Lab 0 – Setting up IDE [30 Marks]

For autograded submission, do not use `printf()` or `scanf()`.

Premise

The Fall 2024 version of COMPENG 2SH4 employs Visual Studio Code (VSCode) as the recommended C/C++ programming integrated development environment (IDE), and the GitHub Classroom as the lab and project submission platform.

The goal of this lab is to help you in setting up VSCode and all the required toolchain for completing the labs and the project for COMPENG 2SH4. This lab offers IDE and toolchain setup instructions for four different operating systems – Windows, MacOS, Ubuntu/Debian Linux, and Chrome OS. Depending on your choice of OS, you will need to read the corresponding sections to complete the IDE setup.

If you choose to adopt a different IDE, you will be responsible for resolving any IDE-related technical issues. The teaching team is only trained to support the VSCode-based toolchain.

Objective

This lab is worth 2% of the course mark.

Despite its simplicity in coding workload, Lab 0 is highly critical for your success in COMPENG 2SH4. The completion of this lab will guarantee you a workable VSCode IDE, C/C++ toolchain, and GitHub repository setup, so you can focus on learning coding in the in-class activities, labs, and course project. You will further benefit from it in COMPENG 2SI3.

The scope of this lab is the following:

- Setting up VSCode, GNU C/C++ toolchain, Memory Profiler, and GitHub account.
- Pull your first GitHub Classroom template repository for programming activities.
- An *extremely* simple one-line code implementation.
- Execute a simple unit test with the provided test bench.
- Push your first GitHub Classroom submission for autograding.

Follow the lab steps to complete the above items and take some time to get familiar with the development environment.

Helpful References

Read the provided documents under Lab0 folder in Avenue to Learn (A2L) and watch the GitHub training video to understand the operating principles of Git Revision Control and unit testing fundamentals. We may also use in-class activity times to work on some coding activities through GitHub classroom.

All COMPENG 2SH4 labs and project activities come with skeleton codes and their corresponding ***Makefile*** for compilation and execution. An example of how to use a ***Makefile*** will be provided in Lab 0 Activity section, as well as in the Lab 0 training video.

Create GitHub Account

Visit github.com and sign up **using your McMaster email address and set your Github username as your Mac ID**.
You need to do so even if you already have a personal GitHub account.

You **MUST use your McMaster email address account (Mac ID)** throughout COMPENG 2SH4 labs to ensure that the teaching team can correctly match your GitHub identity with your A2L identity. This is vital for receiving your lab grades on time on A2L.

If your ID is already taken on GitHub, you can *simply add a custom pre- or post-fix to your ID*. For example, if your ID is **chenw184** but already taken, you can proceed with **chenw184_fall2024** instead. As long as your Mac ID is part of the username, it is acceptable.

Setting Up IDE

This section provides detailed instructions on how to set up Visual Studio Code, GitHub plugin, GNU C/C++ compilers, and other required tools to support all 2SH4 activities.

The instructions cover the setup details for Windows, MacOS, Ubuntu/Debian Linux, and Chrome OS. Please make sure you follow the instructions pertaining to your specific OS.

WARNING for ChromeBook Users – Using ChromeBook for development purposes will require you to download an additional Linux emulator from Google. If your ChromeBook has less than 10GB of internal storage space, you will very likely encounter performance slow-down.

I. Preparing Your OS for Visual Studio Code

Windows
VSCode only works on Win 8/10/11. Make sure your OS is up-to-date before proceeding.

MacOS
VSCode only works on MacOS Monetery 10.11 and above. Make sure your OS is up-to-date before proceeding.

Ubuntu/Debian Linux
VSCode works on many Linux distros.
Call `sudo apt update` to make sure your OS is up-to-date before proceeding.

Chrome OS
VSCode only works on Chrome OS with Linux Emulator plugin.
Step 1) Go to *Setting*, search for keyword *Linux.*
Step 2) Find and enable *Linux Development Environment.*
Allow some time for download and installation.
Once Linux Emulator installation is completed, and the ChromeOS itself is up-to-date, you can proceed.
When lost, you may use this guide for more detailed instructions:
https://code.visualstudio.com/blogs/2020/12/03/chromebook-get-started

II. Obtaining VSCode

Visit the VSCode download page here: https://code.visualstudio.com/download

Windows

1) Download the Windows VSCode installer that matches your computer hardware.
2) Launch the installer, follow through the installation steps.

MacOS

1) Download the Universal (.zip) version. It works as a standalone application. No installation required.
2) Once downloaded, VSCode can be launched directly from the downloaded zip file. You may optionally keep VSCode on dock for convenience.

Ubuntu/Debian Linux

1) Download the Ubuntu/Debian installer that matches your computer hardware.
2) Launch the installer, follow through the installation steps.

Chrome OS

1) Download the Debian installer (.deb) that matches your computer hardware.
2) Launch the installer, follow through the installation steps.

III. Obtaining C/C++ Toolchain and Compiler

Possibly to your surprise, VSCode by itself is just a text editor with some enhanced GUI features. To make it a C/C++ integrated development environment (IDE), we need to download additional plug-ins to customize it to our liking. The very first plug-in is a system-specific C/C++ compiler and toolchain.

This part is VERY DIFFERENT among OSes. Make sure you are reading the correct section.

Windows – MinGW (**Min**imalistic **G**NU C/C++ Compiler for **W**indows)

1) Download MinGW from the URL below (or anywhere trustworthy) https://sourceforge.net/projects/mingw/
2) Launch the installer, leave all basic settings as default, **but make sure the following four boxes are checked**: a) mingw32-base, b) mingw32-gcc-g++, c) msys-base, and d) mingw-developer-toolkit. Also make sure that your MinGW installation path is C:\MinGW.
3) After selecting Mark for Installation for the above four boxes, close the MinGW Installation Manager window. An Action Requires Changes box will open where you should select Review Changes and then Apply.
4) Follow through the rest of the installation steps to complete the setup.
5) In addition, we will need to set up the environment variables on Windows so we can issue compilation commands for C/C++. Follow the following steps (with video having serial number 0)
6) Press Start, then search for "Edit the System Environment Variables".
7) In the *System Properties* window (Advanced tab), click "Environment Variables" at the bottom right corner to bring out the environment variables setup.
8) In System Variable panel (bottom half), double click on Path to bring out the path variable details.
9) Add the following two new paths "C:\MinGW\bin" and "C:\MinGW\msys\1.0\bin", (do not include the " "), then OK.
10) Close all the setting windows. Done.

MacOS – Homebrew + GNU

Homebrew is a lightweight command-line app installer that's very popular among software developer communities working on MacOS and iOS. Once installed, you can gain full control over customizing your development toolchain.

Note: If you prefer installing XCode IDE, you may do that as an alternative option.

1) Launch Terminal on MacOS
2) Copy and paste the following command in the Terminal.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

3) Press ENTER when prompted for confirmation. The remaining installation is automatic.
   It can take up to 30 minutes to complete the entire installation process. Be Patient!
4) Install GNU C/C++ Compiler Toolchain by typing in the following command under Terminal to install the latest version of GNU C/C++ compiler package.

```
brew install gcc
```

Linux and Chrome OS

After installing Linux Development Environment, your ChromeOS behaves more or less identical as Debian Linux.

1) Launch Linux Terminal
2) Update all the installed compiler toolchains to the latest version to prepare new compiler installation without version conflict. Type the following command in the terminal:

```
sudo apt update
```

3) After update is done, type in the following command to install the latest versions of GNU compilers (C/C++ compiler tools included)

```
sudo apt install build-essential
```

IV. Obtaining NCurses Library for C/C++ (Mac/Linux Users Only)

This library is required for the 2SH4 course project and project preparation activities.
You may omit this section if you are using Windows.

MacOS

Type the following command on MacOS Terminal (assuming you've installed Homebrew).

```
brew install ncurses
```

Note: If you have installed XCode, you don't have to explicitly install NCurses library. It's part of the package.

Linux and Chrome OS

Type the following command on Linux Terminal.

```
sudo apt-get install libncurses5-dev libncursesw5-dev
```

V. Installing Memory Analysis Toolchains

As we enter the C/C++ topics of memory management, it is impossible to analyze how well the program manages the memory usage without a profiling tool. Follow the steps below to set up a handy open-source tool "**Dr. Memory**" on Windows, and "**Valgrind**" on Linux, to help you profile the memory management aspects of your program. MacOS has its native memory leakage analysis tool "**leaks**" coming with XCode, so no explicit installation is required on MacOS.

You are expected to use this tool to ensure there is no memory leakage before submitting your work to GitHub classroom. The teaching team will evaluate your program using the same tool.

Windows – Install Dr. Memory

1) Download Dr. Memory local standalone version below.
   https://github.com/DynamoRIO/drmemory/releases/download/release_2.6.0/DrMemory-Windows-2.6.0.zip
2) Unzip the downloaded package under your preferred location, then navigate under the unzipped folder to locate the *bin* folder. Go into the *bin* folder, click on navigation bar to reveal the absolute path of the *bin* folder. Copy the path.
3) Click on Start -> Settings, then search for "*Edit the System Environment Variables"*. In the *System Properties* window (Advanced tab), click "Environment Variables" at the bottom right corner.
4) Under Environment Variables window, select "Path" under System Variables, then click Edit.
5) Press New, then paste the path you've copied in Step 2 above.
6) Press OK for all setting windows. Done.

MacOS – No action required; *leaks* already comes with XCode

Note: If you don't have XCode installed, just install it through Apple Store.

Linux / ChromeOS – Install Valgrind

1) Under Terminal, enter `sudo apt-get install valgrind`.
   Enter Y to confirm, then allow installation to complete. Done.

VI. Configure VSCode to work with GNU C/C++ Compiler Toolchain (with Video having serial number 1)

1) Launch VSCode.
2) Open *Extensions* page by clicking ***File -> Preferences -> Extensions***.
3) Search for the following two extensions and install them.
   a. **C/C++**
   b. **C/C++ Extension Pack**
4) Open *Settings* page by clicking ***File -> Preferences -> Settings***.
5) Search for **cpp Standard**. The result will show a number of configurable parameters. Change the following:

a. CPP Standard	gnu++11
b. C Standard	gnu89

6) (optional, use it when VSCode is stuck on updating C++ Extension pack) Search for "update mode", then change Update: Mode option to none. This will disable the auto update.
7) Then, under ***File -> Preferences -> Settings***, search for **Compiler Path**, then open the ***json*** setting file.

   Find "`C_Cpp.default.compilerPath`" property, and add "`${default}`" to it.

   Note: If VSCode complains about not finding the default compiler when compiling in step VII (next section), come back here and update the compiler path according to your OS setup.

Windows	Add "`C:\\MinGW\\bin`" to it.	**The double-slash is required!**
MacOS	Add "`/usr/bin/clang`" to it.	
Linux / Chrome OS	Add "`/usr/bin/gcc`" for C Update it to "`/usr/bin/g++`" for C++ in later part of the course.	

8) Once done, return to the *Explorer* view by clicking ***View -> Explorer***.

We are now ready to program in C/C++ using VSCode.

VII. Getting Started with VSCode (with Video having serial number 2)

While this is not the most complete starter guide, it is enough to get you started on coding and learning C/C++. All the in-class activities assume you have completed the activity in this guide.

1) Launch VSCode.
2) Upon startup, click on *Explorer* icon (top left corner), then click on *Open Folder*.
3) Create a new folder (maybe name it COE2SH4) somewhere easily accessible on your computer, then click *Open*. You now have a root folder in which you can manage all the lab / project / in-class activities.

   Your VSCode now considers this root folder the ***Workspace*** of your programming project. All the development and debugging tasks will default its target to the program in this folder. More discussions to come in class, as we will regularly change workspaces to carry out more advanced programming tasks.

   Windows
   As a good practice, create this root folder in a drive other than your system drive whenever possible.
   For example, instead of creating the folder under C drive as C:/COE2SH4, create it under D drive as D:/COE2SH4.

   Mac / Linux / ChromeOS
   As a good practice, create this root folder somewhere under your user folder. Do not create this folder at the system root. For example, instead of creating the folder as /COE2SH4, create it as
   /Users/[your account]/Documents/COE2SH4.

4) Under your newly created workspace folder, click on *Create New Folder*. Rename it as "2SH4_Hello_World". This subfolder will hold your first C project.

5) Navigate into 2SH4_Hello_World folder, click *Add a New File* icon, and name it "hello.c". The text editor will immediately show the contents of hello.c, which is now empty.

6) Type the following code into the hello.c text content

```
#include <stdio.h>
// This is the C's standard IO library.  Required to put some text on screen.

int main(void)
{
      printf("Hello World!");  // print something on the screen.
}
```

7) Save the file.
8) Open ***Terminal -> New Terminal***. The terminal window will show up at the bottom of the screen.
9) In the Terminal, you are located under your workspace folder.
Type the following to "Change your Directory" (**`cd`**) to 2SH4_Hello_World:

`cd 2SH4_Hello_World`

Tip: if you don't want to type out the full name, just type **`cd 2SH4`**, then press **`TAB`**. The terminal will be smart enough to autocomplete the rest of the name *as long as* there is only one subfolder with 2SH4 prefix.

10) Under this folder, type in **`gcc hello.c -o hello`**
This is the most basic compilation command that compiles hello.c code into hello.o executable.
    a. This command means "Use GNU C Compiler, compile the C program 'hello.c', and output it to an executable app called 'hello' "
    b. If you see an error message reported by clang on MacOS, you need to press the play button on the top right corner, select GNU compiler to compile the program once.
    c. Then, you should be able to use the **`gcc`** command in the terminal moving forward.

11) Once compilation is done, just for your peace of mind, use the "List Files" command (**`ls`**) to check whether you now have a *hello* executable app under the same subfolder.

12) Then, to run the executable, type **`./hello`**. You should now see a message "**Hello World!**"
`./[appname]` tells the terminal to run the app with the name appname under the current folder.
We will investigate into file paths in the future.

13) Finally, just to make sure the memory profiler is in place, type the following command:

Windows	**`drmemory ./hello.exe`**
MacOS	**`leaks --atExit -- ./hello`**
Linux/Chrome	**`valgrind --leak-check=yes ./hello`**

A wordy memory report will be generated. If a "Command Not Found" error appears, repeat **section V**.
Don't worry about the contents in the report for now. We will talk about it in later course activities.

The activity done in this section is not for grading, but for getting ready to participate in the in-class programming activities and for getting started with the labs.

Lab 0 Activity and GitHub Classroom [30 Marks]

We are finally ready for the graded activities in Lab 0. All 2SH4 labs and project activities are managed through GitHub classroom. As a result, you should get familiar with the workflow introduced in this part of the lab.

I. Accepting Lab 0 Assignment through GitHub Invitation

1) Sign in to GitHub.com using the previously generated account (**MUST BE McMaster EMAIL!!**).
2) Click on the following link to accept the Lab 0 assignment invitation: https://classroom.github.com/a/-IL54EW2
   By accepting the assignment, your private Lab 0 repository will be created under your GitHub account. The repo should have a name resembling: *https://github.com/COE2SH4-F2024/lab0-**[username]***, where ***[username]*** will match (or contain) your McMaster Mac ID.

   WARNING! If ***[username]*** is incorrect, you need to log out and sign up a new GitHub account using your McMaster email and Mac ID (as username) before coming back to this step.

   For Mac Users You may need to use a different internet browser if you encounter an error while accepting the invitation with Safari.

II. Connect VSCode to GitHub Classroom and Pull Lab 0 Starter Code (with Video having serial number 3)

Windows and MacOS
Download and install GitHub Desktop here https://desktop.github.com/download/

Linux / ChromeOS
Linux versions of VSCode requires additional Git plug-in support.
Open Terminal and use command

```
apt install git
```

1) In VSCode, open Terminal, make sure you are at the root folder you've created in the Getting Started section. If you are in the 2SH4_Hello_World folder and want to go back, type **`cd..`** to navigate back to root folder.

2) Under your root folder, type **`mkdir Lab0`**. This will create a new folder called Lab0 under your root folder. You can use the command **`ls`** to confirm the folder has been created.

3) Go into Lab 0 folder with the command **`cd Lab0`**.

4) Under the Lab 0 folder, type the following command to pull the Lab 0 starter code from 2SH4 GitHub classroom:

```
git clone https://github.com/COE2SH4-F2024/lab0-[username].git
```

   [username] MUST BE (OR CONTAIN) YOUR MCMASTER MAC ID!!

5) Step 4 may prompt you to sign in to GitHub again through your internet browser. Follow the instructions to complete the sign-in process.

   If the terminal prompts you for your GitHub credentials, you will need to create a **Personal Access Token (PAT)**. In this case, follow the Setting Up Personal Access Token section towards the end of the lab manual.

6) A sub-folder called lab0-[username] will be created under your Lab0 folder after completing the above steps. You can see it in the explorer window on the left-hand side or use the respective commands under the terminal – **`cd lab0-[username]`**, then **`ls`**.

This folder is your Project Folder, under which you should find the following files:

Test Suite Files	***testCases.c***	(Edit testCases.c to create more test cases)
Lab Header Files	***Question1.h***	
Lab Source Files	***Question1.c***	(Edit this to complete your lab activities)
Auto-Compilation Files	***makefile***	

Once confirmed these files are properly imported, you are ready to start developing the lab.

WARNING! **You should only modify *Question1.c* and *testCases.c*. DO NOT touch other files. For *testCases.c*, you are only allowed to add more test cases, but DO NOT modify the already existing test cases. When the teaching team grades the labs, the environment checks for any changes on these files. Upon violating these rules, you will get zero for violating academic integrity.**

III. Developing the Lab

1) First, go through ***Question1.h***, ***Question1.c***, and ***testCases.c*** and make sure you understand them.

2) Locate the `main()` function, and try to understand how it launches the test cases to test your work (to be completed)

3) Complete the implementation of the function `int addFunction(int input1, int input2)` in ***Question1.c*** file. This function is intended to take two input integer parameters, adds them, and returns the sum. Complete the missing implementation in this function.

4) Add three more test cases to **void TestQ1_Custom1()**, **void TestQ1_Custom2()**, and **void TestQ1_Custom3()**. Try to come up with interesting test cases (such as adding negative integers). If you forget to add these test cases, your autograding result will indicate a failure with reminders. You may remove the reminder printf() message once you develop the test case.

   NOTE: In your final Lab 0 submission, you are expected to submit these 3 additional test cases.

5) Once done, we can compile the project by using the GNU C/C++ auto-make command. Before doing so, make sure that you have saved your work.

6) In the terminal, make sure you are under the project sub-folder.

7) Type the command `make`.
   This command will invoke the compiler to use ***makefile*** to automatically build your code with the test suite, and compile an executable.

Windows	This executable will have a name `Lab0.exe`
Mac/Linux/Chrome	This executable will have a name `Lab0`

   Once your program compiles without syntactic errors, you can proceed to running the test suite.

   During debugging, if you want to have a fresh rebuild of the project, use command `make clean`, followed by `make`.

8) To run the project, type the command under the terminal:

<u>**Windows**</u>	`./Lab0.exe`
<u>**Mac/Linux/Chrome**</u>	`./Lab0`

`./[progName]` tells the terminal to execute the program with the name progName.

9) Observe the output in the Terminal. The unit testing framework will provide details to you whether your program passes all the tests or not. For the failed ones, it displays the expected result vs. your actual output, so you can have an idea of the behavioural error of your algorithm.

   Make sure you pass all the provided tests (ours and your additional tests) to make sure your work is ready for GitHub autograder.

   On our end, we have some undisclosed additional test cases to further test out the correctness of your submission. As a result, you should try to come up with carefully chosen additional test cases to cover all possible input / output scenarios for your algorithms to ensure its robustness.

IV. Work Submission - Pushing Changes to GitHub Repo (with Video having serial number 4)

After passing all the test cases locally, you are ready to submit your work to GitHub classroom for autograding.

Remember, Git is a two-level repository system. Your work is stored in 3 locations

Local Folder	Your local project sub-folder, visible through VSCode explorer
Local Repository	An invisible local Git repository on your computer for version control
Remote Repository	The remote Git repository on the cloud, accessible via GitHub.com webpage

The following instructions will refer to these three locations with the aforementioned names.

1) Open Terminal, navigate to your Local Folder (lab0-[username]) using `cd` command

2) Type `git status`
   This should provide a report of what files you've changed in your Local Folder.

3) Type `make clean`, follow by `git add *`
   This will remove the compiled binary files and add only the required source codes to the Local Repository submission list (formally known as the "commit list")

4) Type `git commit -m "message"`
   This will submit your latest work from your Local Folder to your Local Repository.
   "message" contents should be modified with a meaningful message of your choice.

   WARNING! Commit does not submit your work for GitHub online autograding. It only ensures that your work in the Local Folder now has a backup checkpoint on the Local Repository.

5) Type `git push`

   This will push the latest contents of the Local Repository to the Remote Repository.
   Your lab work is properly submitted for autograding only after completing the Git Push command.

6) Check on GitHub.com webpage. You can see all your commits and their autograding status on your GitHub repository page on GitHub.com.

   Note: Only the commit with the latest timestamp is used for grading.

Tips: Good Version Control Practice

This lab may not be very involved, and you can finish it in one setting. However, in the remaining labs and project activities (and surely, in large-scale real projects), development is done over a period of time with multiple collaborators. It is very important that you keep your code consistent. To do so, follow these easy steps:

1) Each time you start working in the project / lab, do a **`git pull`** command. This will update BOTH your local repository and local folder with any changes conducted on the remote repository, which can be shared among many collaborators.

2) Each time before closing your development session (e.g., closing VSCode), make sure you **`git commit`** + **`git push`** all your contributions from the local folder to BOTH the local and the remote repositories.

   In fact, even within the same session, if you make some big changes (e.g. finishing one function / question), you should also commit and push.

Tips: Setting Up Personal Access Token (Read as Needed)

1) Log into GitHub from your web browser.
2) Click on your user icon (top right corner), go into ***Settings.***
3) On the left panel, go into ***Developer settings.***
4) Go into ***Personal Access Token***, then click on ***Tokens (classic).***
5) On the right-hand side of the page, click on ***Generate New Token*** -> ***Generate New Token (classic)*** **<u>do not use Beta</u>**; we will not support any gotchas in there!!
6) Set the token expiration duration – your choice. 90-day period recommended.
7) Set the scope of the token appropriately – Select all under **repo** access, and can leave everything else unchecked.
8) Click on ***Generate Token***. A new Personal Access Token (PAT) will be generated.
   IMPORTANT! You need to record it somewhere so you can remember it (and copy-and-paste it whenever needed).
9) Go into VSCode, under Terminal, use the following command:

   ```
   git clone https://[PAT]:[PAT]@github.com/COE2SH4-F2024/lab0-[username].git
   ```

   (Replace **`[PAT]`** with your recorded token from step 8)

   This should give you successful access to the repo.

You can find the complete GitHub PAT Guide here:
https://docs.github.com/en/enterprise-server@3.6/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens#creating-a-personal-access-token

Lab Deadline

The deadline for Lab 0 is Friday, September 13, 2024, at 11:59 pm.

Please note that this is a programmed deadline on GitHub classroom.
You will lose access to your GitHub repository beyond the deadline.

As a result, make sure you absolutely respect the deadline and submit your work on time. We will not grant any extensions.

Lab Marking Scheme

- Code Implementation
  - **[15 marks]** Completing the required implementations in Question1.c and passing the default 2 test cases
- Code Behaviour, Analysis, and Test Plans
  - **[15 marks, 5 marks each]** Creating 3 additional valid test cases in testCases.c

IMPORTANT: **Your submission must compile without syntactic error to receive grades. Non-compilable solutions will not be graded.**