GROUP T LEUVEN CAMPUS

Operating Systems
Programming with C: lab 1

Bachelor Electronics/ICT

Course coordinator: Bert Lagaisse
Lab coaches:

- Jeroen Van Aken
- Yuri Cauwerts
- Ludo Bruynseels

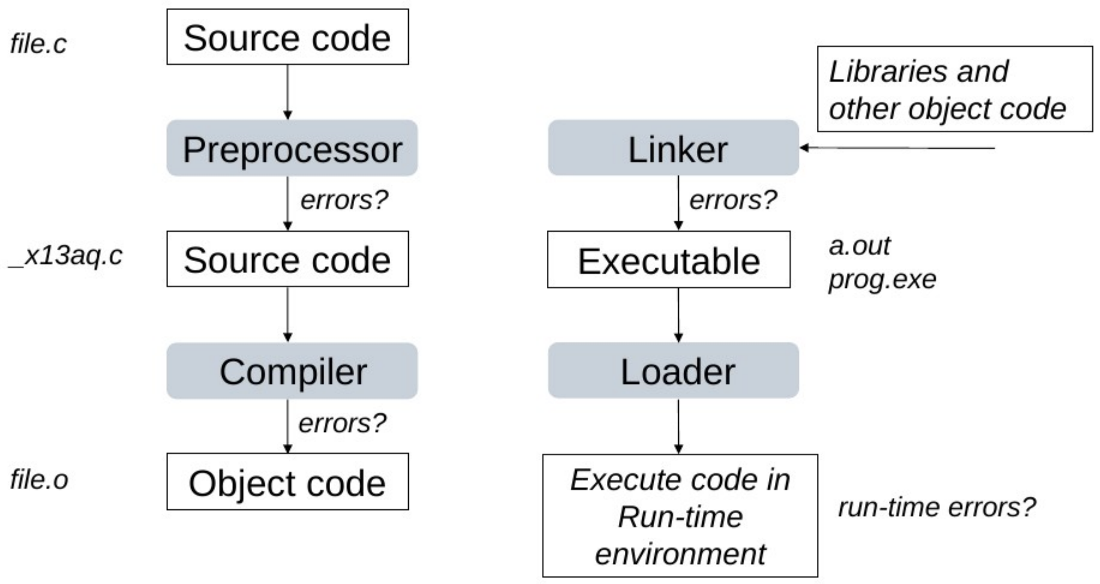
Last update: October 12, 2022

Programming with C: Lab 1

Lab target: writing basic C programs, build them with gcc, run and test the programs.

Some basic repitition: introduction to GCC

The GNU C compiler collection (GCC) is used for **all** programming exercises of this course. Gcc is one of the most used C/C++ compilers, not only on Linux where it comes pre-installed, but on many other platforms as well. GCC is more than only a set of compilers: it's really a collection of libraries and tools like gprof, gcov, gdb, etc. The entire GCC is open-source. Visit the GCC home page[1] for more information.
C program code must go through a number of processing stages before it can be loaded and run. First, C code must be preprocessed. Next, the compiler transforms the code to object code which then can be linked into an executable. The full processing chain is depicted below.



More details on the different processing steps will be given during this course. Hence, be patient. For now, it suffices to know that given a text file 'main.c', an executable can be compiled on the command-line prompt ($) with the command :

```
$ gcc main.c
```

After running this command, there are two possible results. The first - and probably the most likely - result is that an error occurs, typically because of compiler/linker errors. You must first solve all errors before you can retry. Hopefully, after some retries, the other possible result pops up on your screen: a successfully compiled program. In that case, you should find an executable file 'a.out' in the same directory.

1 gcc.gnu.org

Check if this file has executable permission and if needed, modify the permissions of this file. Finally, you can execute the program with the command './a.out'.

```
$ chmod +x a.out
$ ./a.out
```

There are two more options worth knowing before you start programming.

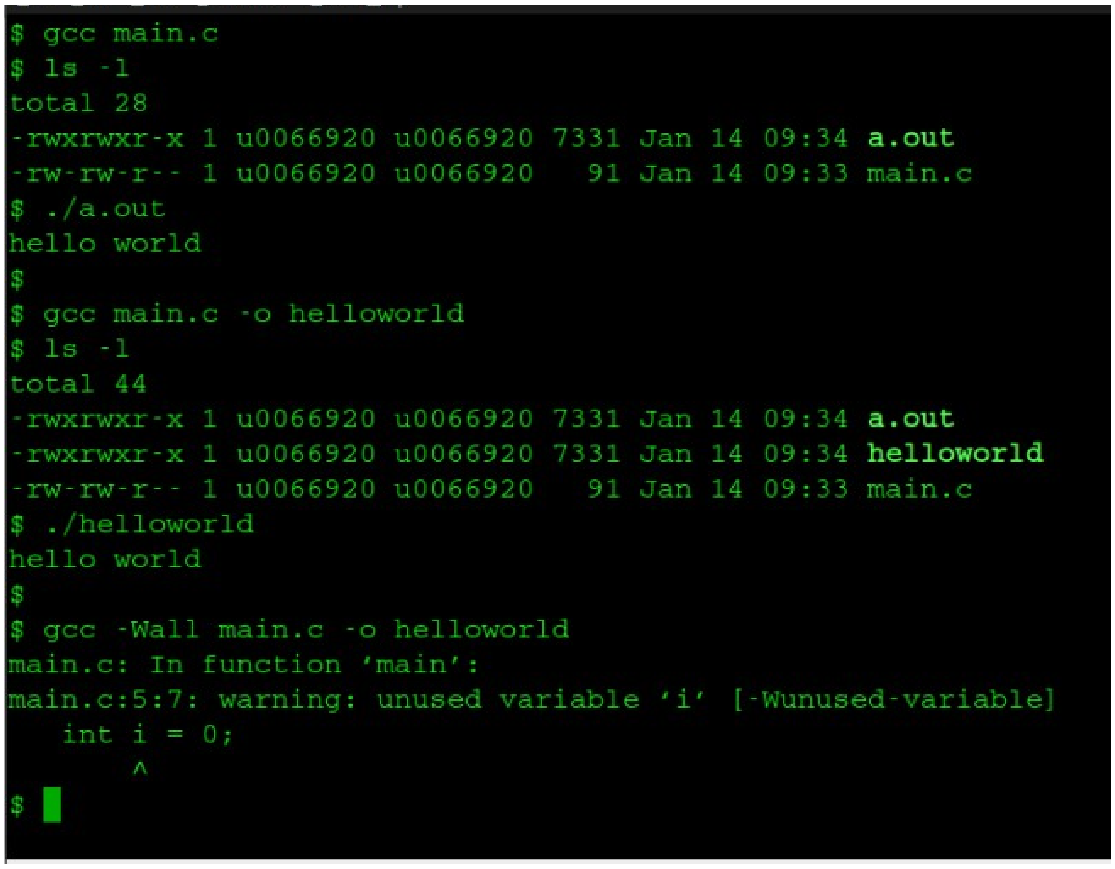
- First, the '-o' options allows you to give a more meaningful name to the executable file. For example, 'gcc main.c -o helloworld' creates an executable file called 'helloworld', assuming there were no errors during compilation.
- Second, the '-Wall' option turns on many warning flags used by gcc to check for potential mistakes. Do **not** neglect warnings because in many cases warnings are forerunners to some future problems.

```
$ gcc -Wall main.c -o helloworld
```

Therefore, some programmers and companies treat warnings at the same level as errors. We also insist that you **always** compile with '-Wall' and that you only consider the program successfully compiled when **no** more warning messages show up. The 'all' in '-Wall' suggests that all warnings will be captured, but that's **not** true. In the man pages of 'gcc' it is explained which warning flags are set and which aren't set with the '-Wall' option.

```
$ man gcc
```

The most important described commands are illustrated in the screenshot below.



```
$ gcc main.c
$ ls -l
total 28
-rwxrwxr-x 1 u0066920 u0066920 7331 Jan 14 09:34 a.out
-rw-rw-r-- 1 u0066920 u0066920   91 Jan 14 09:33 main.c
$ ./a.out
hello world
$
$ gcc main.c -o helloworld
$ ls -l
total 44
-rwxrwxr-x 1 u0066920 u0066920 7331 Jan 14 09:34 a.out
-rwxrwxr-x 1 u0066920 u0066920 7331 Jan 14 09:34 helloworld
-rw-rw-r-- 1 u0066920 u0066920   91 Jan 14 09:33 main.c
$ ./helloworld
hello world
$
$ gcc -Wall main.c -o helloworld
main.c: In function ‘main’:
main.c:5:7: warning: unused variable ‘i’ [-Wunused-variable]
   int i = 0;
       ^
$
```

General hint: There exist very nice references to the C standard library with example code for many library calls. For example:

http://www.java2s.com/Code/C/CatalogC.htm.
https://www.tutorialspoint.com/c_standard_library/index.htm

Exercise 1: create and compile a program and push it to git

Open the Linux console terminal (using ssh or directly with Virtual Box), go to your home directory and clone your git repo for this course. You can find your repo on the https://gitlab.groept.be, also don't forget to add your SSH key in your user settings on the gitlab website.

In a terminal you can clone the repo by:

```
$ git clone git@gitlab.groept.be:osc/xxx.yyy.git
```

(replace xxx.yyy with your username/firstname.lastname on gitlab.groept.be)

This repo contains some basic files and a branch for every lab. Use `git branch` to see what branch you are currently on, use git branch -a to see what branches exist on the server. Then switch to the branch for this lab using `git switch clab1gcc` (on older versions of git the switch option is not recognised, you can then use `git checkout clab1gcc`, which results in the same) Now create a folder ex1 (mkdir ex1) and a main.c file (in the ex1 folder) with the 'hello world' app and compile it using `gcc main.c`.

With `git status` you can check what files are created/altered and waiting to be committed to the repo. Notice that `a.out` is not in the list. You should only commit source code and useful files. This is why we added a hidden file `.gitignore` (don't forget the . in the beginning) in this branch. This is a text file that can be edited with nano or vim.

To make life less difficult we add a `Makefile`, this is a text file interpreted by the `make` command. In this file you can add items with each a list of commands, e.g. :

```
ex1: main.c
      mkdir -p build
      gcc -Wall -Werror -o ./build/ex1 main.c
      ./build/ex1
```

with `make ex1` you can now execute the compile and run commands at once.

Use `git add main.c` or `git add *.c` to add your code to the repo. Now commit your code with `git commit` and add a meanigfull commit message. Then push your code back to the gitlab server with `git push`.

Exercise 2: datatype size, type qualifiers

The memory size of the built-in data types int, float, etc. is system dependent.

- Write a program that prints the memory size of int, float, double, void, and pointer using the `sizeof()` library function.
- What happens to the memory sizes when you use the type qualifiers `short` and `long`?
- Find out if a char is an unsigned or signed data type on the system you use.
- Commit and push all your code to the git repo

Exercise 3: strings

There is no built-in data type 'string' in C. As a matter of fact, strings in C are defined as null-terminated arrays of char. For example:

```
        char name[4] = “luc”;
```

The array 'name' defines a null-terminated string which means that name[0] = 'l', name[1]='u', name[2]='c' and name[3]='\0'. The termination '\0' is very important as all string manipulation functions will use this character to detect the end of the string. Also notice the difference between single and double quotes: 'c' indicates the constant char c, but “c” defines a constant string containing only one character being c. But if you store the string “c” you need a char array of length at least 2 because also the '\0' char must be stored to terminate the string.
Fortunately, to ease the pain of not having a built-in data type string, the C standard library provides an entire set of string manipulation functions. The target of this exercise is to explore a number of these functions.

Write a program that declares the string variables *first*, *second*, *name* and *str* both of some maximum length MAX.

- Use `scanf` to read out your first and second name in *first* and *second*, respectively.
- Convert your second name to all upper case chars and store the result in *str*.
- Apply `strcmp` on *second* and *str* and make sure you understand the result. Is there another compare function ignoring the case of chars?
- Concatenate your first and second name into *name*; be aware that there is an unsafe and a safe string copy and concatenation function. Print the result.
- Read your birth year in some int *year*.
- Concatenate *first*, *second* and *year* into *name*. This time, use `snprintf` to do the concatenation. Notice that there is also an unsafe version of `snprintf` being `sprintf`.
- Finally, use `sscanf` to read from *name* the *first*, *second* and *year* values again. Print the result on screen to check the result.