This course has already ended.

« 1 Foreword Course materials 3 Data types and Variables »

ELEC-A7100 / 1. Basics / 2 Introduction to C-language

First round tasks as a `ZIP file.`

Introduction to C-language¶

The wikipedia article on C language gives a succinct summary of the relevant properties of C, and its relationship to other common programming languages.

C is a procedural language, where programs are built from functions and datas. It does not support separate namespaces, or classes. C program is written in a text-based source code file, which almost always use the *.c* - terminal file name. Typically a bigger program consists of a number of such files, that may be even structured into several directories. In addition to source code files with *.c* suffix, a C program usually uses header files with *.h* suffix. These contain variable definitions and functions prototypes needed by the C program, and this enable using function definitions that are external to a particular source code file. A C program takes these definitions into use with a series of #include preprocessor directives that are located in the beginning of a C source code file. There are two kinds of header files: user-defined headers and the headers needed for using system libraries. The `#include` directive format differs a bit in these two cases (e.g., #include "source.h" vs. #include `<stdio.h>`).

There are different ways of working with the C code. You can edit the source code files using a separate text editor such as **kate**, **emacs** or **vi** (all installed in Aalto Linux systems), and use the command line shell to compile and test the code. Alternatively, you can use an Integrated Development Environment (IDE), that has an integrated graphical user interace for editing code, compiling it, debugging it, and so on. Using either of these alternatives are possible on this course.

More information about compiling tools for different platforms can be found in Module 0.

If you find setting up development environment on your own machine too difficult, one option is to use browser-based tools such as **repl.it**.

These source code files are processed by a compiler and linker that produce a binary executable file understood by the underlying computer. While the text-based C programs are intended to be portable, i.e., the same code should work in different computers, the binary executable is specific to the architecture it was compiled for (for example, Intel-based 64-bit Linux). When moving the program code to a different machine, it therefore needs to be recompiled for that machine. Also, every time you modify the source code, you will need to recompile it. This is a significant difference to higher-level interpreted languages, such as Python.

Building an executable from C source code happens in the following distinct phases, in the following order:

- **Preprocessor**: processes macros, inclusion of header files, conditional compile instructions, etc, to prepare the source code for actual compiling into binary code.
- **Compiler**: compiles the preprocessed, text-based code into native binary object code. The object code still contains symbols for external references (for example, functions implemented in other libraries), and cannot be executed as such. If a C program is split into multiple source code files (marked with *.c* suffix), a separate object file (with *.o* suffix) is produced from each source code file.
- **Linker**: links together the different object files and resolves the references, producing the actual executable that can be run in the system.

Program executions starts only from **main** function which must be present in any one of the linked *.c* source files.

It is important to understand the above steps so that it will be easy to debug compilation messages you encounter in the future. Typically, the compiler will do all the steps in sequence automatically, but it may also be necessary to perform the above steps separately.

The two most commonly used compilers are **gcc** and **clang** where both compilers do same thing but produces slightly different information for example when an error occurs. These compilers may either use a command line interface or IDE to display the results.

A lot of addtional information on C language can be found on the Internet (or books). One of the starting point is Wikipedia, which tells the history of the language, and more a general description.

The general source of information used by programmers is Stack Overflow, where you can find response to a variety of problems related to programming. Together with Google (or other search services), it is possible to find solutions to lot of problems encountered during programming. However, use of such search may be deceptive: in addition to the resolution of an individual problem, it is important that you understand more deeply what was the problem, and why the solution worked. However, you will find interesting discussions, interesting questions and answers linked to several other materials here and there.

The First C-Program¶

Below is a very simple C-program. It prints one line of text to screen and then exits.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      /* The following line will print out some text */
6      printf("Hey! How are you?\n");
7
8      return 0;
9  }
```

The first line tells that the program uses the definitions of **Standard I/O** functions provided by the standard library. We therefore, included in the program **stdio.h** - header file, in which standard I/O functions are defined. This header is needed, for example, for printing text on the screen. We will look into these functions in more detail later. This line is handled by preprocessor, which is expressed in the beginning of the line `#` - mark.

Every executable C program must have the **main** function that the system calls when program execution starts. In the above example, **main** function starts on line 3. **int** and **void** present on line 3 will be discussed later.

The definition of the function is included in a block enclosed with braces (from line 3 to 7), and consists of one or (usually) more statements. Each statement ends with a semicolon. In this example we only have one statement on line 6, calling the printf function that outputs text on screen. On line 5 there is a descriptive comment enclosed inside `/*` and `*/` markers. Such comments can contain any free-form text, and are intended for programmer to leave clarifying notes for the reader of the code. Compiler ignores anything inside the comment marks.

The **printf** function shown in the example outputs text on the screen. It is defined in the Standard I/O library (stdio.h), which is included in the beginning of the program with the **#include** directive. If stdio.h is not included, then program compilation will result in failure as the compiler do not know the definition of **printf** function.

In this example the **printf** function contains one parameter (inside parenthesis), which is a string that is written to the screen. Printable text (i.e, a string) is given in quotation marks.The string ends with a newline character `\n`, causing the following output to be printed on the next line. The 'printf' function call, like all other C statements, ends in semicolon (;). Forgetting the semicolon is a common mistake for beginners that causes compilation of source code to fail.

Function call in a program can be identified by the function name always followed by parentheses. Within the parentheses is the list (comma separated) parameters, which are supplied to the function. It is also possible that the function has no parameters at all. In this case, the printf function call on line 6 has one parameter which is a string. C language is very strict about this format.

You can try running the program. If compiled successfully and the program is executed, it should print the text on the screen. Try to change the program in some way, then compile and run it again.

Task 1.1: Hello World¶

Modify the above program so that it prints three lines as follows:

```
Hey!
How are you?
I am fine, thank you.
```

Points 8 / 8 My submissions 4 Deadline Tuesday, 8 June 2021, 19:59
To be submitted alone

⚠ This course has been archived (Saturday, 31 December 2022, 20:00).

Hello World

Select your files for grading

main.c

Choose File No file chosen

Submit

Compilation tools¶

Next you can learn how to use a compiler: Copy the above sample program to your text editor, for example, save the file as "hello.c". If you are using command line, you can try, for example, the following command

`gcc -o hello hello.c`

Now, when you run the program by typing " `./hello` ", you should see the following output:

```
Hey! How are you?
```

The server used to grade the exercises typically runs the following command:

```
gcc -g -Wall -std=c99 -o test source.c
```

where

- "gcc" is the name of the compiler
- "-g" produces debug information that will be helpful if analyzing the program with debugging tools
- "-Wall" informs the compiler that we want to see all the relevant warnings. If any warnings are listed, they should be fixed since most likely they indicate a programming mistake. Also, in the programming tasks 50% of the points are deducted if there are any warnings.
- "-std=c99" informs that we are following the C99 standard in this application
- "-o test" informs that the compiler produces an executable file named "test"
- "source.c" is the name of the source code file

When using an IDE, command line isn't used. Instead, a new application project is created, you will write the needed code to a ready-made layout (find the main function), and use some buttons to compile and run the code.

A major feature in C - in contrast to, for example, Python - is that C compiler does not care about the design of the program at all (though preprocessor cares, but more on that later). For example, you can write many lines of program separated by a semicolon in one line. Nevertheless, it is important that the programs are formatted in beautiful and consistent manner, so that reading would be easy. When the program size increases, and if it is poorly designed, then it is very difficult to understand the program. In most cases, the same code will be used and modified by many people, so code readability is important.

Some good programming practices:

- **Indent code** based on program blocks: whenever a new block of statements are started, indent the start of the line by a consistent space. Always use consistent spacing (for example 4 spaces for every indent, or one tab for every indent)
- apply clear and **consistent style in naming** of variables and functions. They should be descriptive enough, but very_long_local_variable_names are usually not a good idea.
- divide program in logical, not-very-long **functions**, instead of writing everything in the main function.
- **use comments** to explain logic that might not be easy to follow for someone else by just reading the code.

Text editors, especially in IDEs, often try to assist the programmer in following consistent style, for example by applying indentation automatically (sometimes to the point of irritation, if the programmer disagrees with the style).

First Program¶

The video below explains (in Finnish) how the first real programming task (i.e. 1.5 below) was made of Ubuntu Linux - based system. If the programming and the necessary tools are new to you, we recommend you watch the full video. Tasks can be solved in other operating systems and tools, such as Visual Studio, Windows, or Mac XCode or command-line tools.