10. Pointeri şi tablouri. Transferul de argumente către *main( )*. Pointeri spre funcţii. (Pointers and arrays. Transfer of arguments to *main()*. Pointers to functions.)

1. Obiective:
- Înţelegerea legăturii dintre pointeri şi tablouri.
- Înţelegerea modului în care se transmit parametrii din *linia de comandă*.
- Utilizarea pointerilor spre funcții
- Scrierea şi rularea de programe care exploatează aceste facilităţi.

1'. Objectives:
- Understanding the relation between pointers and arrays.
- Understanding the way the command line arguments are transmitted.
- Pointers to functions usage
- Writing and running programs that have these functionalities.

2. Breviar teoretic

Un nume de tablou (fără index) este un identificator care specifică adresa primului element din tablou. El poate fi tratat ca un pointer constant catre primul element din tablou deoarece comportamentul este în mare parte același.

Consecinţe:
- Adresa tabloului (data de numele lui) poate fi atribuita unui alt pointer (ne-constant), pointer ce poate fi utilizat pentru accesul la elementele tabloului;
- un pointer se poate indexa ca şi când ar fi un tablou, dar dacă nu pointează pe un tablou rezultatul este imprevizibil;
- dacă pointerul *p* pointează pe un tablou compilatorul C/C++ generează cod executabil mai scurt pentru **(p+i)* decât pentru *p[i];*
- numele unui tablou specifică adresa primului element din tablou, care nu se poate modifica, dar poate fi folosită printr-un pointer aritmetic (adunând la el o variabilă cu rol de index. Vezi Exemplul 1).

Tablouri de pointeri

Tablourile de pointeri se definesc similar cu tablourile altor tipuri predefinite şi se folosesc mai ales la crearea de tabele de pointeri către şiruri de caractere care pot fi selectate funcţie de anumite valori întregi fără semn, cu rol de indice:

```
const char *p[] = { "Out of range", "Disk full", "Paper out" };
```

Tablourile de pointeri pot fi accesate şi cu pointeri dubli.

Pointerii si modificatorul const

Există *pointeri către constante* şi *pointeri constanţi*, care nu pot fi modificaţi.

- pointeri către constante

```
const char *str1 = "pointer catre sir de caractere constant";
str1[0] = 'P';          // incorect: sir declarat imutabil
str1 = "ptr la const"; //OK: pointerul nu e declarat cst
```

- pointeri constanţi către constante

```
const char *const str2 = "pointer constant la constanta";
str2 = "ptr const la const";          // incorect
str2[0] = 'C';                        // incorect
```

Indirectarea multiplă

Când un pointer pointează pe alt pointer avem un proces numit *indirectare dublă (multiplă)*. El arată astfel: *Pointer către pointer -----> Pointer ----> Obiect*. Când un pointer pointează alt pointer, primul pointer conţine adresa celui de-al doilea pointer, care pointează locaţia care conţine obiectul. Declararea se face utilizând un asterisc suplimentar în faţa numelui pointerului.

Pointeri spre funcţii

La o funcţie, ca şi la o structură sau un tablou, sistemul de operare îi alocă o adresă de memorie fizică. Un pointer către o funcţie va conţine această adresă.
Declararea unui pointer spre o funcţie trebuie să precizeze tipul returnat de funcţie şi numărul, respectiv tipul parametrilor formali:

*tip_rezultat (*pf) (lista_param_formali);*

Regulă practică: declaraţia se face ca şi cum am scrie un prototip de funcţie, numele funcţiei fiind substituit de construcţia *(*nume_pointer_la_funcţie)*.

Numele unei funcţii este sinonim cu adresa de început a codului său executabil în memorie, astfel că:

pf = nume_functie; este sinonimă cu *pf = &nume_functie;*

Apelul unei funcţii prin intermediul unui pointer se face cu construcţia:

*(*pf) (lista_param_actuali);*

iar dacă funcţia returnează o valoare care se poate atribui unei variabile (funcţie cu tip), atunci apelul poate fi:

*var = (*pf) (lista_param_actuali);*

Standardul C++ permite simplificarea apelului folosind pointeri la functii astfel:

var = pf (lista_param_actuali);

In biblioteca C/C++ (*stdlib.h*) există funcția *qsort( )* care este folosită pentru a sorta un tablou:

*void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*));*

unde:

base - este indicatorul către primul element al tabloului care urmează să fie sortat
nitems - este numărul de elemente din tablou
size - este dimensiunea în octeți a fiecărui element din tablou
compar - este funcția care compară două elemente (se folosester ca si pointer spre ea)

Transferul de argumente către funcţia *main( )*

La lansarea în execuţie multe aplicaţii permit specificarea unor argumente în linia de comandă.
Programele scrise în limbajul C/C++ permit introducerea de argumente în linia de comandă, prin definirea unor parametri pentru funcţia *main()* :

*int main([int argc, char *argv[] [,char *env[]]]) { ... }*

- *argc*, „argument count”, conţine numărul argumentelor (>= 1)
- *argv*, „argument vector” este un tablou de pointeri către şiruri de caractere, unde:
  argv[0] pointează pe numele şi calea programului care se lansează în execuţie
  argv[1] pointează pe primul argument din linia de comandă, ş.a.m.d.
- *env*, „environment variables” este un tablou de pointeri către şiruri de caractere care reprezintă o listă de parametri ai SO (tipul prompt-erului...); ultimul pointer are valoarea NULL, marcând sfârşitul listei de parametri.

Caracteristici:
- Între argumente se consideră ca separator caracterul spaţiu şi tab, iar dacă un argument va conţine spaţiu (tab), acel argument trebuie încadrat între ghilimele.
- Numele *argc, argv, env* nu sunt impuse, utilizatorul poate folosi şi alte nume.
- Pentru că aceste argumente se primesc sub forma de şiruri de caractere, aceste şiruri pot fi convertite spre un format intern de reprezentare în memorie cu funcţiile *atoi(), atof(), atol()*.
- Indiferent de modul de utilizare a funcţiei *main()* de către utilizator, cei 3 parametri sunt alocaţi pe stivă.

Testarea programelor ce transferă argumente din linia de comandă se poate face în două moduri:
- din sistemul de operare, după ce s-a creat fişierul executabil, lansând programul folosind opţiunea *Start->Run* şi apoi selectând fişierul executabil aferent programului, după care se tastează argumentele, separate prin spaţiu; pentru fiecare lansare în execuţie trebuie repetată această operaţie.
- din mediul de programare, stabilind în setările proiectului curent argumentele ce vor ajunge la funcţia *main();* setările respective şi modul în care se ajunge la acestea depinde de mediul de programare cu care se lucrează; odată stabilite, aceste setări sunt valabile pentru toate programele ce se testează şi pentru orice lansare în execuție. (vezi setări Visual Studio)

3. Exemple

Exemplul 1: program pentru determinarea celui mai mic element dintr-un tablou unidimensional.

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

#define DIM 20

int detMin(int *, int);//prototip ce ret min din tablou

int main(void){
int i, dim, min;
int x[DIM];

        printf("\nIntroduceti dimensiunea tabloului unidimensional: ");
        scanf("%d", &dim);
        if(dim<=0 || dim > DIM){
                printf("\n Dimensiune gresita !\n");_
                return 1;
        }
        printf("\n Introduceti elementele tabloului unidimensional:\n");
        for(i=0; i<dim; i++)
        {
```

```
            printf("\tx[%d] = ", i);
            scanf("%d", (x+i));
        }
        min = detMin(x, dim);
printf("\n Cel mai mic element din tabloul unidimensional este: %d\n", min);
return 0;
}


int detMin(int *x, int n){
int i, min;
        min = *x++;
        for(i=1; i<n; i++)
            if(*x < min) min = *x++;
           else x++;
  return min;
}
```

Exemplul 2: program pentru exemplificarea accesului la tablorile de pointeri catre siruri de caractere cu pointeri dublisi tablori de pointeri.

```
#define _CRT_SECURE_NO_WARNINGS
#include <cstdio>
void Err(const char**, int nr_err);
const int dim = 20;

int main( ) {
        const char* p[] = { "Ok\n", "Disk full\n", "Paper out\n" };
        Err(p, 0); Err(p, 1); Err(p, 2);
        char s1[dim], s2[dim], s3[dim];
        printf("\nEnter first string: ");
        scanf("%s", s1);
        printf("\nEnter second string: ");
        scanf("%s", s2);
        printf("\nEnter third string: ");
        scanf("%s", s3);
        const char* pp[] = { s1,s2,s3 };
        Err(pp, 0); printf("\n"); Err(pp, 1); printf("\n"); Err(pp, 2); printf("\n");
}//main

void Err(const char** p, int nr_err) {
        printf(p[nr_err]);
}
```

Exemplul 3: transferul catre o functie a unui tablou de siruri de caractere prin pointer dublu, respectiv prin referinta unui pointer.

```
// double pointer C - pointer reference CPP
#include <iostream>
using namespace std;

const char* someFuncC(const char **p2c);
const char* someFuncCPP(const char *&r);
const int dim = 10;
```

```
int main( ) {
        const char* sir[dim] = { "aa", "bbb", "cc", "dddd", "ee", "ff", "ggg", "hh", "iii", "jj"
};
        const char* res;
        cout << "\nCall C double pointer function" << endl;
        res = someFuncC(sir);
        cout << res << " sir[0] " << sir[0] << endl;
        cout << "\nCall CPP pointer reference function" << endl;
        res = someFuncCPP(*sir);
        cout << res << " sir[0] " << sir[0] << endl;
        return 0;
}

const char* someFuncC(const char **p2c){
        const char* q = "abc";
        *p2c = q;//Change the value of the pointer p2c itself by q which is an address
        return *p2c;
}
const char* someFuncCPP(const char *&r){
        const char* q = "cba";
        r = q;// Change the value of the reference r itself
        return r;
}
```

Exemplul 4: program ce utilizează pointeri spre funcţii.

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

int get_result(int, int, int (*)(int, int));

int max(int, int);
int min(int, int);

int main(void){
int result;

        result = get_result(1, 2, max);
        printf("Max dintre 1 si 2 este: %d\n", result);

        result = get_result(1, 2, min);
        printf("Min dintre 1 si 2 este: %d\n", result);
        return 0;
}

int get_result(int a, int b, int (*compare)(int, int)){
        return(compare(a, b));  // apel functie prin pointer
}


int max(int a, int b){
                printf("Apel functia max:\n");
                return((a > b) ? a: b);
}
```

```
int min(int a, int b)
{
		printf("Apel functia min:\n");
		return((a < b) ? a: b);
}
```

Exemplul 5: program care preia de la consolă mai multe numere întregi reprezentand valori de rezistenţe şi calculează rezistenţa echivalentă grupării serie sau paralel.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <ctype.h>
//#include <conio.h>

#define DIM 10

float calcul(int*, int, float (*)(int*, int));
float serie(int *, int);
float paralel(int *, int);

int main(void)
{
int i, n, tab[DIM];
float rez;
char grp;
float (*pf)(int*, int);

	printf("\nIntroduceti  numarul de rezistente <=10 : ");
	scanf("%d", &n);
	if(n==0) return 1;

	printf("\nIntroduceti %d valori de rezistente :\n", n);
	for(i=0; i<n; i++)
	{
		printf("Rezistenta %d :", i+1);
		scanf("%d", &tab[i]);   //(tab+i)
	}

	printf("\nGrupare serie/paralel (s/p) ? ");
	//grp=_getche();
	fflush(stdin); scanf(" %c",&grp);
	switch (grp) //toupper(grp) -  ctype.h
	{
	case 'S':
	case 's':		pf = serie; break;
	case 'P':
	case 'p':		pf = paralel; break;
	default:		printf("\n Operatie invalida !");
			return 1;
	}//end switch
	rez=calcul(tab,n,pf);
	printf("\n Rezistenta echivalenta este: %f\n", rez);
	return 0;
}//end main
```

```
float calcul(int *tab, int n, float (*pf)(int *, int)){
//  alte prelucrari
        return(pf(tab, n));
}//end calcul

float serie(int *tab, int n){
float suma = 0.;
        while(n)
                suma += tab[--n];
        return(suma);
}//end serie

float paralel(int *tab, int n){
float suma = 0.0f;
        while(n)
                suma += 1.0f/tab[--n];
        return(1.0f/suma);
}//end paralel
```

Exemplul 6: program care permite sortarea unui tablou unidimensional de numere intregi si double folosind functia *qsort( ).*

```
//sorting array of integers/doubles with qsort( )
#include <stdio.h>
#include <stdlib.h>

const int dim = 10;
int comp_int(const void* a, const void* b);
int comp_double(const void* a, const void* b);

int main( ) {
   int i_numbers[dim] = { 1892, 45, 200, -98, 4087, 5, -12345, 1087, 88, -100000 };
   double d_numbers[dim] = { 1892, 45, 200, -98, 4087, 5, -12345, 1087, 88, -100000 };
   int i;
   /* Sort descending the int array */
   qsort(i_numbers, dim, sizeof(int), comp_int);
   printf("\nOrdering descending int values:\n");
   for (i = 0; i < dim; i++) printf("Number = %d\n", i_numbers[i]);
   /* Sort ascending the double array */
   qsort(d_numbers, dim, sizeof(double), comp_double);
   printf("\nOrdering ascending double values:\n");
   for (i = 0; i < dim; i++) printf("Number = %.2lf\n", d_numbers[i]);
   return 0;
}

int comp_int(const void* a, const void* b) {
    return (*(int*)b - *(int*)a); //descending
}
int comp_double(const void* a, const void* b) {
   if (*(double*)a > *(double*)b) return 1; //ascending
   else  if (*(double*)a < *(double*)b)return -1;
    else return 0;
}
```

Exemplul 7: program care preia din linia de comandă numere întregi şi afişează suma acestora.

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

int main (int argc, char *argv[]){
int i, suma=0, n;

    if (argc == 1)  {
        printf("\n\n\tNu ati introdus numerele de adunat !");
        exit(1);
    }//end if

   for(i=1; i<argc; i++)
        {
            n = atoi(argv[i]);
            suma += n;
        }//end for
   printf("\nSuma argumentelor din linia de comanda este : %d\n", suma);
   return 0;
}//end main
```

4. Intrebări:

- Care sunt operatorii specifici pentru pointeri ?
- Care sunt operaţiile permise cu pointerii ?
- Care este diferenţa între un pointer constant şi un pointer către o constantă?
- Ce este un pointer către o funcţie ?
- Cum se poate face apelul unei funcţii folosind pointeri ?
- Ce reprezintă parametri din linia de comandă ?
- Cum pot fi accesaţi parametri din linia de comandă într-un program C/C++?

5. Teme:

1. Rezolvaţi minimum 3 probleme din laboratorul de aplicaţii cu tablouri (laboratorul 8) folosind accesul la elementele tabloului prin pointeri.
2. Se consideră doi parametri întregi şi alţi doi flotanţi din linia de comandă. Să se afişeze suma şi produsul lor.
3. Scrieţi o aplicaţie care citeşte de la tastatură un şir de caractere cu lungimea mai mare decât 7. Folosiţi un pointer pt. a accesa şi afişa caracterele de pe poziţiile 1, 3, 5 şi 7.
4. Citiţi de la tastatură elementele a 2 matrici de valori întregi. Scrieţi o funcţie care primeşte ca parametri pointerii la cele 2 matrici şi returnează un pointer la matricea sumă. Rezultatul însumării matricelor va fi afişat în funcţia main. Afişaţi elementele de pe diagoanala secundară a matricii sumă, folosind acelaşi pointer.
5. Definiţi un tablou de pointeri către şiruri de caractere. Fiecare locaţie a tabloului conţine adrese către unul din urmatoarele şiruri de caractere:
   - "valoare prea mică"
   - "valoare prea mare"
   - "valoare corectă"

   Aplicaţia generează un număr aleator între 1 şi 100 şi apoi citeşte în mod repetat intrarea de la tastatură până când utilizatorul introduce valoarea corectă. Folosiţi

mesajele definite pentru a informa utilizatorul, la fiecare pas, despre relaţia existentă între numărul generat şi ultima valoare citită.

6. Scrieţi un program în care să definiţi un tablou de pointeri spre şiruri de caractere, pe care să-l iniţializaţi cu diferite mesaje. Afişaţi mesajele.
7. Să se scrie un program care preia din linia de comandă şiruri de caractere şi afişează şirul rezultat din concatenarea acestora.
8. Să se scrie un program care inversează şirul de caractere citit din linia de comandă.
9. Scrieţi un program care citeşte de la tastatură elementele de tip float ale unui tablou unidimensional, elemente ce reprezintă mediile unei grupe de studenţi. Să se scrie o funcţie care determină numărul studenţilor cu media >= 8. Afişaţi rezultatul în main. (lucrati cu pointeri, fără variabile globale).
10. Scrieţi un program C/C++ în care se citesc de la tastatură elementele de tip întreg ale unui tablou unidimensional *a*, utilizând o funcţie. Scrieţi o funcţie care completează un alt tablou unidimensional *b*, fiecare element al acestuia fiind obţinut prin scăderea mediei aritmetice a elementelor din *a* din elementul corespunzător din *a*. Scrieţi o funcţie care permite afişarea unui tablou unidimensional şi afişaţi tablourile unidimensionale *a* şi *b*. (utilizând pointeri, fără variabile globale).
11. Scrieţi un program în care se citesc de la tastatură elementele de tip întreg ale unei matrici pătratice, utilizând o funcţie. Scrieţi o funcţie care determină numărul de elemente negative de deasupra diagonalei secundare. Afisaţi rezultatul în main() (fără variabile globale).
12. Scrieţi un program în care se citesc de la tastatură elementele de tip întreg ale unei matrici pătratice, utilizând o funcţie. Scrieţi o funcţie care interschimbă două linii ale matricii. Afişaţi cu o funcţie matricea iniţială şi cea obţinută. Dimensiunea matricii şi numerele ce identifică liniile care vor fi interschimbate se citesc de la tastatură, în funcţia main. (fără variabile globale).
13. Considerând algoritmul care preia numerele de la tastatură direct ordonate crescător într-un tablou unidimensional, folosiţi mecanismul de acces la elemente prin poinetri. Scrieţi o aplicaţie C/C++ care implementează o funcţie care are ca parametri formali un pointer la un tablou unidimensional de tip float şi dimensiunea. (*void dir_sort(float *, int n);*)
14. Scrieţi algoritmul care interclasează două tablouri unidimensionale de tip întreg. Folosiţi pointeri.
15. Ordonaţi crescător un tablou unidimensional de numere intregi citit de la tastatura folosind funcţia de bibliotecă *qsort( )* din *stdlib.h*. Folosiţi acelaşi algoritm pentru numere de tip *float* pe care sa le ordonaţi descrescător.

5'. Homework

1. Resolve minimum 3 problems referring to arrays (from the previous lab number 8), using pointers.
2. Considering two integer and two float parameters from the command line, display the sum and the product of these parameters.
3. Write a C/C++ application that reads from the keyboard an array of characters that has more than 7 elements. Use a pointer for displaying the characters that have the indexes 1, 3, 5 and 7.
4. Write a C/C++ application that reads from the keyboard the elements of 2 integer matrices. Write a function that receives as parameters the pointers to the matrices and returns the pointer to the sum matrix. The result is to be displayed in function main. Display the elements from the second diagonal of the sum matrix using the returned pointer.
5. Define an array of character pointers. Each location will store one of the following messages:

- "value too small"
- "value too big"
- "correct value"

The application generates a random number between 0 and 100 and then reads repeatedly the keyboard until the user enters the right number. Use the previously defines messages for informing the client about the relation between the auto-generated number and the last value entered from the keyboard.

6. Write a C/C++ application that defines an array of pointers to character strings and initialize them with different messages. Display the messages.
7. Write a C/C++ program that reads some character arrays from the command line and displays the resulting concatenated string.
8. Write a C/C++ program that inverts a string of characters read from the command line.
9. Write a C/C++ program that reads from the keyboard the float type elements of a one-dimensional array. The values represent the average marks of a group of students. Write a function that determines the number of students having the average mark >= 8. Display the result in the main function. (use pointers, avoid global variables).
10. Write a C/C++ program that implements a function for reading from the keyboard some integer values. The function stores the values in a one-dimensional array named A. Write another function that fills a different one-dimensional array B, each element being obtained by subtracting the mean value of the initial values from the corresponding element located in the one-dimensional array A. Write a function that displays a one-dimensional array and use it for A and B one-dimensional array. (use pointers, don't use global variables)
11. Write a C/C++ program that defines a function for reading from the keyboard the integer type values that form a [n x n] matrix. Write a function that determines the number of negative elements that are located above the secondary diagonal. Display the result in the main function (don't use global variables).
12. Write a C/C++ program that defines a function for reading from the keyboard the integer type values that form a [n x n] matrix. Write a function that interchanges two lines of the matrix. Use another function for displaying the initial and the processed matrices. Read from the keyboard (in the main function) the dimension (n) of the matrix and the indexes that indicate the rows to be switched (do not use global variables).
13. Considering the algorithm that directly introduces the numbers from KB in a sorted mode in a one-dimensional array, use the mechanism to access by pointers the elements. Develop a C/C++ application considering a function having as formal parameters a pointer to a one-dimensional array of float type and the dimension. (*void dir_sort(float *, int n);*)
14. Develop the algorithm able to interclass two one-dimensional arrays of int type. Use pointers.
15. Order in increasing mode a one-dimensional array of integer type introduced from the keyboard using *qsort( )* from *stdlib.h*. Use the same algorithm for *float* numbers and a decreasing order.