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
- Knowledge how to create a pointer

Aims:

In this laboratory student will learn how to:

- Create list
- Modify list and add/remove list's elements

1 Theoretical background

1.1 List

In array it is possible to collect set of data. However, when you want to add a new item you have to create a new array of greater size (number of elements), copy all data from old array and add a new element. Then old array should be removed to save memory space. Adding later another new element should be done in this same way as well. However, such a solution is not recommended especially when a lot of elements are stored in array.

Second problem is when you have sorted array and you want to add a new elements in the middle of such an array. Then you have to move (push) other elements by one position. If such an array has a lot of elements then such a simple task became complicated and time consuming.

To solve these issues you can use a list. List is a collection of data. And each elements has its value as wall as two pointers. One pointer points next list's elements, second pointer points previous list's element. Lets say you have defined already such a structure:

```
struct List
{
    float value;
    unsigned short *next;
    unsigned short *prev;
};
```

In the main function you can have the following example source code:

```
struct List MyElement1, MyElement2, MyElement3;
MyElement1.value = 10;
MyElement1.next = &MyElement2;
MyElement1.prev = NULL;

MyElement1.value = 11;
MyElement1.next = &MyElement3;
MyElement1.prev = &MyElement1;

MyElement1.value = 12;
MyElement1.next = NULL;
MyElement1.prev = &MyElement2;
```

There are three elements: MyElement1, MyElement2, and MyElement3. Each element points next element (the *next* pointer), previous element (the *prev* pointer), and has its own value *value*. In this example the first element has no previous element (because it is the first element in the list) and the last element has no next element (because it is the last element in the list).

To be sure that it works correctly the following source code could be added:

```
printf("%f\t%p\t%p\n", MyElement1.value, MyElement1.next,
       MyElement1.prev");
printf("%f\t%p\t%p\n", MyElement2.value, MyElement2.next,
       MyElement2.prev");
printf("%f\t%p\t%p\n", MyElement3.value, MyElement3.next,
       MyElement3.prev");
```

Relevant values displayed in the console should be the same. If there are not, fix errors.

2 Exercises

1. Create a new list of 15 elements (random values). Display these elements.
2. Modify program from the previous task in a way where all elements are sorted from the smallest one to the biggest one. Display such a list in the terminal.
3. Repeat task 2., however, this time elements are sorted from the biggest one to the smallest one.
4. Create 4-elements 1D array where each element of this array is a list of random number of elements. Ask user to provide each value from keyboard.
5. Sort each list from task 4. from the smallest one to the biggest one. Display result in the terminal.
6. Sort right now all list together in a way where:
   - the first list consists only even elements below 0,
   - the second list consists only odd elements below 0,
   - the third list consists only even elements above 0,
   - the fourth list consists only odd elements above 0.