Lab 5: C Programming

This laboratory requires the following equipment:

- C compiler + debugger
- Geany
- 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 exam 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 or executing the code.
- The notation Q means that the question can be answered theoretically, without any simulation or code execution.
- The notation I means that the problem has to be solved by implementing a piece of code and performing a simulation or executing the code.
- 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 bring you some reminders in C programming (IDE, compiler and debugger) and to highlight also some of the differences between interpreted programming languages such as MATLAB and compiled ones such as C. In particular, this lab covers the following topics: basis of functions; compilation and Makefiles; memory management (string, structure, pointers); debuggers.

1.3 Getting Started

This lab is the only one focusing exclusively on C programming. We assume that the majority of the concepts treated during this lab are known from the previous course ENG-270 Informatique pour l’ingénieur de l’environnement. Nevertheless, it is useful to have a quick look to the new website of the ENG-270 course (https://coursc.ch) as a reminder.

1.3.1 Installation

If you don’t have set up your computer for the C lab yet, download the zip archive provided on Moodle (https://moodle.epfl.ch/pluginfile.php/2842793/mod_resource/content/1/setup_c_lab.zip) and follow the installation guidelines for your specific OS (SIS_20-21_C_lab_{windows, mac, ubuntu}). If you have still some issues with the installation, you can try to use the online virtual machine provided by EPFL (https://vdi.epfl.ch/) but it might be slow and unstable depending of your internet connection thus we highly recommend you to install the tools on your own computer or laptop.

1.3.2 How to open a terminal

Mac

If you have the spotlight button in your menu bar, just press on ⌘+ Space and type Terminal. Click on the icon to open the terminal.

Note: If it is not working try one of the options described on this page (https://www.idownloadblog.com/2019/04/19/ways-open-terminal-mac/).
Ubuntu:
Press on Ctrl + T.

Windows:
Click Esc + R and enter cmd. Click on the OK button. This will open the terminal of Windows.

1.3.3 How to change the settings of Geany

![Image of Geany settings](image)

![Image of Geany build commands](image)
1.3.4 How to Compile your program with Geany

1.3.5 How to build your program with Geany

1.3.6 How to run your program with Geany
1.3.7 How to clean your program with Geany

1.3.8 How to debug your program with Geany
1.3.9 Geany documentation

If you are interested to learn other functionalities of Geany, you can have a look at its online documentation (https://www.geany.org/manual/current/index.html).
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): Start Geany, 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 function?
   - How do we obtain the sensor readings from the file? (check the functions in sensors.c).

3. (S): Change the settings of Geany (see Section 1.3.3) and in the build section add the following command:
   ```
   gcc sensors_main.c sensors.c -o sensors_main
   ```
   Press the build button to start the compilation (see Section 1.3.5). Note, you can also compile your files using directly your terminal.
   - Note that you have to mention all the C files which include parts of your code (called dependencies) in front of the gcc command. 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. Again, change the settings of Geany (see 1.3.3) and enter the following command in the execute section:
     ```
     ./sensors_main messages.txt
     ```
   Note that with this line, you are giving messages.txt as an argument to your code. Run your program (see Section 1.3.6).
   - 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 following the steps described in Section 1.3.7 and enter sensors_main instead of clean. 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 recompile the program as before. Does it compile? What is the name of the output executable file?

6. (Q): Fill in the missing labels in Figure 1: Compilation Diagram. The vertical boxes correspond to commands the Makefile executes. What are they?
7. (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. Implement the function print_sensor_value in sensor.c line 34, in order to print the values of time and temperature. Modify line 39 of sensors_main.c to call your new function. Compile and test your program using your new Makefile (don’t forget to make clean (see Section 1.3.7) before compiling with make). Remember you always need to compile after making any change in your C code.

8. (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 argument, and checks whether it is above 35 degrees. If it is above the threshold, a warning should 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 three 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 c_vs_matlab.

3.1 Coding aspects

9. (I): Open MATLAB 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.

```matlab
% matlab_test.m : Compare two strings
text = 'this is some text';
if(text == 'this is some text')
a = 1;
else
    a = 2;
end
```

- Create a C code named `c_test.c` in Geany which does the same operation.

**Hint:**
- You take inspiration from your previous c files to create `c_test.c`
- To initialize a string variable do as follows:

```c
char text[20]="This is some text\0"; // Initialize a string variable
```

- To compare two strings, use `strcmp` as follows and include `<string.h>` on the top of your file:

```c
strcmp(text"This is some text\0"); // Compare two strings
```

- Compile using gcc and test your program, don’t forget to change the Geany settings (see Section 1.3.3).
- Is this implementation intuitive for you and did you manage to make it working directly? Did you need to compile the MATLAB script?

**Hint:**

3.2 Loops

10. (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, `modola(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.

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

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

12. (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, `modola(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 (see Section 1.3.3), run it. Ensure that you have the same result with both MATLAB and C. Note the time that it took to execute.

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

### 3.3 Matrix Multiplication

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

14. **(I):** Equation (2) describes the formula for the multiplication of two matrices, where \( M=250 \) 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 \( A*B \))*

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

16. **(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](http://www.cplusplus.com/reference/cstdlib/rand/?kw=rand)

17. **(Q):** Which program is faster? Why? Which program is more complex to understand?
4 Memory Management in C

The following two exercises will remind you first, the key concepts of memory allocation (static vs dynamic) and second, the basis of data organization and management using structure and pointers. The code needed for this section is in the folder *memory_management*.

4.1 Static and dynamic memory allocation

18. (S): The program below, when executed, prints “Your name is” followed by the name of the user taken as an input from the terminal. The source code is provided in the file *name_user.c*. Use gcc to compile your source file by set the command `gcc name_user.c -o name_user` in the compile section (see Section 1.3.3). Then, execute the program (see Section 1.3.6).

```c
#include <stdio.h>
void name_user(char* name) {
    printf("Your name is %s!\n", name);
}
int main(int argc, char* args[]) {
    char name[10];
    printf("Enter your name: ");
    scanf("%s", name);
    name_user(name);
    return 0;
}
```

19. (Q): Consider the case when the user’s name is longer than 10 characters. What will happen then?

20. (S): Why? Now, re-execute the program and input a name much longer than 10 characters.

21. (Q): What if the user’s name is exactly 10 characters long? *Hint: think about the termination character \0 of a string.*

22. (Q): Can you think of a few problems that may arise as a consequence of exceeding an array bound?

23. (B): Why is it possible that the program from S18 executes without an explicit warning when you enter a name equal to or slightly longer than 10 characters? In the general case, is it a safe practice to reserve a buffer which is a few characters shorter than needed for the input?

24. (I): You should always make an effort to make your program safe. By using a field of *scanf* function, you can limit the number of characters you will read from input to the size of your buffer. Consult the documentation (http://www.cplusplus.com/reference/cstdio/scanf/?kw=scanf) of *scanf* and update the line 12 to only read at most 9 characters.

25. (Q): You saw during the lecture that pointers and arrays are closely related. Compare the effect of the following two declarations on the memory and explain your reasoning.
char *p;
char a[20];

26. (I): Open `name_user2.c`. Now, modify `name_user2.c` so that the user is asked to enter the size of the buffer (i.e., the length of his name), and allocate the memory for this buffer (using `malloc`). Do not forget to free the memory at the end (using `free`).

*Hint:* You can read the malloc page from ENG-270 course ([https://coursc.ch/#malloc](https://coursc.ch/#malloc)). Don’t forget that a char needs “0” at the end, choose the memory size carefully.

### 4.2 Data Structures and pointers

27. (Q): In `account_structure.c` which you can find in the Code folder, a structure is defined to be used for account information. The code is provided below. Notice, that there is one structure instance (`a`) and one pointer to the structure (`ptr_a`). How would you access the value of the account balance? Choose all answers that apply.

```c
struct account {
    int account_number;
    double balance;
    char *first_name;
    char *last_name;
};
struct account a;
struct account* ptr_a = &a;
```

a) `ptr_a.balance`
b) `a.balance`
c) `ptr_a -> balance`
d) `a-> balance`
e) `(a).balance`
f) `*(ptr_a.balance)`
g) `(*ptr_a).balance`
h) `(a)->balance`

28. (I): In `account_structure.c`, complete the code.

- In the main function, assign some values (you can choose them) to each field of the struct
- Complete the function `print_account`, in order to print each field of the structure.

*Note:* We use a pointer as input of the function.

- Set the following command on the Geany settings window (see Section 1.3.3) `gcc account_structure.c -o account_structure`, then compiling and execute the program.
5 C Debugger and Additional Exercises

In this section, we want you to get familiar with the functionalities provided by gdb that allows for easier debugging. If you need to check the value of a variable, stop and check your variable if a certain condition is met, step-by-step go through your code to get a better understanding of what is going on or watch how a certain variable changes, there are already commands at your disposal (see Annex A). All you need to do is to learn how to use them and understand how to debug a program. The code needed for this section is in the folder debugger.

29. (Q) Open the file crash_test.c with Geany. Compile the program and run it. What do you obtain?

30. (Q) Compile the program using the Makefile with the debug option and run it (see Section 1.3.8). What do you obtain?

31. (Q) A terminal window with the debugger running should have started. Type the run command to launch the program with the debugger. What are the outputs and can you find where the problem is?

32. (S) Use the command backtrace to see where the program crashed. Quit the debugger by using the quit command. Re-launch the program from Geany and set a break point to the line 6 with the command break 6. Use the commands print number and print *number to check the current value of number. What are the outputs?

33. (I) Replace the line 6 by the following instruction. Compile and run your code. What is the result? Could you explain the bug?

```c
number = "?";
```

Note: It is bad practice to use a char to print a number.

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

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

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

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

36. (I): You can implement the previous codes and compare the results with your prediction. You can use the online compiler for ENG-270 course (https://course.ch/#bac-a-sable).

If you think you need more exercises, you can have a look to some of the of ENG-270 course (https://viereck.ch/c/simple-exercises/)
## Annex A : GDB main commands

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td></td>
<td>Starts the program.</td>
</tr>
<tr>
<td>run</td>
<td></td>
<td>Runs the program</td>
</tr>
<tr>
<td>break</td>
<td>Line number or function name</td>
<td>Sets a breakpoint in the given location</td>
</tr>
<tr>
<td>step</td>
<td></td>
<td>Single-step</td>
</tr>
<tr>
<td>next</td>
<td></td>
<td>Execute next line step over functions</td>
</tr>
<tr>
<td>print</td>
<td>expression</td>
<td>Displays the value of the variable</td>
</tr>
<tr>
<td>watch</td>
<td>expression</td>
<td>Sets a watchpoint for the expression</td>
</tr>
<tr>
<td>continue</td>
<td></td>
<td>Continue normal execution.</td>
</tr>
<tr>
<td>Backtrace</td>
<td>full</td>
<td>Shows the entire stack of the program</td>
</tr>
<tr>
<td>delete/enable/disable</td>
<td>Breakpoint or watchpoint</td>
<td>Delete/enable/disable</td>
</tr>
<tr>
<td>Info</td>
<td>args/arg/address/symbol</td>
<td>Show arguments of selected the frame</td>
</tr>
<tr>
<td></td>
<td>break</td>
<td>show defined breakpoints</td>
</tr>
<tr>
<td></td>
<td>watch</td>
<td>show defined watch points</td>
</tr>
</tbody>
</table>

**Note:** If you are interested in learning more about C debugger, you can check the documentation of gdb (http://sourceware.org/gdb/current/onlinedocs/gdb.pdf) and lldb (https://lldb.llvm.org/use/map.html);