

# Getting Started with the CodeLite Environment

## APS 105F - Computer Fundamentals

Winter 2018

---

### I. INTRODUCTION

In APS105 you will use the **CodeLite** software to help you create, compile and find errors in the programs you will write. CodeLite is the kind of software that is called an *Integrated Development Environment*, or IDE. An IDE is a programming environment which programmers typically use as the tool to write code in plain text, to compile and build it into what the computer can execute, to debug the program if necessary (it will be!), and finally to execute (or run) the program. This document tells you how to begin using CodeLite to make a short C-language program.

### II. USING CODELITE ON YOUR OWN COMPUTER

CodeLite is available on the computers in the Engineering Computing Facility (ECF) laboratories that you'll be using. You can do all of your preparation work (prior to the lab period) on those computers when the labs are open.

If, however, you have your own computer, we strongly suggest that you also install and use CodeLite on it because it is convenient. There are two ways of doing it. Note, though, that you *should not* install anything on the laboratory computers. The alternatives for installation given below are for your own computer.

First, you can download the CodeLite IDE directly from the website and install it. It can be found at <https://downloads.codelite.org/>. You will have to choose the appropriate installation according to the operating system and architecture (that is, 32-bit or 64-bit) of your machine. This may be a good time for you to find out more about the hardware you have, so check out your system settings. The installation is very straight forward: download the installer and run it.

A second option is to create within your computer an environment that is just like the one you will be using in the lab computers. This is done via what is called a "Virtual Machine" (VM). To get that going on your computer, see the document **Installing & Using the APS105 Virtual Machine** for more information about this. This process is more involved and you will have to follow a number of steps, which at the end will include one to connect directly with your ECF folder.

### III. LEARNING CODELITE BASICS

We will now walk through the creation of a simple C-language program using CodeLite. Follow each of the instructions below in turn:

- 1) Start the CodeLite program (on Linux or ECF you can type **codelite &** onto a terminal window; on Windows and a Mac, just double-click the CodeLite icon.) Once it is running, you should see a screen that looks like Fig. 1, below.

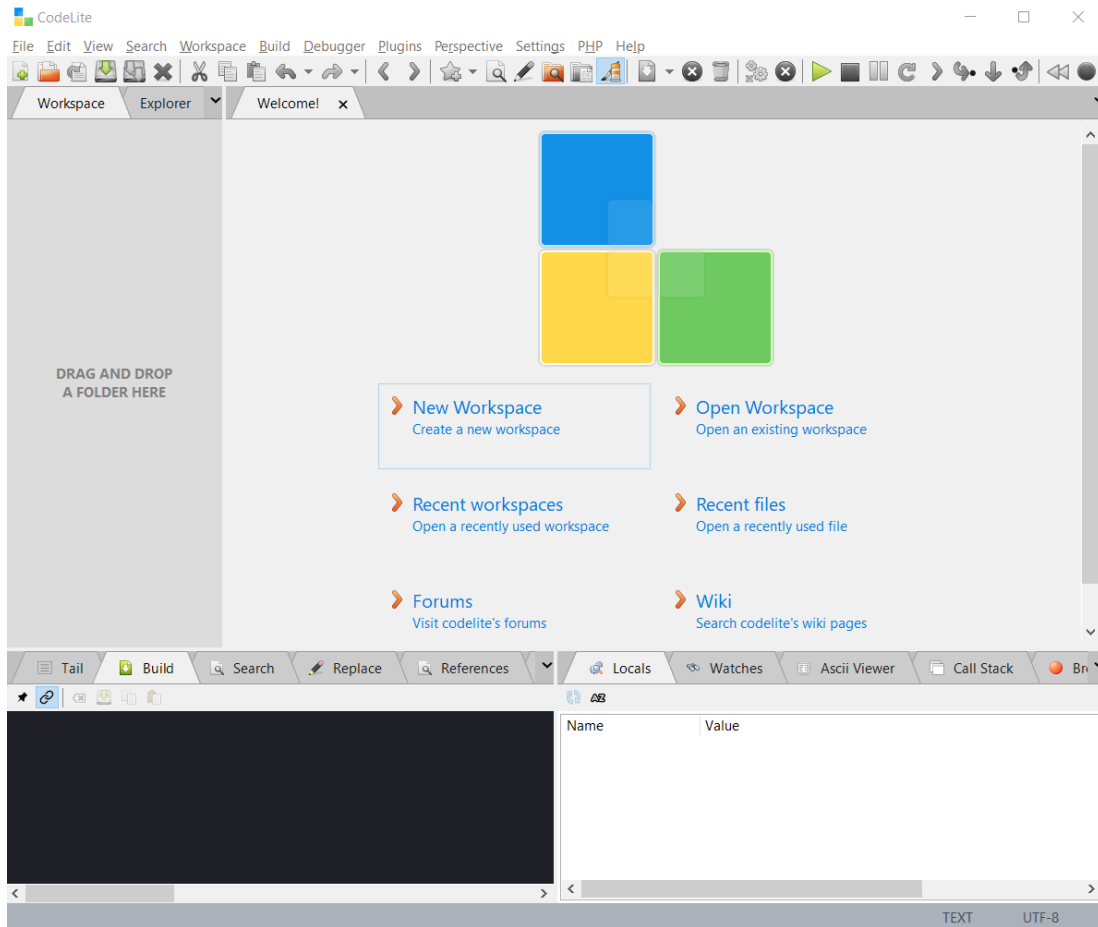


Fig. 1: The Opening Screen You'll see when you first run CodeLite

- 2) All programs that you will make in this course will become what are called *projects* in CodeLite. First, however, you must create a *Workspace* in which your projects will be. Click on *New Workspace*. This should open the small window like Fig. 2, below

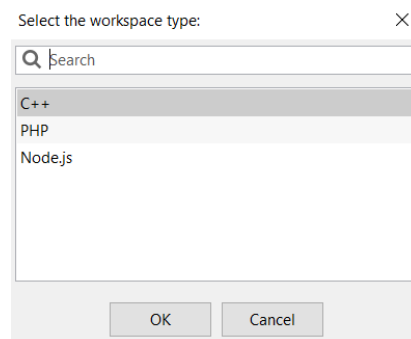


Fig. 2: Creating a New Workspace

- 3) You will select C++ and click OK. After clicking OK you will see a window like the one presented below. Name your workspace and select a directory for it.

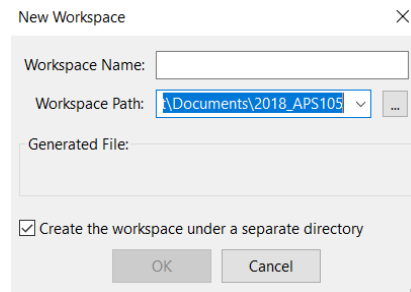


Fig. 3: Naming the New Workspace

- 4) After this is said and done, you will create *Projects* within the workspace. These projects is where the program you will write in C will be located. Later in the course you will learn that all types of files necessary for the full compilation of your code will be found in the project as well. Let us say at this point that you have created a new workspace aptly named `MyNewWorkspace`. The top left corner of your IDE will have this name showing as the location (or root) of all your projects. Now click on `File -- New -- New Project`. A new window will appear, like the one below.

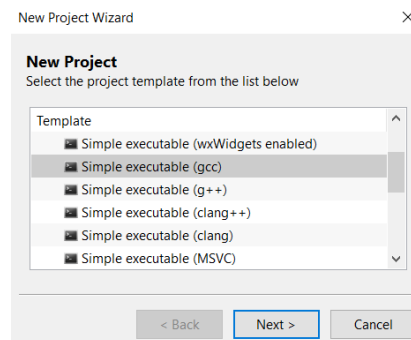


Fig. 4: Creating a New Project

- 5) This is where you tell CodeLite you are writnig a program in the C language. Choose `gcc` (that stands for GNU Compiler Collection), press Next. On the new window that will appear, default compiler shown has been installed along with CodeLite, so you do not need to touch these options unless you want to use another compiler of your choice.

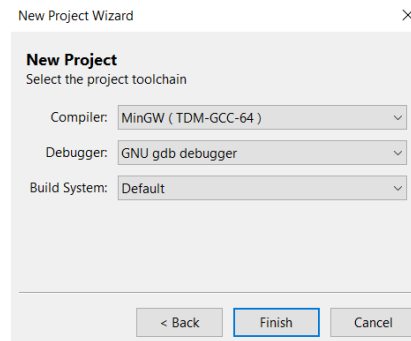


Fig. 5: Selecting a Compiler for the New Project

- 6) When you click `Finish`, you will notice that at the top left corner of CodeLite's main window a new project has been created with the name you chose, and this project is nested under the workspace that you created as well. If you now click on the `>` on the left side of the project's name, the tree will expand and show you the `src` (stands for "source") folder. In this folder you will find – surprise! – the `main.c` file already put there for you.
- 7) CodeLite goes one step beyond, and in addition to giving you all this, it creates a `main.c` file with the few lines of code needed for the famous "Hello World" program. If you double-click on `main.c`, you will find the main window looking like the figure below.

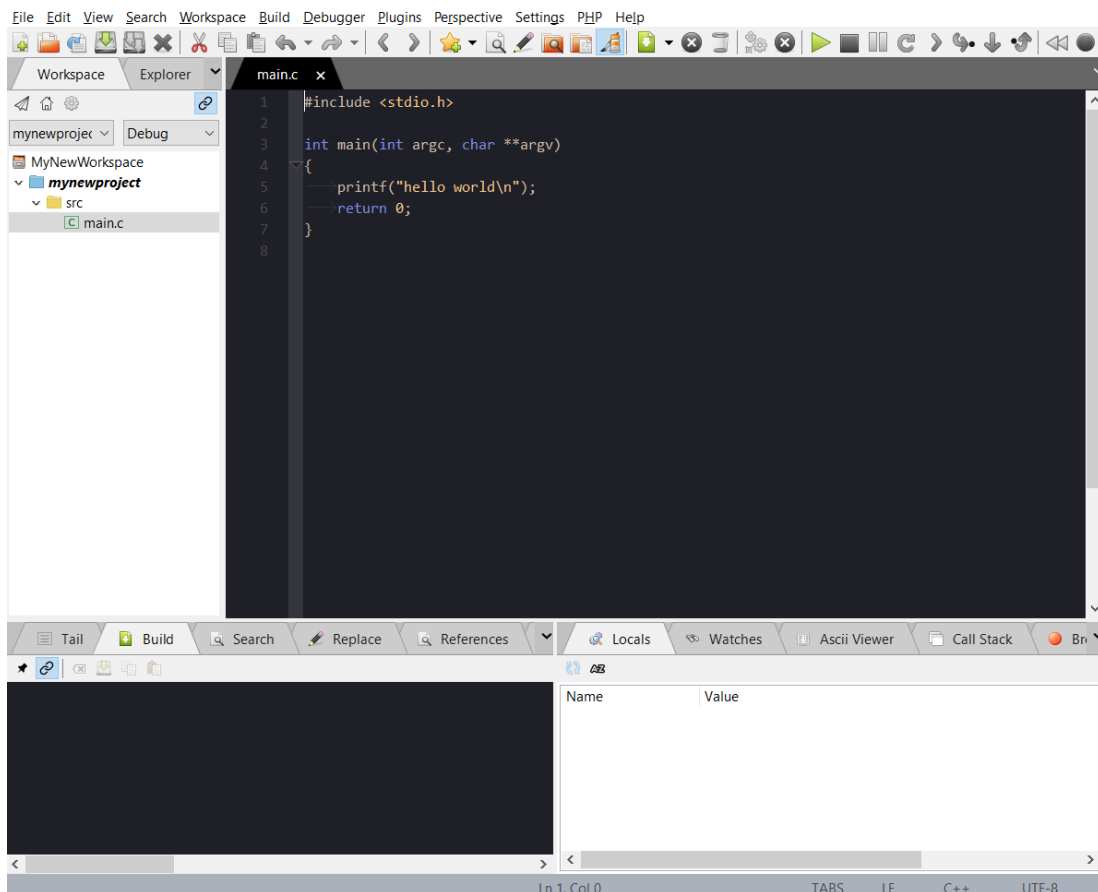


Fig. 6: Workspace, Project and Program

- 8) You can compile and build your project into an executable program by going to the Build -- Build Project option from the top menu, or simply pressing F7. If you want to build *and* run the executable program created, you can go under Build -- Build and Run Project or press ctrl-F9 (for some Windows machines, this means "at the same time, press the control key, the function key and the F9 key). When the program runs, a new window will appear, and the output of your program will be there. For this case, we have:

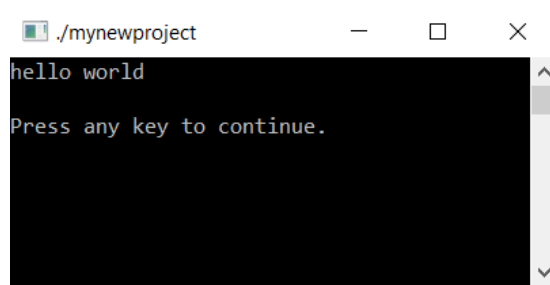


Fig. 7: Workspace, Project and Program

- 9) If you want a faster way to compile and build the program, the "play" button (a green, right-pointing triangle at the top menu) will cause the program to be compiled, built into an executable program ready to be debugged if you so desire. Note, however, that once you have your project *built*, and there are no errors to be resolved, you need to *execute it*. That means, you have to run it after you sort out the errors pointed out by the compiler.
- 10) Now that we have it, let's break it. (don't quote this particular statement...) You have compiled and run your first C-language program, so now you will try to modify it and observe the result. First, remove the semicolon at the end of the print statement. Try to build the program. What happens?
- 11) Now fix what you broke, and put in another printf statement that causes the additional output "The rain in Spain lies mainly in the plain." If you forget to put the `\n` at the end of the line, you'll notice that there is a missing new line space after the text on the output. Try this now. Remember, after editing, compile/build, and run (if no errors are found by the compiler).
- 12) One last, important item. You need to find where the **main.c** file actually is. You will need to remember the path you put in when you defined the location of your Workspace, and your Project will be within that folder. The **main.c** file for the chosen project will be within the project's folder. Try now to find the file (program) you have just executed.
- 13) Congratulations, you're on your way to becoming a C programmer!

#### IV. TESTING YOUR CODELITE PROJECT THROUGH THE LINUX TERMINAL

We will now walk through the process of testing the code that you created with Codelite. If you have not already read the part of Lab 0 that is titled **Getting Started with Linux and ECF**, please do so now, as knowledge of the Linux terminal is a prerequisite for this section.

For every lab assignment in this course, you will need to use the Linux terminal to run command line programs (or “scripts”) that perform a variety of functions. There are four scripts, with the following names and functions:

- **exercise**: runs your C program and tests it with a few test cases to find errors in the output. You will use this program to check if the output format of your program is compatible with the program that will be used later to check your answers. It is important that this program agree that your output is correct, and that you fix it if not. However, this program does not check all of your program for correctness, just the format
- **submit**: this is what you run to hand in your weekly lab assignments; you must run this program prior to your submission deadline.
- **viewsubmitted**: checks that your submitted assignment was actually received. You should always run this to make sure the submit process was successful, don’t that assume that it was.
- **marker**: displays the grade that the auto-marker gave you, showing you why you lost marks (if at all). This program can only be run after the deadline for submission has passed, and marking has occurred

This section will show you how to use the **exercise** script. You will be taught how to use the remaining scripts in the upcoming labs. In order to use the exerciser, we will first create a Lab 0 C program.

##### A. The Lab 0 Program

Let’s create a simple C program just like before, that prints “This is my lab 0 program!”. We will then use the exercise script to check if the C program operates as we expect.

- 1) Create a new CodeLite project, which was the process you learned above in this document. This time, name the project “Lab0”, and change the default name of the C file from “main” to “Lab0” – do this on the screen shown in Figure 6 earlier in this document. Right-click on the file name `main.c` and select `Rename`.
- 2) Change the print statement to:

```
printf("This is my Lab 0 program\n");
```

- 3) Save your changes and compile your program, making sure that you have no syntax errors.

## B. Testing Lab 0 with the exerciser

We now want to ensure that our program is correct - that is, that we print out "This is my Lab 0 program!". We will use the Lab 0 exerciser to do this.

- 1) Open a Terminal window, which (hopefully you'll get this memorized!) is done by right clicking anywhere on the (Linux) desktop and selecting "open in terminal."
- 2) To run the exercise script on this lab, you need to first change the working directory to be where your CodeLite project directory is. The name of the project directory is the same as the name of the project (i.e., "Lab0" in this case). The project folder is within the workspace folder.

So, in order to change directories to our Lab 0 project's directory, use the following command:

```
cd ~/..... path to directory Lab0
```

You should now use the "ls" command to make sure that one of the files in that directory is called **Lab0.c**. If it is not there, then you haven't done the change directory command properly, or you didn't set the name of file correctly in step 1) above.

- 3) Now that our current working directory is correct, we can run the exercise script. Every lab will have a dedicated exercise script, located at the following path:

```
/share/copy/aps105s/labX/exercise
```

where **X** is the number of the lab we are working on.

So, now you should run the exerciser for our Lab0 project by invoking the exercise script as follows (enter this command into the Terminal):

```
/share/copy/aps105s/lab0/exercise
```

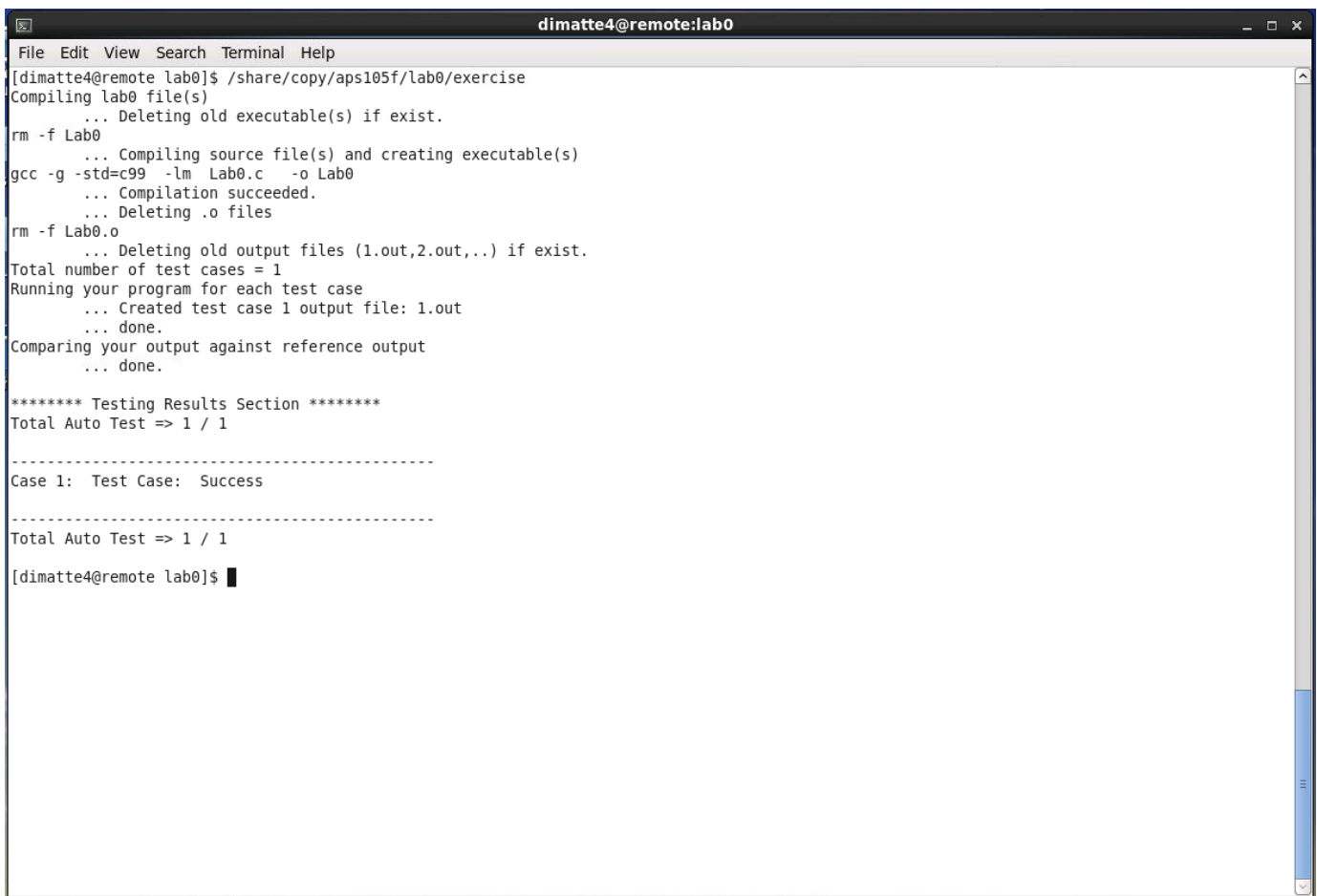
- 4) The exercise script will generate quite a bit of output. This begins with cleaning up the directory, and then it runs the commands that compile your program to make it executable. Then it executes your program one or more times, with different inputs (that the script supplies) to see if the outputs are correct. Each time it runs the program is called a "Case" - Case 1, Case 2 ...etc.

For this example program, there is just one Case, and if you typed in exactly what was asked above, it should report an error: The exercise program expected your program to output "This is my lab 0 program!", but your program did not, because there is a missing exclamation mark at the end of the text you created. (If you actually did put the exclamation mark in your program, please take it out so that you can see what an error looks like). Since the output of your program does not match the output of the reference solution, the exercise program "fails" this single test case.

Read carefully the output of the exercise program. It is very important that you learn to

read the output from the exercise program, as you must correct the errors that it detects, to make sure your output format is correct. Your mark will depend on the outputs being correct. If you have an error from the exercise program, you must fix it!

- 5) Go back to CodeLite and modify the print statement to contain an exclamation mark after the work "Program". Save your changes.
- 6) Re-run the exerciser and confirm that the test case now passes. That is, there are no errors. Below, on Figure 8 there is an example of what the message will look like if there are no errors.



```
dimatte4@remote:lab0
File Edit View Search Terminal Help
[dimatte4@remote lab0]$ /share/copy/aps105f/lab0/exercise
Compiling lab0 file(s)
... Deleting old executable(s) if exist.
rm -f Lab0
... Compiling source file(s) and creating executable(s)
gcc -g -std=c99 -lm Lab0.c -o Lab0
... Compilation succeeded.
... Deleting .o files
rm -f Lab0.o
... Deleting old output files (1.out,2.out,..) if exist.
Total number of test cases = 1
Running your program for each test case
... Created test case 1 output file: 1.out
... done.
Comparing your output against reference output
... done.

***** Testing Results Section *****
Total Auto Test => 1 / 1

-----
Case 1: Test Case: Success
-----
Total Auto Test => 1 / 1

[dimatte4@remote lab0]$
```

Fig. 8: Successful output from the exercise command.

- 7) Congratulations, you built and tested your first C program!