Lab 2: C Programming II

This laboratory requires the following equipment:

- C compiler
- Matlab

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 continues to exercise your C programming skills, and also aims to show you differences with other programming frameworks (Matlab). Moreover, this lab covers functions, Makefiles and compilation in general. The lab machines have both Windows and Linux installed – the Linux distribution we will 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.

1.3 Getting Started (Short reminder)

To start with this lab, you will need to download the material available on Moodle. Download *lab02.tar.gz* in your personal directory. Now, extract the lab archive (you can type: `tar xvfz lab02.tar.gz`.)

2 Compilation & Functions

Imagine a temperature sensor that monitors the danger of fire. The sensor sends temperature readings to a central processing node every one second. These readings contain the temperature measurement and the time. The central node reads these values through a text file `messages.txt`. Then, it has to issue a warning when the temperature is above 35 degrees. The files `sensors.c`, `sensors.h`, and `sensors_main.c`, implement the main tasks of this central node, they are located in the folder *sensors*.

1. **(Q)**: Open `messages.txt` and check its contents. Each line in this file is one temperature reading that has been sent by a sensor. The first number in each line is the time (in seconds) and the second number is the temperature value. Which character separates the two numbers?
2. **(Q)**: Open `sensors_main.c` and describe what it is doing.
   - The `main` function is declared as `main(int argc,char *args[])`. Here, `args` stores the arguments passed on to the program while executing it. For example, if you have a program called my_program, and you need to enter someone's name before executing the program, you would pass that name as an argument:
     `./my_program` **`donald_duck`**
   - Check in the code and see where the arguments (`args`) are used in `sensors_main.c`. What is the name of file that will be opened by the `fopen` command?
   - How do we obtain the sensor readings from the file? (check the functions in `sensors.c`).

3. **(S)**: Compile the code with gcc using the following command:
   `gcc sensors_main.c sensors.c -o sensors_main`

- Note that you have to mention all the C files that include parts of your code (called dependencies) in front of the `gcc`, otherwise it will give you compilation errors.
- The output of this command (indicated by `-o`) is a file named `sensors` that is an executable file. Run the executable file by typing `./sensors_main messages.txt` Note that with this line, you are giving `messages.txt` as an argument to your code.
- You should see multiple lines with this text: *"Message received but print not implemented yet!! format should be: at time <time> sensor sent temperature value of <temperature_reading>".* Where does this message come from?

4. **(S)**: Compile the program using the provided Makefile, by running `make` in the terminal. Does it compile? What is the problem? *Hint: open the Makefile in an editor and check missing file dependencies in the Makefile.*

5. **(I)**: Edit the `Makefile` as following: replace line 10 with
   `sensors_main: sensors_main.o sensors.o`
   Now compile the program by running `make` in the terminal. Does it compile? What is the name of the output executable file?

6. **(I)**: Now we want to print the time and the temperature values to the terminal, in `sensors_main.c`. There are already two functions implemented in `sensors.c` that have been called in `sensors_main.c`. Find these functions in `sensors.c` and explain what they do. Find the prototype of these two functions in `sensors.h`. Why are the prototypes defined in the *.h* file? Check how these two functions are called in `sensors_main.c`. Modify line 36 of `sensors_main.c` in order to print the values of time and temperature. Compile and test your program using your new Makefile (don't forget to `make clean` before compiling with `make`). Remember you always need to compile after making any change in your C code.

7. **(I)**: You must have noticed that the warning feature of the central node (when the temperature passes the 35 degrees threshold) is not yet implemented. Now you will do it following these steps:

- Implement a new function, called `check_temp`, in `sensors.c`. This new function takes a temperature value as an input arguments, and checks whether it is above 35 degrees. If it is above the threshold, a warning should to be printed on the screen (using `printf`).
- Insert the prototype of the new function in the *.h* file (`sensors.h`).
- Call the `check_temp` function inside your main program (`sensors_main.c`).
- Compile using the Makefile and test your program.

3 C versus Matlab

In this exercise, you will write programs to solve similar tasks, in both Matlab as well as C. Note the differences. The code needed for this section is in the folder *matlab.*

3.1 Coding aspects

8. **(I)**: Open MATLAB by typing `matlab` in a terminal, and create a script `matlab_test.m` that contains the following code. You can run the script using the green arrow button on top of the editor.

```
text = 'this is some text';
if(text == 'this is some text')
    a = 1;
else
    a = 2;
end
```

Write down a C code `c_test.c` which does the same operation.

Hint: to initialize a string variable do as follows:
```
char text[20]="This is some text\0";  // Initialize a string variable
```

To compare two strings, use strcomp as follows:
```
strcmp(text,"This is some text\0");  // Compare two strings
```

Compile using gcc and test your program. How many lines of code did you use? Did you need to compile the MATLAB script? *Hint: check out the function strcmp() provided by string.h. Tutorials are provided in http://www.cplusplus.com/ specially you can check http://www.cplusplus.com/reference/cstring/strcmp/*

3.2 Loops

9. **(I)**: Equation (1) describes a simple formula. Open the code skeleton `matlab_loop.m` in Matlab and complete the program to calculate the result *S. (Hint: In Matlab, modulo(a,b) is* `mod(a,b)`*. The structure of one for-loop is already provided to you)*. You can run the file by typing `matlab_loop`; in the Matlab command window. **(Note: As the file contains a function, and not a script, you cannot use the green arrow button on top of the editor to run it)**

$$S = \sum_{i=1}^{1000} \sum_{j=1}^{1000} \sum_{k=1}^{1000} i \cdot j \cdot k \cdot modulo\big((i+j+k), 2\big) \qquad (1)$$

10. **(Q)**: What result did you obtain? The program also prints how much time it took to produce the result. How long did it take?

11. **(I)**: Now do the same implementation in C. Open the code skeleton `c_loop.c` and complete the code to calculate Eq. (1). (*Hint: In C, modulo(a,b) is* `a%b`*. The structure of a for-loop is already provided to you*). Compile your program with gcc (by using the command `gcc c_loop.c -o c_loop`), run it (using the command `./c_loop`). Ensure that you have the same result with both Matlab and C. Note the time that it took to execute.

12. **(Q):** Which program is faster? Why?

3.3 Matrix Multiplication

$$P_{ij} = (A \times B)_{ij} = \sum_{k=1}^{M} A_{ik} B_{kj} \qquad (2)$$

13. **(I)**: Equation (2) describes the formula for the multiplication of two matrices, where *M*=500 and is the number of rows and columns in *A* and *B* (square matrices). Open the code skeleton `matlab_matmult.m` in Matlab and complete the program to calculate the result *P*. *(Hint: In Matlab, simply use A*B)*

14. **(Q)**: What result did you obtain? The program also prints how much time it took to produce the result. How long did it take?

15. **(S)**: Now we do the same implementation in C. Open the code `c_matmult.c`. This code is already completed. Try to read and learn how to multiply two matrices in C. Compile the program with gcc, run it and note the time that it takes to execute. For more information on the function `rand()` used to get a random number, check out http://www.cplusplus.com/reference/cstdlib/rand/?kw=rand

16. **(Q)**: Which program is faster? Why? Which program is more complex to understand?

4 Bitwise Operations and Structures

Bitwise operations are a fast and simple way to manipulate values on a computer at the bit level (0s and 1s). On low cost processors, such as the ones used in embedded systems, they allow to implement operations such as division and multiplication fast and efficiently since they require less machine resources.

17. **(I)**: What is the result of the bitwise operations below in C? Write a C-code that displays these values. Is the result what you were expecting?

```
17 | 4 = ?          17 & 5 = ?          17 ^ 5 = ?
```

Hint: use binary notation; here below you will find an example of code snippet:

```
#include <stdio.h>
int main(void) {
       int a = 0b00000100;  // Writing value of 'a' in binary.
                            // What is the corresponding decimal value?
       int b = 0b00000010;
       int c = b | a;       // result will be 0b00000110
       int d = 0x0A;        // In Hex. What is the value in decimal?

       printf("a= %d, b= %d, c= %d, d= %d \n", a, b, c, d);
}
```

18. **(Q)**:Considering that an integer has 2 bytes length (16 bits), what is the result of the following integer expression? (Note: '0x' denotes 'hexadecimal' numerical system)

```
(8 >> 3) + (~0xFFFFFFF5 << 4) = ?
```

19. **(I)**: Go to the folder *sensor*s. In `sensors_main.c` declare a `struct sens_msg` that contains the time of the sensor reading and the temperature reading. Alter the code of `sensors_main.c` in order to use this structure instead of the variables `tm` and `temp`. Compile and test your program.

20. **(Q):**Consider the following declaration of structure s:

```
typedef struct t_sensor {
    int data[100];
} sensor;

int main(void){
    sensor s;
}
```

We want to print the 65th element of s. Which one of the following lines of code is more correct for performing that task:

a) `printf("%d",data[65]);`
b) `printf("%f",data[64]);`
c) `printf("%d",s.data[64]);`
d) `printf("%f",s.data[64]);`

5 A Practical Example: Outlier Removal

From time to time, sensors transmit wrong values (outliers), which might tamper with the decision on the processing nodes. In this exercise, you will write some code to remove outlier data points from a temperature data set, extracted by one of the previous temperature sensors, written in the file `data.txt`. We have prepared a program for you which reads the sequence of values from the file and processes those values. The code needed for this section is in the folder *outlier*.

21. **(B)**: Type `make outlier_removal` in the terminal and execute your program as follows: `./outlier_removal 1 data.txt out.txt`. The file `data.txt` is in the lab02 folder. What does the program do when its first argument is 1? Plot the values in `data.txt` and `out.txt` in Matlab (use the `load` command from Matlab).

22. **(B)**: Modify the function `process_values` in `outlier_removal.c` such that when the program is called with its first argument equal to 2, the output is the result of an outlier-removal function. To help you get started, look at this Matlab implementation of the outlier-removal:

```
% Import the sample data
load data.txt;

% Calculate the mean and the standard deviation
mu = mean(data)
sigma = std(data)

% the location of outliers
outliers = abs(data - mu) > 3*sigma;

% Remove outliers
data(outliers)=[];
```

Hints: You need to take the following steps:

1. *Calculating the mean of the data: you need to write a for-loop, sum up all the data and divide it by the number of data points.*
2. *Calculating the standard deviation of the data. To calculate the standard deviation, you can look up the formula online: http://en.wikipedia.org/wiki/Standard_deviation, To calculate the square root in C, use the `sqrt(x)` function and include `math.h`*
3. *Write a for-loop (like the one already written in the C code) and copy input data to the output if the data is not an outlier (i.e., `fabs (data-mu) < 3*sigma`)*
4. *Compile your code with `make`. Make sure the compile process does not report any error.*

23. **(B)**: Run your function with `./outlier_removal 2 data.txt out.txt`. Your filtered data is saved in `out.txt`. Again, plot your data in Matlab and verify that the outlier removal worked. *Hint: use `hold on` in Matlab to have several plots in the same window.*