Prerequisites:

- Basic knowledge of using computer and text editor
- Knowledge how to create a program
- Knowledge how to create a function
- Knowledge of different types
- Knowledge about conditional statements
- Knowledge about loops
- Knowledge how to work with files

Aims:

In this laboratory student will learn how to:

- create pointer(s)
- read and write particular data into memory

1 Theoretical background

1.1 Definition

To define a new variable which is a pointer you should write for example:

```
unsigned short *my_variable;
```

In this case this variable has size of 4B for 32-bit program (or 8B for 64-bit program). It is advised to set memory address of a newly created pointer, because without this it points some randomly address from memory and sooner or later is will be source of problem(s) in your program. If you do not know what address should be given in such a moment (because for example it will be known later in your program), you can set value 0 for it:

```
unsigned short *my_variable = 0;
```

However, it is even better to use a special symbol NULL defined specially in C language:

```
unsigned short *my_variable = NULL;
```

It makes program more clearly.

1.2 Variable's Address

To use address of some variable you can use already known operator &. In the following source code there is some value stored in variable `a`:

```
unsigned short a = 12345;
unsigned short *Pointer = &a;
```

Pointer `Pointer` allows to point a memory address of variable `a`. It allows also to write and modify the content of this memory cell.

1.3 Amorphous Pointer

However, as you can see, you do not know exact address of this variable. To read an exact address you have to use the amorphous pointer:

```
int a = 5;
int *Pointer = &a;
void *Amorphous = Pointer;
```

It could be written also in a simplest way:

```
int a = 5;
void Amorphous = &a;
```

however, for beginners it could be not so clearly as the previous example.

1.4 Dereference

If you want to read a content of some memory cell you should use a pointer which consists such a address, for example:

```
unsigned short a = 12345;
unsigned short *Pointer = &a;
printf("%d", *Pointer);
```

It is called a **dereference**. In turn, to write a new data into this memory cell where is stored content of variable `a` you can do:

```
*Pointer = 54321;
```

for putting a new values straight from your program or by using:

```
scanf("%hu", Pointer);
```

to read a new value from the keyboard. It should be noted that `scanf` is using a memory address as an argument therefore there is no need to use operator & as it was done before for typical variables.

2 Exercises

1. Create a new pointer `my_pointer` and display in the terminal its default memory's address. Display content of this memory cell as well.
2. Create a new integer array (50 elements). Fill it using random values. Sort all values from the smallest to the biggest one using to this only one pointer (not normal variables).
3. Repeat previous task, however, this time sort all values from the biggest to the smallest one.
4. Create four new pointers and use them to store four values (one per pointer). Write memory address of each pointer and its content is some file (create it and display it content in a text editor).
5. Create a new program, define five new pointers (set them to NULL), and read content of the text file created in a previous task in a way where addresses written in the text file will be granted to four your pointers. Write also values taken from this text file to memory cells as well. Use right now the fifth pointer to point the first memory address, modify its content and display all variables (pointers' contents) in the terminal. Then modify next variable's value using also only the fifth pointer and display all variables (pointers' contents) in the terminal, and so on till all four values of used variables will be changed. At the end write a new values into this same file which was read at beginning of this task – do not overwrite this file, just add a new values to the previous one!