1 Lab 1: Introduction to Linux and C Programming

This laboratory requires the following equipment:

- C compiler
- Make tool

The laboratory duration is approximately 3 hours. Although this laboratory is not graded, we encourage you to take your own personal notes as the lab verification test might leverage results acquired during this laboratory session. For any questions, please contact us at **sis-ta@groupes.epfl.ch**.

1.1 Information

In the following text you will find several exercises and questions.

- The notation **S** means that the question can be solved using only additional simulation.
- The notation **Q** means that the question can be answered theoretically, without any simulation.
- The notation **I** means that the problem has to be solved by implementing a piece of code and performing a simulation.
- The notation **B** means that the question is optional and should be answered if you have enough time at your disposal.

1.2 Outline

This lab is intended to make you familiar with the Linux environment and C programming. The lab machines have both Windows and Linux installed – the Linux distribution that we use in this course is called *Ubuntu*. If your machine is currently running Windows, simply reboot it and select Ubuntu Linux from the boot menu. In this lab, you will first familiarize yourself with this Linux environment, and subsequently compile and run a few simple C programs.

1.3 Getting Started

Log in with your GASPAR username and password. Open a terminal by using the keyboard shortcut `CRTL+ALT+t`. When you open a new terminal, you start in your home directory (`/home/username`). Type `ls` to see the files in your home directory. Note the directory `myfiles` (located in `~/Desktop/`). **Always store your work in this folder**. Anything stored just in `/home/username` will be deleted when the computer is restarted.

To start with this lab, you will need to download the material available on Moodle. Download `lab01.tar.gz`. By default, files are downloaded to the Download directory (`/home/username/Downloads`).

Now we are going to extract the lab archive. To do so, start a *Terminal*. First move the file you just downloaded into your `myfiles` directory using the `mv` command. Next, use the `cd` command to move to the directory that now contains `lab01.tar.gz`. It is worth noting that filenames can be completed automatically by using Tab. Once you are at the correct location you can type: `tar xvfz lab01.tar.gz`. This command will create a new folder `lab01` in which the material for this lab can be found. Only then navigate into this newly created folder by using `cd lab01`, and leave the Terminal open at this location. For editing files in this lab, we recommend you to use `gedit`. Run it from the terminal using the command `gedit filename.c &` (the ampersand causes it to run in the background).

It is worth noting that to avoid problems it is recommended to avoid folder and file names with spaces in them. If separations are needed to make a file or folder name readable, underscores are suitable replacements.

2 Lab: Linux Warm-up

For this exercise only, do everything in your `~/Desktop` directory. This is because `myfiles` is hosted on a windows server where file permissions are different.

1. (Q): In your current folder `lab01`, create a new folder called `branch`. Move to the newly created folder. Then create a folder called `leaf` and move to that folder. Next, go back to the `branch` folder and finally return to the original folder `lab01`. (Hint: the parent folder is "`..`", so if you want to return to the parent folder, just type `cd ..`).

2. (Q): Type `man ls` into the terminal (press '`q`' to close the man page). How can you get the details of the contents of the directory you are currently in? What is the argument that you need to provide in order to find out when the folder you are currently in was last modified? How do you find out if an item is a directory or a file?

3. (Q): To create an empty document, type `touch my_document.txt`. Check the current file permissions using `ls -l`. Now, you would like to make this document write-protected. In order to do this, we use the command `chmod`. Again, use the manual pages to find out what argument you must use. For more on file permissions, check out:
http://en.wikipedia.org/wiki/Filesystem_permissions#Notation_of_traditional_Unix_permissions

4. (Q): In the previous question, you created a non-writable file. Open it in your favorite text editor and type 'Hello'. Try to save the file. What happens, and why?

5. (Q): If you would want to make `my_document.txt` writable for yourself while leaving it protected for others, what is the command you have to use?

6. (Q): In your lab archive, you have a program called '`my_program`'. What do you have to type to start it? After you started the program, open a second terminal and type `ps aux` (this will list all the processes currently running). Now find the process which corresponds to '`my_program`'. Use its process ID to kill it. Which command did you use?

7. (Q): In the tutorial you saw the command `grep`. Using the pipe '|' together with the command `grep` repeat the above exercise to find the process ID of '`my_program`' faster (without having to read the whole list of processes).

3 Lab: Variables and Makefiles

8. (Q): Determine on paper what the output will be.

```
int a = 3;
int b = 4;
if (b < 4.5) b--;
else b++;
printf("%d\n", a/b);
```

9. (Q): Determine on paper what the output will be.

```
int i;
for (i=2; i<4; i++){
   int i = 5;
   printf("%d\n",i);
}
```

```
printf("%d\n",i);
```

10. (Q): How do you print a float with only two digits after the decimal point using the `printf` command? What is the `\n` *escape sequence* used for in the `printf` functions?

11. (I): For this question you need to work with the files in the `variables` folder. `main.c` contains a program to output the result of the code segments which you have previously seen in questions 8 and 9. Each code segment is implemented in an individual file and then included in `main.c`. To compile this code which has a number of dependencies we will use a Makefile. Try to run the Makefile using the command `make`. Open the Makefile and try to understand what each line of this file is trying to do. What is the purpose of the new files generated by the `make` command? (`main`, `test1.o`, `test2.o`)

12. (Q): Fill in the missing labels in Figure 1. The vertical boxes correspond to commands the Makefile executes. What are they?

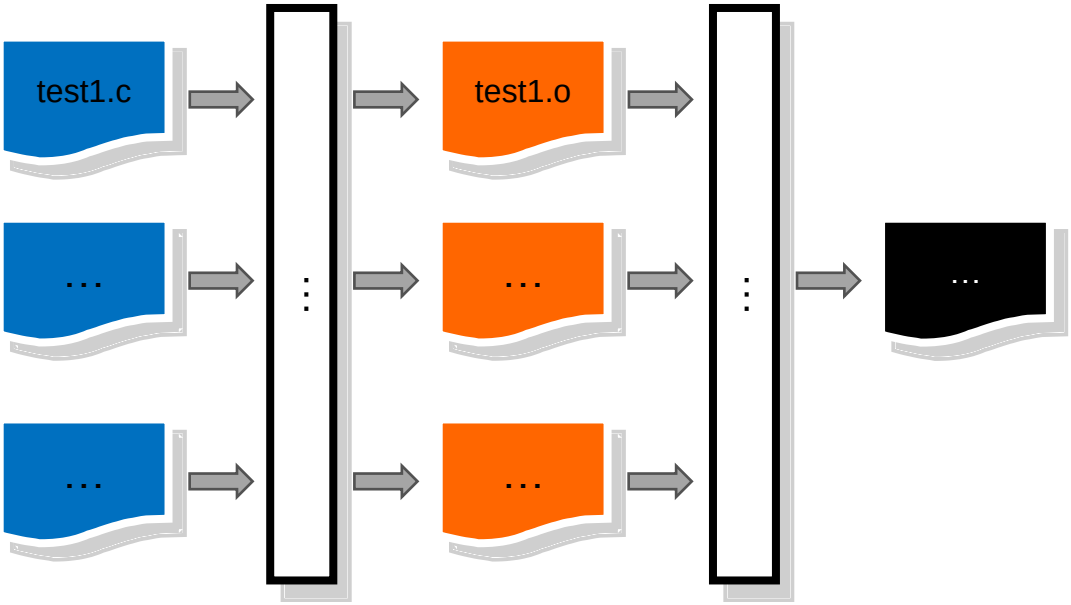

Figure 1: Compilation diagram

13. (I): Execute the program generated in question 11. Are the outputs of the sets of code what you expected? If not, why?

14. (I): If you were asked to add another test in a new file `test3.c` and then output its result in `main.c` how would you modify the Makefile? Try to run `main` after this modification and see whether it works properly.

4 Lab: A simple program to calculate your age

15. (Q): What is the purpose of the function `scanf` found in C? Provide an example that uses this function. Where did you find this information?

16. (I): Now, you will go through the steps of writing a full program that calculates your age when given your birth date. (Hint: For `scanf` to work it needs the pointer to the variable: https://en.wikipedia.org/wiki/Pointer_(computer_programming). The pointer can be found by using `&` in front of the variable as in `&my_variable`)

a) Write the code that asks for your birth year, month and day and repeats it in a sentence.
The execution of that program should look like the following:

```
> ./calculate_age
Enter your birth year: 1985
Enter your birth month: 12
Enter your birth day: 3
You were born on the 3rd of December, 1985.
```

b) Now complete your program so that it calculates your age, and prints it into the terminal. (Note: you will have to hard-code today's date into the program).
The execution should look like the following:

```
> ./calculate_age
Enter your birth year: 1985
Enter your birth month: 12
Enter your birth day: 3
You were born on the 3rd of December, 1985.
Today, you are 24 years old.
```

17. (B): Add code to your program so that it asks for a second date (same format as first one) and calculates the difference in days between the two. Take leap years (which happen only when the year is a multiple of 4 but not if multiple of 100. If the year is multiple of 400, the year is a leap year) into account.

18. (I): Implement a program which asks the user for two numbers and returns the result of the first number modulo the second one (the remainder of the Euclidean division of the first by the second number) without using the % operator.

19. (Q): What is the result of the program below? (Hint: beware of invisible characters when copy-pasting the code to verify your answer with the computer. To do so, copy the code into a simple text file and verify that there are no pdf characters. Particularly quotes usually need to be replaced.)

```
#include <stdio.h>

void main(void){

      int res = 0, i = 0, j;

      do{
            if  (i > 9)  break;

            for(j = i; j < 10; j++){
                  res++;
            }

      }while(++i);

      printf("%d",res);
}
```