char** )a) < strlen(*(char **)b)) return -1;
8     if(strlen(*(char** )a) > strlen(*(char **)b)) return 1;
9     return 0;
10 }
11
12 int cmp2(const void* a, const void* b)
13 {
14     int l = *(int*)a;
15     int lSir = strlen(*(char **)b);
16     if (l == lSir)
17         return 0;
18     if (l > lSir)
19         return 1;
20     return -1;
21 }
22
23 int main()
24 {
25     char* nume[5] = {"Miruna","Claudiu","Ionut","Paul","Stefan"};
26     int i;
27
28     qsort(nume,5,sizeof(char*), cmpSiruri);
29
30     int lungimeSir = 5;
31     char** p = bsearch(&lungimeSir,nume,5,sizeof(char *),cmp2);
32     if (p)
33     {
34         printf("Am gasit sirul %s care are lungimea %d \n",*p,lungimeSir);
35     }
36     return 0;
37 }
```

functii comparator diferite
folosite de qsort si bsearch

Functia bsearch - exemplu

```
exemplubsearch2.c x

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 int cmpSiruri(const void* a, const void* b)
6 {
7     if(strlen(*(char**)a) < strlen(*(char **)b)) return -1;
8     if(strlen(*(char**)a) > strlen(*(char **)b)) return 1;
9     return 0;
10 }
11
12 int cmp2(const void* a, const void* b)
13 {
14     int l = *(int*)a;
15     int lSir = strlen(*(char **)b);
16     if (l == lSir)
17         return 0;
18     if (l > lSir)
19         return 1;
20     return -1;
21 }
22
23 int main()
24 {
25     char* nume[5] = {"Miruna", "Claudiu", "Ionut", "Paul", "Stefan"};
26     int i;
27
28     qsort(nume, 5, sizeof(char*), cmpSiruri);
29
30     int lungimeSir = 5;
31     char** p = bsearch(&lungimeSir, nume, 5, sizeof(char *), cmp2);
32     if (p)
33     {
34         printf("Am gasit sirul %s care are lungimea %d\n", *p, lungimeSir);
35     }
36     return 0;
37 }
```

Programarea generică vs. specifică

- ❑ 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.
- ❑ funcție specifică care rezolvă problema pentru numere întregi

```
int lsearch(int x, int v[], int n)
{
    for(int i = 0; i < n; i++)
        if(v[i] == x)
            return i;

    return -1;
}
```

- ❑ ce se modifică și ce rămâne fix la implementarea generică?

Programarea generică vs. specifică

- ❑ ce se modifică și ce rămâne fix la implementarea generică?

```
int lsearch(int x, int v[], int n)
{
    for(int i = 0; i < n; i++)
        if(v[i] == x)
            return i;

    return -1;
}
```

- ❑ antetul trebuie să permită date de tip generic (void *)
- ❑ for-ul rămâne acolo
- ❑ tipul void* nu are aritmetică -> trebuie să precizez dimensiunea în octeți a unui element
- ❑ compararea a două elemente generice din tablou nu o pot face cu == (nu merge pe structuri, siruri de caractere, etc)

Programarea generică vs. specifică

Varianta 1

```
void* lsearch1(void* x,void* t,int n, int d)
{
    char* p = t;
    for(int i=0; i< n; i++)
        if(memcmp(p+i*d,x,d)==0)
            return p+i*d;
    return NULL;
}
```

- ❑ lucrul cu o adresa spre un tip generic (void *) necesită transmiterea unui parametru suplimentar care să exprime lungimea tipului de date a elementelor tabloului unidimensional

Varianta 1

```
exemplul1.c

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 void* lsearch1(void* x,void* t,int n, int d)
6 {
7     char* p = t;
8     for(int i=0; i< n; i++)
9         if(memcmp(p+i*d,x,d)==0)
10             return p+i*d;
11     return NULL;
12 }
13
14 int main()
15 {
16     int v[6] = {1, 2, 3, 4, 5 ,6};
17     int x=2;
18     int* p = lsearch1(&x,v,6,sizeof(int));
19     if(p)
20         printf("Prima aparitie a lui %d in tabloul v este pe pozitia = %ld \n",x, p-v);
21     else
22         printf("%d nu apare in tabloul v\n",x);
23
24     char* nume[5] = {"Miruna","Claudiu","Ionut","Paul","Stefan"};
25     char* w = "Stefan";
26     char** q = lsearch1(w,nume,5,sizeof(char *));
27     if(q)
28         printf("Prima aparitie a lui %s in tabloul nume este pe pozitia = %ld \n",w, q-nume);
29     else
30         printf("%s nu apare in tabloul nume\n",w);
31
32
33     return 0;
34 }
```

Varianta 1

```
exemplul1.c

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 void* lsearch1(void* x,void* t,int n, int d)
6 {
7     char* p = t;
8     for(int i=0; i< n; i++)
9         if(memcmp(p+i*d,x,d)==0)
10             return p+i*d;
11     return NULL;
12 }
13
14 int main()
15 {
16     int v[6] = {1, 2, 3, 4, 5 ,6};
17     int x=2;
18     int* p = lsearch1(&x,v,6,sizeof(int));
19     if(p)
20         printf("Prima aparitie a lui %d in tabloul v este pe pozitia = %ld \n",x, p-v);
21     else
22         printf("%d nu apare in tabloul v\n",x);
23
24     char* nume[5] = {"Miruna","Claudiu","Ionut","Paul","Stefan"};
25     char* w = "Stefan";
26     char** q = lsearch1(w,nume,5,sizeof(char *));
27     if(q)
28         printf("Prima aparitie a lui %s in tabloul nume este pe pozitia = %ld \n",w, q-nume);
29     else
30         printf("%s nu apare in tabloul nume\n",w);
31
32     return 0;
33 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ gcc exemplul1.c
[Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ ./a.out
Prima aparitie a lui 2 in tabloul v este pe pozitia = 1
Stefan nu apare in tabloul nume]
```

Programarea generică vs. specifică

Varianta 1

```
void* lsearch1(void* x,void* t,int n, int d)
{
    char* p = t;
    for(int i=0; i< n; i++)
        if(memcmp(p+i*d,x,d)==0)
            return p+i*d;
    return NULL;
}
```

- funcția memcmp va compara adresele pointerilor și nu sirurile de caractere
- nu e potrivită pentru comparare generică
- folosim o funcție comparator

Programarea generică vs. specifică

Varianta 2

```
void* lsearch2(void* x,void* t,int n, int d, int (*cmp)(const void*,const void*))  
{  
    char* p = t;  
    for(int i=0; i< n; i++)  
        if(cmp(p+i*d,x)==0)  
            return p+i*d;  
    return NULL;  
}
```

- folosim o funcție comparator
- definesc funcția comparator în funcție de ce am nevoie

exemplul1_var2.c

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 void* lsearch2(void* x,void* t,int n, int d, int (*cmp)(const void*,const void*))
6 {
7     char* p = t;
8     for(int i=0; i< n; i++)
9         if(cmp(p+i*d,x)==0)
10             return p+i*d;
11     return NULL;
12 }
13
14 int cmpIntregi(const void *a , const void *b)
15 {
16     int va = *(int *)a;
17     int vb = *(int *)b;
18     if(va < vb) return -1;
19     if(va > vb) return 1;
20     return 0;
21 }
22
23
24 int cmpSiruri(const void* a, const void* b)
25 {
26     return strcmp(*(char **)a, (char *)b);
27 }
28
29 int main()
30 {
31     int v[6] = {1, 2, 3, 4, 5 ,6};
32     int x=2;
33     int* p = lsearch2(&x,v,6,sizeof(int),cmpIntregi);
34     if(p)
35         printf("Prima aparitie a lui %d in tabloul v este pe pozitia = %ld \n",x, p-v);
36     else
37         printf("%d nu apare in tabloul v\n",x);
38
39     char* nume[5] = {"Miruna","Claudiu","Ionut","Paul","Stefan"};
40     char* w = "Stefan";
41     char** q = lsearch2(w,nume,5,sizeof(char *),cmpSiruri);
42     if(q)
43         printf("Prima aparitie a lui %s in tabloul nume este pe pozitia = %ld \n",w, q-nume);
44     else
45         printf("%s nu apare in tabloul nume\n",w);
46
47 }
```

exemplul1_var2.c

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 void* lsearch2(void* x,void* t,int n, int d, int (*cmp)(const void*,const void*))
6 {
7     char* p = t;
8     for(int i=0; i< n; i++)
9         if(cmp(p+i*d,x)==0)
10             return p+i*d;
11     return NULL;
12 }
13
14 int cmpIntregi(const void *a , const void *b)
15 {
16     int va = *(int *)a;
17     int vb = *(int *)b;
18     if(va < vb) return -1;
19     if(va > vb) return 1;
20     return 0;
21 }
22
23
24 int cmpSiruri(const void* a, const void* b)
25 {
26     return strcmp(*(char **)a, (char *)b);
27 }
28
29 int main()
30 {
31     int v[6] = {1, 2, 3, 4, 5 ,6};
32     int x=2;
33     int* p = lsearch2(&x,v,6,sizeof(int),cmpIntregi);
34     if(p)
35         printf("Prima aparitie a lui %d in tabloul v este pe pozitia = %ld \n",x, p-v);
36     else
37         printf("%d nu apare in tabloul v\n",x);
38
39     char* nume[5] = {"Miruna","Claudiu","Ionut","Paul","Stefan"};
40     char* w = "Stefan";
41     char** q = lsearch2(w,nume,5,sizeof(char *),cmpSiruri);
42     if(q)
43         printf("Prima aparitie a lui %s in tabloul nume este pe pozitia = %ld \n",q-v);
44     else
45         printf("%s nu apare in tabloul nume\n");
46     return 0;
47 }
```

Bogdan-Alexes-MacBook-Pro:curs13 bogdan\$ gcc exemplul1_var2.c
Bogdan-Alexes-MacBook-Pro:curs13 bogdan\$./a.out
Prima aparitie a lui 2 in tabloul v este pe pozitia = 1
Prima aparitie a lui Stefan in tabloul nume este pe pozitia = 4

```

25 int cmpSiruri(const void* a, const void* b)
26 {
27     return strcmp(*(char **)a, (char *)b);
28 }
29
30 int cmpSiruri2(const void* a, const void* b)
31 {
32     return strcmp(*(char **)a, *(char **)b);
33 }
34
35
36 int main()
37 {
38     int v[6] = {1, 2, 3, 4, 5 ,6};
39     int x=2;
40     int* p = lsearch2(&x,v,6,sizeof(int),cmpIntregi);
41     if(p)
42         printf("Prima aparitie a lui %d in tabloul v este pe pozitia = %ld \n",x, p-v);
43     else
44         printf("%d nu apare in tabloul v\n",x);
45
46     char* nume[5] = {"Miruna","Claudiu","Ionut","Paul","Stefan"};
47     char* w = "Stefan";
48     char** q = lsearch2(w,nume,5,sizeof(char *),cmpSiruri);
49     if(q)
50         printf("Prima aparitie a lui %s in tabloul nume este pe pozitia = %ld \n",w, q-nume);
51     else
52         printf("%s nu apare in tabloul nume\n",w);
53
54     q = lsearch2(&w,nume,5,sizeof(char *),cmpSiruri2);
55     if(q)
56         printf("Prima aparitie a lui %s in tabloul nume este pe pozitia = %ld \n",w, q-nume);
57     else
58         printf("%s nu apare in tabloul nume\n",w);
59     return 0;

```

```
Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ gcc exemplul1_var2.c
Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ ./a.out
Prima aparitie a lui 2 in tabloul v este pe pozitia = 1
Prima aparitie a lui Stefan in tabloul nume este pe pozitia = 4
Prima aparitie a lui Stefan in tabloul nume este pe pozitia = 4
```

Programarea generică

- ❑ exemplul 2: scrieți o funcție generică care să furnizeze adresa unui element cu valoare maximă dintr-un tablou unidimensional. Criteriul de comparație dintre elementele tabloului va fi implementat printr-o funcție comparator de tipul celei utilizate în funcția qsort din stdlib.h (de exemplu, funcțiile cmpIntregi și cmpSiruri din exemplul anterior).

```
void* maxim(const void *tablou, int nr_elem, int dim_elem, int (*cmp)(const void*, const void *))  
{  
    int i;  
    char *pcrt, *pmax;  
  
    if(nr_elem == 0)  
        return NULL;  
  
    pmax = (char *)tablou;  
    for(i = 1; i < nr_elem; i++)  
    {  
        pcrt = (char *)tablou + i*dim_elem;  
        if(cmp(pcrt, pmax) > 0)  
            pmax = pcrt;  
    }  
  
    return pmax;  
}
```

```
00
01 15 void* maxim(const void *tablou, int nr_elem, int dim_elem,
02 16     int (*cmp)(const void*, const void *))
03 17 {
04 18     int i;
05 19     char *pcrt, *pmax;
06 20
07 21     if(nr_elem == 0)
08 22         return NULL;
09 23
10 24     pmax = (char *)tablou;
11 25     for(i = 1; i < nr_elem; i++)
12 26     {
13 27         pcrt = (char *)tablou + i*dim_elem;
14 28         if(cmp(pcrt, pmax) > 0)
15 29             pmax = pcrt;
16 30     }
17
18     return pmax;
19 }
20
21
22 35 int main()
23 {
24 36     int a[10] = {0, 1, 4, 9, 5 ,6, 9, 9, 9, -5};
25 37     int n=10;
26 38     int *pmax = maxim(a, 10, sizeof(int), cmpIntregi);
27 39     int vmax = *pmax;
28
29     while(pmax != NULL && *pmax == vmax)
30     {
31
32         printf("Valoarea maxima %d apare pe pozitia %ld!\n", vmax, pmax - a);
33         pmax = maxim(pmax+1, n-(pmax-a)-1, sizeof(int), cmpIntregi);
34
35     }
36
37     return 0;
38 }
```

```
00 15 void* maxim(const void *tablou, int nr_elem, int dim_elem,
00 16     int (*cmp)(const void*, const void *))
00 17 {
00 18     int i;
00 19     char *pcrt, *pmax;
00 20
00 21     if(nr_elem == 0)
00 22         return NULL;
00 23
00 24     pmax = (char *)tablou;
00 25     for(i = 1; i < nr_elem; i++)
00 26     {
00 27         pcrt = (char *)tablou + i*dim_elem;
00 28         if(cmp(pcrt, pmax) > 0)
00 29             pmax = pcrt;
00 30     }
00 31
00 32     return pmax;
00 33 }
00 34
00 35 int main()
00 36 {
00 37     int a[10] = {0, 1, 4, 9, 5 ,6, 9, 9, 9, -5};
00 38     int n=10;
00 39     int *pmax = maxim(a, 10, sizeof(int), cmpIntregi);
00 40     int vmax = *pmax;
00 41
00 42     while(pmax != NULL && *pmax == vmax)
00 43     {
00 44         printf("Valoarea maxima %d apare pe pozitia %d!\n", *pmax, Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ gcc exemplul2.c
00 45         pmax = maxim(pmax+1, n-(pmax-a), cmpIntregi); Bogdan-Alexes-MacBook-Pro:curs13 bogdan$ ./a.out
00 46     }
00 47     return 0;
00 48 }
```