10. Pointeri și tablouri. Transferul de argumente către funcția main(). Pointeri spre funcții

Pointers and arrays. Passing of arguments to main() function. Pointers and functions.

1.1. Obiective

- Înțelegerea legăturii dintre pointeri și tablouri.
- Înțelegerea modului în care se transmit parametrii din linia de comandă.
- Utilizarea pointerilor spre funcții
- Scrierea și rularea de programe care exploatează aceste facilități.

1.2. Objectives

- Understanding the relation between pointers and arrays.
- Understanding the way, the command line arguments are transmitted.
- Usage of pointers to functions.
- Writing and running programs that make use of these functionalities.

1.3. Breviar teoretic

Un nume de variabilă tablou, atunci când este folosit fără operatorul de indexare, este un identificator care specifică adresa primului element din tablou. El poate fi tratat ca un pointer constant către primul element din tablou, deoarece comportamentul este în mare parte același.

Consecințe:

- Adresa tabloului, dată de numele lui, poate fi atribuită unui alt pointer (ne-constant, de același tip), pointer ce poate fi utilizat pentru accesul la elementele tabloului;
- Un pointer se poate indexa ca și când ar fi un tablou, dar dacă nu conține adresa unui tablou rezultatul este imprevizibil;
- Dacă `p` conține adresa unui tablou, compilatorul C/C++ generează cod executabil mai scurt pentru `*(p+i)` decât pentru `p[i]` (compilatoarele moderne – cu mai multe treceri, sunt capabile a optimiza codul automat, iar această regulă nu mai e neapărat valabilă);
- Numele unui tablou specifică adresa primului element din tablou, adresă care nu se poate modifica. Acesta poate fi folosit ca un pointer aritmetic (adunând la el o variabilă cu rol de index. Vezi Exemplul 1).

1.3.1. Tablouri de pointeri

Tablourile de pointeri se definesc similar cu tablourile altor tipuri predefinite și se folosesc mai ales la crearea de tabele de pointeri către șiruri de caractere care pot fi selectate în funcție 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 și cu pointeri dubli la prelucrare.

1.3.2. Pointerii și modificatorul const

Există pointeri către constante și pointeri constanți, care nu pot fi modificați.

- pointeri către constante:

```
const char *str1 = "pointer catre sir de caractere constant";
//str1[0] = 'P';   // incorect: sirul fiind declarat imutabil (const)
```

```
str1 = "ptr la const"; //OK: pointerul nu e declarat const, poate fi
folosit pentru a indica spre un alt sir
```

- pointeri constanți către șiruri de caractere (nu sunt acceptate de noile compilatoare C++). O valoare de tip `const char *` nu poate fi folosită pentru a inițializa o entitate de tip `char * const`, acceptată de compilatoarele C:

```
char *const str2 = "pointer constant";
//str2 = "ptr to const";  // incorect, str2 fiind constant
str2[0] = 'P';    // ok, sirul nu e constant, dar functioneaza doar in C
```

- pointeri constanți către constante:

```
const char *const str3 = "pointer constant la constanta";
//str3 = "ptr const la const";      // incorect
//str3[0] = 'P';                    // incorect
```

1.3.3. Indirectarea multiplă

Când un pointer conține adresa unui alt pointer avem un proces numit indirectare dublă (multiplă). El arată astfel: Pointer către pointer -----> Pointer ----> Obiect. Primul pointer conține adresa celui de-al doilea pointer, care conține adresa de memorie unde se află stocat obiectul. Declararea se face utilizând un asterisc suplimentar în fața numelui pointerului.

1.3.4. Pointeri spre funcții

Unei funcții, ca și unei structuri structuri sau unui tablou, sistemul de operare îi alocă o adresă de memorie fizică. Un pointer către o funcție va conține această adresă.

Declararea unui pointer spre o funcție trebuie să precizeze tipul returnat de funcție și lista de parametri formali:

```
tip_rezultat (*pf)(lista_param_formali);
```

Regulă practică: declarația se face ca și cum am scrie un prototip de funcție, numele funcției fiind substituit de construcția (`*nume_pointer_la_funcție`).

Numele unei funcții 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 funcții prin intermediul unui pointer se face cu construcția:

```
(*pf)(lista_param_actuali);
```

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

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

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

```
var = pf(lista_param_actuali);
```

În 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 folosește ca pointer spre ea).

1.3.5. Transferul de argumente către funcția main()

La lansarea în execuție, multe aplicații 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 funcția `main()`:

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

- `argc`, „argument count", conține numărul argumentelor (>= 1)
- `argv`, „argument vector" este un tablou de pointeri către șiruri de caractere, unde:
  - `argv[0]` conține adresa șirului de caractere cu numele și calea programului care se lansează în execuție
  - `argv[1]` conține adresa șirului de caractere ce reprezintă primul argument din linia de comandă, ș.a.m.d.
- `env`, „environment variables" este un tablou de pointeri către șiruri de caractere care reprezintă o listă de parametri ai SO (tipul prompterului,...); ultimul pointer are valoarea NULL, marcând sfârșitul listei de parametri.

Caracteristici:

Între argumente se consideră ca separator caracterul spațiu și tab, iar dacă un argument va conține spațiu (tab), acel argument trebuie încadrat între ghilimele.

Numele `argc`, `argv`, `env` nu sunt impuse, utilizatorul poate folosi și alte nume.

Pentru că aceste argumente se primesc sub forma de șiruri de caractere, aceste șiruri pot fi convertite spre un format intern de reprezentare în memorie cu funcțiile `atoi()`, `atof()`, `atol()`, `atoll()`.

Indiferent de modul de utilizare a funcției `main()` de către utilizator, cei 3 parametri sunt alocați pe stivă.

Testarea programelor ce transferă argumente din linia de comandă se poate face în două moduri:

1. din sistemul de operare, după ce s-a creat fișierul executabil, lansând programul folosind opțiunea `Start->Run` și apoi selectând fișierul executabil aferent programului, după care se tastează argumentele, separate prin spațiu; pentru fiecare lansare în execuție trebuie repetată această operație.
2. din mediul de programare, stabilind în setările proiectului curent argumentele ce vor ajunge la funcția `main( )`; setările respective și modul în care se ajunge la acestea depinde de mediul de programare cu care se lucrează.

1.4. Theoretical brief

The name of an array variable, when used without the indexing operator, is an identifier that specifies the address of the array's first element. It can be treated as a constant pointer to the first element of the array, as the behavior is largely the same.

Consequences:

- The address of the array, given by its name, can be assigned to another (non-constant, same type) pointer, which can be used to access the array's elements;
- A pointer can be indexed as if it were an array, but if it doesn't point to an array, the result is unpredictable;

- If `p` points to an array, the C/C++ compiler generates shorter executable code for `*(p+i)` than for `p[i]` (modern compilers – multi-pass compilers, are capable of optimizing code automatically, so this rule is not necessarily applicable);
- The name of an array variable specifies the address of the first element in the array, an address that cannot be modified, and it can be used as an arithmetic pointer (adding to it a variable acting as an index). See Example 1.

1.4.1. Array of pointers

Arrays of pointers are defined similarly to arrays of other predefined types and are mainly used to create tables of pointers to strings that can be selected based on certain unsigned integer values, serving as indices:

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

Arrays of pointers can also be accessed with double pointers (not pointer to double but it is a

type **) when are processed.

1.4.2. Pointers and the const modifier

In C/C++ there are pointer to constants and constant pointers, which cannot be modified:

- pointers to constants:

```
const char *str1 = "pointer to a constant string";
//str1[0] = 'P';  // incorrect: the string is declared immutable (const)
str1 = "ptr to const";//OK: the pointer is not declared const, it can be used for another string
```

- Constant pointers to a string (not accepted by new C++ compilers - A value of type “const char*” cannot be used to initialize an entity of type “char * const” – Accepted by C compilers)

```
char *const str2 = "constant pointer";
//str2 = "ptr to const";  // incorrect, str2 is constant
str2[0] = 'P';    // ok, the string is not constant, but only works in C
```

- constant pointers to constants:

```
const char *const str3 = "constant pointer to a constant";
//str3 = "const ptr to const";      // incorrect
//str3[0] = 'P';                    // incorrect
```

1.4.3. Multiple indirection

When one pointer points to another pointer, we have a process called double (multiple) indirection. It looks like this: Pointer to pointer -----> Pointer ----> Object. When a pointer points to another pointer, the first pointer contains the address of the second pointer, which points to the location containing the object. The declaration is made using an additional asterisk in front of the pointer name.

1.4.4. Pointers to functions

Similar to a structure or an array, an operating system allocates a physical memory address to a function. A pointer to a function will contain this address.

Declaring a pointer to a function must specify the type returned by the function and the list of formal parameters:

```
return_type (*pf)(list_of_formal_parameters);
```

Practical Rule: The declaration is made as if writing a function prototype, with the function name being replaced by the construct `(*function_pointer_name)`.

The name of a function is synonymous with the starting address of its executable code in memory, so:

`pf = function_name;` is synonymous with `pf = &function_name;`

Calling a function through a pointer is done with the construct:

```
(*pf)(list_of_actual_parameters);
```

and if the function returns a value that can be assigned to a variable (typed function), then the call can be:

```
var = (*pf)(list_of_actual_parameters);
```

C++ standard allows **simplifying calls** using function pointers as follows:

```
var = pf(list_of_actual_parameters);
```

In the C/C++ library (`stdlib.h`), there is the `qsort()` function, which is used to sort an array:

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

where:

base is the pointer to the first element of the array to be sorted,

nitems is the number of elements in the array,

size is the size in bytes of each element in the array,

compar is the function that compares two elements (used as a pointer to it).

1.4.5. Parameter passing to main function

Upon execution, many applications allow specifying arguments through the command line.

Programs written in C/C++ allow the introduction of command line arguments by defining parameters for the main() function:

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

- `argc`, or "argument count", contains the number of arguments (≥ 1).
- `argv`, or "argument vector", is an array of pointers to character strings, where:
  - `argv[0]` points to the name and path of the program being executed,
  - `argv[1]` points to the first argument from the command line, and so on.
- `env`, or "environment variables", is an array of pointers to character strings representing a list of OS parameters (like the type of the prompter...). The last pointer is NULL, marking the end of the parameter list.

Characteristics:

Space and tab characters are considered separators between arguments. If an argument contains a space (tab), it should be enclosed in quotes.

The names `argc`, `argv`, and `env` are not mandatory; users can utilize other names.

Since these arguments are received in the form of character strings, they can be converted to an internal memory representation format with functions such as `atoi()`, `atof()`, and `atol()`, `atoll()`.

Regardless of how the user utilizes the `main()` function, the three parameters are allocated on the stack.

Testing of programs that transfer arguments from the command line can be done in two ways:

1. From the operating system: After the executable file has been created, launch the program using the `Start -> Run` option and then select the corresponding executable file. Afterward, type the arguments separated by space. This operation must be repeated for each execution.
2. From the development environment: Establish in the current project settings the arguments that will reach the `main()` function. These particular settings and how to access them depend on the development environment being used. Once set, these settings are valid for all programs being tested and for any execution. (See Visual Studio settings for instance).

1.5. Exemple / Examples

1.5.1. Ex. 1

```
/* Program for determining the minimum value in a one-dimensional array
*/

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

#define DIM 20

// Prototype of function to determine the minimum value from an array
int detMin(int *, int);//pointer to a one-dimensional array and dimension of array

int main( ){
    int i, dim, min;
    int x[DIM]={};//int x[DIM]{};
    do
    {
    printf("\n Type the size of the one-dimensional array <=%d: ", DIM);
    scanf("%d", &dim);
    if (dim <= 0 || dim > DIM)printf("\nIncorrect size (must be >0 and <=%d !", DIM);
    } while (dim <= 0 || dim > DIM);

    printf("\n Type the elements of the one-dimensional array:\n");
    for(i=0; i<dim; i++)
    {
        printf("\tx[%d] = ", i);
        scanf("%d", (x+i));
    }
    min = detMin(x, dim);

    printf("\n The minimum value of the one-dimensional array is: %d\n", min);
    return 0;
}//main

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;
}//detMin
```

1.5.2. Ex. 2

```
/*  Program for showcasing the access with array of pointers to character arrays
(strings) using double pointers.
*/

#define _CRT_SECURE_NO_WARNINGS
#include <cstdio>
void err(const char **, int nr_err);
constexpr 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]);
}//err
```

1.5.3. Ex. 3

```
/*  Passing an array of pointers to character arrays to a function by double
pointer and by reference to a pointer.
*/
// double pointer C - pointer reference CPP
#include <iostream>
using namespace std;

const char *someFuncC(const char **p2c); //double pointer parameter
const char *someFuncCPP(const char *&r); //reference to pointer parameter
constexpr 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;
}//main

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;
}// someFuncC

const char *someFuncCPP(const char *&r){
    const char *q = "cba";
    r = q;// Change the value of the reference r itself
    return r;
}// someFuncCPP
```

1.5.4. Ex. 4

```
/*  Program that makes use of pointers to functions.*/

#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( ){
    int result;

    // Call to get_result, passing the function max as parameter
    result = get_result(1, 2, max);
    printf("The maximum out of 1 and 2 is: %d\n", result);

    // Call to get_result, passing the function min as parameter
    result = get_result(1, 2, min);
    printf("The minimum out of 1 and 2 is: %d\n", result);
    return 0;
}//main

int get_result(int a, int b, int (*compare)(int, int))
{
    // Call to function using pointer
    return(compare(a, b));
}// get_result

int max(int a, int b)
{
    printf("Call to function max:\n");
    return((a > b) ? a: b);
}//max

int min(int a, int b)
{
    printf("Call to function min:\n");
    return((a < b) ? a: b);
}//min
```

1.5.5. Ex. 5

```
/*  Program that reads from the console multiple integers representing values of
resistors and computes the equivalent resistor for series or parallel grouping.
*/

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

#define DIM 10

//functions prototypes
float calcul(int *, int, float (*)(int *, int));
float serie(int *, int);
float paralel(int *, int);

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

    printf("\n Type the number of resistors <=%d: ", DIM);
    scanf("%d", &n);
    if (n <= 0 || n>DIM){
        printf("\n Incorrect value supplied!\n");
        return 1;
    }

    printf("\n Type %d values representing values of resistors:\n", n);
    for (i = 0; i < n; i++)
    {
        printf("Resistor %d :", i);
        scanf("%d", &tab[i]);   //(tab+i)
    }

    printf("\n The type of the desired grouping is (s - series/p - parallel) ? ");
    //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 Invalid value selected, quitting !");
       return 1; //non-zero exit code signals an error to the parent process
    }//end switch

    printf("\n The equivalent %s grouping has the resistance: %.2f\n",
        pf == serie ? "series" : "parallel",
        calcul(tab, n, pf));
    return 0;
}//main
```

```
float calcul(int *tab, int n, float (*pf)(int *, int))
{
    // we may add more common processing steps here (a validation or other)
    return(pf(tab, n));
}//calcul

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

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

1.5.6. Ex. 6

```
/*  Program that sorts an array of int or double using the qsort() function.
*/

#include <stdio.h>
#include <stdlib.h>

constexpr int dim = 10;
//protypes for sorting refinements
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;
}//main
```

```
int comp_int(const void *a, const void *b) {
    return (*(int *)b - *(int *)a); //descending
}// comp_int

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;
}// comp_double
```

1.5.7. Ex. 7

```
/*  Program that reads integers from the command line and shows their sum.
*/

#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\tYou must specify the numbers to add in the command line!");
        exit(1);
    }//end if

    for (i = 1; i < argc; i++)
    {
        n = atoi(argv[i]);
        suma += n;
    }//end for
    printf("\nThe sum of the command line arguments is: %d\n", suma);
    return 0;
}//main
```

1.6. Întrebări

- Care sunt operatorii specifici pentru pointeri?
- Care sunt operațiile permise cu pointerii?
- Care este diferența între un pointer constant și un pointer către o constantă?
- Ce este un pointer către o funcție?
- Cum se poate face apelul unei funcții folosind pointeri?
- Ce reprezintă parametrii din linia de comandă?
- Cum pot fi accesați parametrii din linia de comandă într-un program C/C++?

1.7. Questions

- What are the operators that can be used with pointers?
- What operations are allowed on pointers?
- What is the difference between a constant pointer and a pointer to a constant?
- What is a pointer to a function?
- How can a function be called using pointers?

- What do the parameters in the command line represent?
- How can the parameters from the command line be accessed in a C/C++ program?

1.8. Lucru individual

1. Rezolvați minimum 3 probleme din laboratorul anterior de aplicații cu tablouri, folosind accesul la elementele tabloului prin pointeri.
2. Se consideră doi parametri întregi și alți doi flotanți preluați din linia de comandă. Să se afișeze suma primelor 2 valori întregi și produsul ultimelor 2 valori de tip `float` într-o aplicație C/C++.
3. Scrieți o aplicație care citește de la tastatură un șir de caractere cu lungimea mai mare decât 7. Folosiți un pointer pentru a accesa și afișa caracterele de pe pozițiile 1, 3, 5 și 7.
4. Folosind limbajul C/C++ citiți de la tastatură elementele a 2 matrice de valori întregi (pătrate). Scrieți o funcție care primește ca parametri pointerii la cele 2 matrice și returnează un pointer la matricea sumă (alocată dinamic - sau gestionată static cu un pointer). Rezultatul însumării matricelor va fi afișat în funcția `main()`. Afișați elementele de pe diagonala secundară a matricei sumă, folosind același pointer. Considerați și cazul generalizat în care matricile nu sunt pătrate și ajustați dimensiunile pentru a face operațiile cerute.
5. Definiți un tablou de pointeri către șiruri de caractere. Fiecare locație a tabloului conține adrese către unul din următoarele șiruri de caractere:
   - "valoare prea mică"
   - "valoare prea mare"
   - "valoare corectă"

   Aplicația C/C++ generează un număr aleator între 1 și 100 și apoi citește în mod repetat intrarea de la tastatură până când utilizatorul introduce valoarea corectă. Folosiți mesajele definite pentru a informa utilizatorul, la fiecare pas, despre relația existentă între numărul generat și ultima valoare citită.
6. Scrieți un program C/C++ în care să definiți un tablou de pointeri spre șiruri de caractere, pe care să-l inițializați cu diferite mesaje. Afișați mesajele.
7. Să se scrie un program C/C++ care preia din linia de comandă șiruri de caractere și afișează șirul rezultat din concatenarea acestora.
8. Să se scrie un program C/C++ care inversează șirul de caractere citit din linia de comandă.
9. Scrieți un program C/C++ care citește de la tastatură elementele de tip `float` ale unui tablou unidimensional, elemente ce reprezintă mediile unei grupe de studenți. Să se scrie o funcție care determină numărul studenților cu media >= 8. Afișați rezultatul în `main()`. (lucrați cu pointeri, fără variabile globale).
10. Scrieți un program C/C++ în care se citesc de la tastatură elementele de tip întreg ale unui tablou unidimensional `arr_A`, dimensiunea reală fiind preluată din linia de comandă, utilizând o funcție. Scrieți o funcție care completează un alt tablou unidimensional `arr_B` de tip real, fiecare element al acestuia fiind obținut prin scăderea mediei aritmetice a elementelor din `arr_A` din elementul corespunzător din `arr_A`. Scrieți două funcții care permit afișarea unui tablou unidimensional intreg respectiv real și afișați tablourile unidimensionale `arr_A` și `arr_B`. (utilizând pointeri, fără variabile globale).
11. Scrieți un program C/C++ în care se citesc de la tastatură elementele de tip întreg ale unei matrice pătratice, utilizând o funcție. Scrieți o funcție care determină numărul de elemente negative de deasupra diagonalei secundare. Afișați rezultatul în `main()` (fără variabile globale).
12. Scrieți un program C/C++ în care se citesc de la tastatură elementele de tip întreg ale unei matrice pătratice, utilizând o funcție. Scrieți o funcție care interschimbă două linii ale matricei. Afișați cu o funcție matricea inițială și cea obținută. Dimensiunea matricei și

numerele ce identifică liniile care vor fi interschimbate se citesc de la tastatură, în funcția `main()`. (fără variabile globale).

13. Considerând algoritmul care preia numerele de la tastatură direct ordonate crescător într-un tablou unidimensional, folosiți mecanismul de acces la elemente prin pointeri. Scrieți o aplicație C/C++ care implementează o funcție care are ca parametri formali un pointer la un tablou unidimensional de tip `float` și dimensiunea. (`void dir_sort(float *, int n);`)
14. Scrieți în C/C++ algoritmul care interclasează două tablouri unidimensionale sortate, de tip întreg. Folosiți pointeri.
15. Ordonați crescător în C/C++ un tablou unidimensional de numere întregi citit de la tastatură (sau generat aleator) folosind funcția de bibliotecă `qsort()` din `stdlib.h`. Folosiți același algoritm `qsort()` pentru valori de tip `float` pe care să le ordonați descrescător.

1.9. Individual work

1. Resolve minimum 3 problems referring to arrays from the anterior array laboratory, using pointer access mechanism.
2. Considering two integers and two `float` parameters from the command line, display the sum of first two integers and the product of the last two `float` parameters using a C/C++ application.
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 square integer matrices. Write a function that receives the pointers to the matrices as parameters and returns the pointer to the sum matrix (dynamically allocated or managed as a static array with a pointer). The result is to be displayed in function `main()`. Display the elements from the second diagonal of the sum matrix using the returned pointer. Also consider the generalized case where the matrices are not square and adjust the dimensions to do the required operations.
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 C/C++ 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 defined 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 real dimension introduced from command line. The function stores the values in a one-dimensional array named `arr_A`. Write another function that fills a different one-dimensional array `arr_B of real type`, each element being obtained by subtracting the mean value of the initial values from the corresponding element located in the one-

dimensional array `arr_A`. Write functions that displays a one-dimensional array of integer and real type and use it for `arr_A` and `arr_B` one-dimensional arrays. (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 in C/C++ the algorithm able to interclass two one-dimensional sorted arrays of int type. Use pointers.
15. Order using C/C++ in increasing mode a one-dimensional array of integer type introduced from the keyboard (or generated in random mode) using `qsort()` from `stdlib.h`. Use the same `qsort()` algorithm for `float` numbers and a decreasing order.