

Prerequisites:

- Basic knowledge of using computer and text editor

Aims:

In this laboratory student will learn how to:

- Create a first program
- Create a new functions
- Mathematical operation on variables (changing values)
- Displaying text and values in the terminal

1 Introduction

1.1 Environment

To run C programming on Linux, there are needed at least:

- a compiler → we will be using the GCC compiler;
- a development tools and libraries → we will be using standard libraries;
- an editor → any text editor with syntax coloring will be fine, we will be using the gEdit or xed.

1.2 The First Program

Let's write at first your first program and then try to analyze it. Create at the desktop a new folder called *Lab01*. Open this folder and create a new file named *lab01.c* or alternatively you can just open the text editor and save a new file as *lab01.c*. **The extension of this file is very important!** Provide a source code from Listing 1.1.

```
1 #include <stdio.h>
2
3 int main()
4 {
5     // My first program in C
6     printf("Hello world!\n\n");
7     return 0;
8 }
```

Listing 1.1: *Hello world!* program

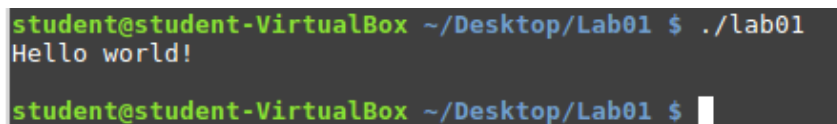
Right now open the terminal (command line) and navigate to the directory where you saved your file. You can use for that: *cd name_of_directory* and *cd ..* commands to go in and go out of a directory, respectively. To compile your first program just provide the following command (starting from gcc):

```
my_project_directory$ gcc lab01.c -o lab01
```

If there are some errors fix them (take a look carefully at Listing 1.1). If there is no error during the compilation process you can run right now your first program by providing the following command:

```
my_project_directory$ ./lab01
```

You should see *Hello world!* sentence written in the terminal (see Fig. 1.1). **Congratulations! It is your first program written in C!**

A terminal window with a dark background. The prompt is 'student@student-VirtualBox ~/Desktop/Lab01 \$'. The user enters './lab01' and the output is 'Hello world!'. The prompt is then 'student@student-VirtualBox ~/Desktop/Lab01 \$' with a cursor at the end.

```
student@student-VirtualBox ~/Desktop/Lab01 $ ./lab01
Hello world!
student@student-VirtualBox ~/Desktop/Lab01 $
```

Fig. 1.1: The result of the *Hello world!* program

1.3 Analyze of the first program

As you can see, your first program is not so complicated and it is doing nothing special. Do not be afraid because more advanced programs will be written later. Right now take a look more carefully once again at the source code (see Fig. 1.1). You can distinguish some parts which will be used by all of your future programs. A typical program can be build from the following parts:

- Preprocessor commands
- Functions
- Variables
- Statement
- Expressions
- Comments

1.4 Preprocessor commands

Here you can include some libraries and files which will be used in your program. In line 1 there is `#include <stdio.h>`. This is a place where additional file called `stdio.h` is used in your program. This file is localized in a system libraries path – we are using here `<...>`. If your header file will be localized in your program's folder (*Lab01* in our example) then instead of `<...>` you have to use `"..."` (in our example it will looks like `"stdio.h"`).

1.5 Functions

In line 3 there is `int main() {...}` function. This is the main function without which any program cannot exist. Later you will be writing other functions, however, this function cannot be omitted. It is the most important function and each program starts from it.

This function is called `main` and it returns some integer value (before name of this function there is a special word `int`). To return any value by any function a special `return` keyword should be used (in our example it is done in line 7 – see Listening 1.1). This function has also body given in curly braces `{...}` (see lines 4–8 in Listening 1.1).

You can find another function called in line 6:

```
printf("Hello world!\n\n");
```

However, definition (body) of this function is placed in another file due to usage of external libraries.

As you can see, function `printf` has one argument. In this case it is some sentence which will be written in the terminal.

1.6 Comments

In line 5 there is part of the source code which is not visible for the compiler. It is called a comment. There are two types of comments: one-line or multi-line.

- one-line comment → is done using `//` sign in some line and text placed after this sign till end of this line is not compiled (see line 5 in Listing 1.1);
- multi-line comment → is done using `/*...*/` signs in one or through many lines (see Listing 1.2).

```
10 /*  
11     This is  
12     a multi-line  
13     comment!  
14 */
```

Listing 1.2: Example of a multi-line comment

1.7 Types

There are a lot of different types which can be used for functions or variables. In our example the `int` type is used as a type of value returned by the `main` function (in fact value 0 is returned in line 7 – see Listing 1.1). Some often used types are as follows:

- `int` → integer, no decimal places
- `float` → floating point, with decimal places, for example 12.3978
- `double` → double precision floating point, with even more decimal places
- `char` → a single character, for example `'t'`
- `string` → chain of characters, for example word `"car"`

1.8 Variables

Variables are used to store some data. However, before some variable will be used it must be declared. Declaration means that a particular name will be used to store a particular kind of data in a computer memory. For example,

```
int a;
```

means that integer value will be stored in variable a. In this case value a has no value yet and before first usage it must be initialized, it means some value should be put into a. It is advised to put value 0 in place of declaration to avoid later some trashes as a variable's value and troubles in your programs.

2 Exercises

1. Write your name in the terminal.
2. Write your surname in the terminal.
3. Write your name and surname in the terminal, however, put each of them in a different line.
4. Draw an asterisk sign `*` in the terminal.
5. Write a new function which will be responsible for drawing a rectangle (build from asterisks) in the terminal.
6. Check for what are used following signs in text written in a terminal: `\n`, `\b`, `\t`, and `\r`.
7. Put value 12 into variable `my_variable` and display it in the terminal.
8. Create a new function `Triangle` which allows to calculate field of some triangle. Take into account function's arguments.
9. Check for what are used `%i`, `%f`, `%lf`, `%c`, `%x` in `printf` function.
10. Limit the number of decimal places for `float` type, use for example 4 places. Use `%.nf`.
11. Check options: `%m.nf`, `%0m.nf`, `%-m.nf`, and `%+-m.nf`.