Lab 1: Getting Started

ELEC 278: Fundamentals of Information Structures

Learning Goals for ELEC278 Labs:

- Implement and manipulate the data structures discussed in class and understand their properties and use cases.
- Explore how different data structures are used to solve different kinds of problems.
- Gain practice programming and debugging in C-language.

The goals for Lab 1 are:

- Install your development environment and run your first `Hello World!` in this class.
- Perform an array manipulation function to get you started on programming in C.

Schedule

ELEC278 will have 6 labs: one introductory lab for environment setup and the rest covering data structures that have been presented in class. Each lab session is 2 hours long with each lab section attending their session every second week.

The teaching assistants (TAs) will grade your lab in-person during the lab sessions. After completing the lab exercises, you will present your program to one of the lab TAs, and discuss the approach and correctness of your program with them. The TAs will evaluate your program in alignment with marking criteria that will be included in the lab assignment. Typically, the grading will assess your understanding of the program as shown by your code comments and your discussion with the TA (make sure your code is readable), and your program's correctness and success in handling corner cases. For the latter, generate a list of test cases to demonstrate to the TA your ability to anticipate and handle corner cases and write correct functional code.

The lab portion of ELEC278 counts toward 20% of the final grade, with your highest 5 out of 6 lab grades contributing 4% each towards the total. Lab 1 is introductory and is mainly to get you started developing in C.

Your TA will inform you of your grade and upload your grade to onQ in the lab. Make sure that you can see your grade on onQ before you have left the lab.

Lab assignments will be posted early on onQ so that you start working on the lab early. While recommended you finish by the end of your assigned lab session, if you need more time after your assigned lab session to complete the lab, you will have three more lab sessions to get your lab graded.

For example, the second lab (lab 2) for Section 009 will be on September 16 (Monday) 8:30-10:30AM. Students in Section 009 will be able to get their lab 2 graded during any of the lab sessions during the week of September 16 (Monday 8:30-10:30AM, Thursday 2:30-4:30PM, and Thursday 6:30-8:30PM) and September 23 Monday 8:30-10:30AM. The deadline to present lab 2 to the TAs for students in Section 009 is September 23 at 10:30AM.

In other words, pay attention that the deadline of your lab is day 7 of your scheduled lab day. No accommodation will be provided after this deadline. Please make sure that your results will be marked by the TA before your deadline and before the lab has ended.

Similarly, this applies to other lab section as follows, and extends in the same manner to other lab assignments.

- Section 009: September 16 (Monday) 8:30AM - 10:30AM; Deadline: September 23, 10:30AM.
- Section 005: September 19 (Thursday) 2:30PM - 4:30PM; Deadline: September 26, 4:30PM
- Section 006: September 19 (Thursday) 6:30PM - 8:30PM; Deadline: September 26, 8:30PM.
- Section 010: September 23 (Monday) 8:30AM - 10:30AM; Deadline: September 30, 10:30AM.
- Section 007: September 26 (Thursday) 2:30PM - 4:30PM; Deadline: October 3, 4:30PM
- Section 008: September 19 (Thursday) 6:30PM - 8:30PM; Deadline: October 3, 8:30PM.

Development Environment

You can use any C development environment that you are familiar and comfortable with.

We recommend VSCode or CLion (the latter, especially, from Mac users), if you do not already have an environment installed. Note that your TAs may not be able to help you with debugging around the specifics of your development environment if they themselves are not familiar with the environment; so, make sure to familiarize yourself with writing, debugging, and running code on your environment.

VSCode

VSCode is a light-weight source-code editor, which offers a wide range of plug-ins that enable seamless integration with various programming languages and is quickly becoming an industry standard. Getting familiar with VSCode in this course will ease getting started with other programming languages in following courses.

You can download VSCode by following this link: `https://code.visualstudio.com/` and downloading the version compatible with your device.

Next, follow the instructions in this link: `https://code.visualstudio.com/docs/languages/cpp` to get started with C-programming. The installation steps for C and C++ are the same in VSCode. But instead of creating `.cpp` files, you will create `.c` files.

This is a `Hello World!` program in C.

```
#include <stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

Follow the instruction to run the `Hello World!` program in VSCode.

CMake on VSCode

In addition, we will install CMake in VSCode to help us build and test our C-projects using configuration files (CMakeLists.txt) during the course.

In VSCode, install the CMake Tools extension by searching for 'CMake tools' in the Extensions view (Ctrl+Shift+X). You can follow steps 4 and then 3 under the Prerequisites in this page: `https://code.visualstudio.com/docs/cpp/cmake-linux`.

For step 4, download and install CMake from the `https://cmake.org/download/` by following the installation instructions for your operating system.

CLion

Sign up with your university email to get a free student license to use CLion `https://www.jetbrains.com/community/education/#students`.

Next, download CLion for your operating system here: `https://www.jetbrains.com/clion/download/`

You can familiarize yourself with the environment by reading through the Quick Start at `https://www.jetbrains.com/help/clion/clion-quick-start-guide.html#open-create-prj`

Testing CMake

- Unzip the Lab 1 folder and open your code editor inside the `helloWorld` project directory.
- Inside the project folder, you will find a file named `CMakeLists.txt` with the following content:

  ```
  project(HelloWorld C)
  cmake_minimum_required(VERSION 3.0)
  add_executable(hello_world main.c)
  ```

  This configuration file lists:

  - the name of the project (`HelloWorld`; or whatever you would like to name it; no spaces, and, optionally, the project language),
  - the minimum version of CMake to use, and

  - the name of the executable (`hello_world`) that will be created to run the source-code file (`main.c`)

- A file named `main.c` with your `Hello World!` program.
- Open the VSCode (or other environment's) terminal (you should be in your project directory). Create a `build` directory by typing:

  ```
  mkdir build
  ```

  Check that the `build` directory has been created. Type in the terminal

  ```
  ls
  ```

  and you should see:

  ```
  build  CMakeLists.txt  main.c
  ```

  Go into the `build` directory and call Cmake:

  ```
  cd build
  cmake ..
  ```

  Note that `..` is the path to the parent directory (Where CMake expects a CMakeLists.txt file).

  Call Make:

  ```
  make
  ```

  Alternatively, you can do this outside the terminal as follows:

  1. Open the VSCode Command Palette by pressing `Ctrl+Shift+P`.
  2. Type "CMake: Configure" and select it. This will configure your project.
  3. After configuration, open the Command Palette again and type "CMake: Build" and select it. This will build your project.
  4. Open the Terminal in VSCode by pressing `Ctrl+`` (backtick).
  5. Navigate to the build directory (usually `build`).

- Lastly, run the executable by typing `./hello_world` (or `hello_world.exe` on Windows).

  You should see the output:

  ```
  Hello World!
  ```

Grading (fraction of exercise marks)

0 No code.

1/2 Functioning code, but fails corner cases, if any, or not readable/not well-explained.

1 Functioning code that handles corner cases

Exercise 1. (2 marks) Open the exercises directory that you extracted (unzipped) in your code editor. You will complete the following exercises by completing the functions in the `main.c` file.

Complete the function `float calculate_mean(float arr[], int size)` that returns the mean (average) of the elements in the array. `size` is the size of the array.

Note that if you use `math.h` library and want to run or debug in your environment, you need to add `"-lm"` to the args of your tasks in your `tasks.json`. For example:

```
{
    "tasks": [
        {
            "type": "cppbuild",
            "label": "C/C++: gcc build active file",
            "command": "/usr/bin/gcc",
            "args": [
                "-fdiagnostics-color=always",
                "-g",
                "${file}",
                "-o",
                "${fileDirname}/${fileBasenameNoExtension}",
                "-lm"       // <========== Link math library
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            },
            "detail": "Task generated by me."
        }
    ],
    "version": "2.0.0"
}
```

Exercise 2. (2 marks) Complete the function `float calculate_standard_deviation(float arr[], int size)` that returns the standard deviation of the elements in the array.