Prerequisites:

- Basic knowledge of using computer and text editor
- Knowledge how to create a program
- Knowledge how to create functions
- Knowledge of different types
- Knowledge about conditional statements

Aims:

In this laboratory student will learn how to:

- use `while` loop
- use `do ... while` loop
- use `for` loop
- use arrays

1 Theoretical background

1.1 The while loop

It is the simplest loop. Such a loop is divided into two parts – checking of the logical statement(s) and block of instructions. If logical statement is `true` then set of instructions included in loop's block section are running, if logical statement is `false` then loop is finished and program skips do the first line just after this loop. The general structure of the `while` loop is as follow:

```
while (logical_expression) something_to_do;
```

It is also possible to put more instructions inside this loop using `{` and `}` braces. Then `while` looks as follows:

```
while (logical_expression)
{
    instruction1;
    instruction2;
    instruction3;
    ...
    instructionN;
}
```

1.2 The do ... while loop

This loop differs from the `while` loop only in order of logical statement(s) checking and running the set of instructions. It means that this loop will iterate at least once. So, at first set of instructions is done and then is checked logical statement. The `do ... while` loop has structure as follows:

```
do
{
    instruction1;
    instruction2;
    instruction3;
    ...
    instructionN;
} while (logical_expression);
```

1.3 The for loop

This is the most complicated loop in the C language. However, it is possible to write previous loops using the `for` loop. In general, the structure of the `for` loop is as follows:

```
for (start_statement; loop_condition; end_statement)
{
    instruction1;
    instruction2;
    instruction3;
    ...
    instructionN;
}
```

If you are using some counter in your loop then obviously the `for` loop should be chosen.

1.4 Breaking loop and skipping to the next iteration

If there is need to escape from some loop before all loop's iterations will be finished a special word `break` should be used inside this loop. It means that program abandon rest instructions in this loop and goes to the first line just after the loop.

In other case, if you have a list of instructions and only few first instructions should be done in some loop's iteration (and the other instructions should not be done) then a special word `continue` should be used. It allows to go instantly to the next loop's iteration.

1.5 Random Numbers

It is possible to draw a number using some random generator. In the C language the simplest way do to this is to use the `rand` function, which definition is included in the `stdlib.h` library. Example usage is as follows:

```
#include <stdlib.h>
...
    int my_number = rand();
```

The minimum value is `0` and maximum depends on compiler, processor, and so on and it is given by `RAND_MAX`. If you want do draw a number from set 0-9, then you should do:

```
    int my_number = rand() % 9;
```

Try to run some program with this function. As you can figure out values given by the `rand` functions are always the same. It is because there is used some algorithm and the starting number is always the same. To fix this issue you can use system time, which will be different each time you start your program. To use a system time you can use `srand(time(NULL))` function – the `time.h` library is required as well.

1.6 Arrays

To define a new array some name of this array has to be used and a size of this array has to be given as well. Size is given in `[` and `]` braces and minimum value is 2 (it i possible to use 1-element array, however, it has no sense since it is easier to use a simple type then).

To define a new array of name `Students` which has 30 people (`int` type) the following instruction has to be provided into a source code:

```
int Students[30];
```

2 Exercises

1. Draw some random number in you program.
2. Ask user to give some number and compare it with the drawn number by your program. Inform user if he guessed it or not.
3. Draw algorithm of this program.
4. Modify your previous program – it should be possible to provide by user a new number till he does not guess the right one. Solve this problem using three different types of loops (write 3 programs).
5. Draw algorithm of these programs.
6. Right now try to count how many attempts user needed to guess the right number. Display this number in the terminal when user guessed the right number.
7. Draw algorithm of this program.
8. Ask user to give some number of integer values. Store all of them in some array and when user finish providing these numbers display them in the terminal in one row separated using Tab.
9. Draw algorithm of this program.
10. As previous, however, this time these number should be sorted from the smallest one to the biggest one.
11. Draw algorithm of this program.
12. Prepare some menu of your first game which allows user to choose some option.
13. Give some source code for each position of this menu.
14. Draw algorithm of this program.
15. Are you able to draw a "snake" board and a snake itself? Let's say right now the game will not depends on time by only on pressed keys by user. Randomly put some values on the board (in some places only, not everywhere). Each time the game starts these numbers will be placed in different places and will have different values.