Lab #0
CodeLite, MinGW and Git

Purpose:

The purpose of this lab is to introduce each student to the tools that will be used throughout this course.

All the tools are online and freely downloadable or run online. Set them up on your personal computer and ensure they run ok.

You may work with other students on this lab, but everyone must complete all the steps outlined below

Preparation:

Very little preparation is required. You may just go to the lab and carry out the tasks enumerated. However, you will find your understanding enhanced if you allow yourself a little time to experiment with the various commands and procedures. If you get some ideas beforehand of what you might like to try, it could be an especially productive session.

Here is the list of the tools we'll be using in this course:

CodeLite: https://codelite.org/

MinGW: https://sourceforge.net/projects/mingw-w64/

Git tools: https://git-scm.com/downloads

GitHub Desktop: https://desktop.github.com/

Procedure:

Git and GitHub Tutorial

This course will use Git and GitHub for all lab work and lab source code submissions. The starting files for each lab will be contained in a GitHub repository that you will fork to your own personal repository. You will populate your repository with your source code work and leverage GitHub to be the safe place to keep your latest work

There are lots of sources of help for Git. For this section of the lab, you will need to do the following:

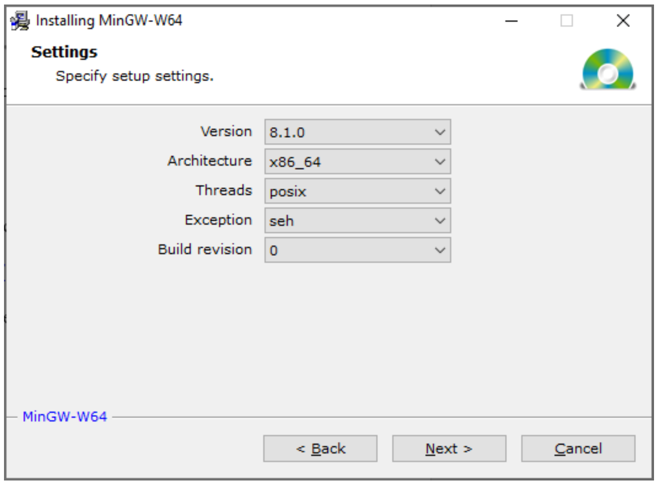
1) Create a public GitHub account using your SU email (it's free!)
2) Go to https://try.github.io/ and complete the online tutorial
    a. Skip the part about creating a Code School account (not needed)

    b. Complete this tutorial online. It gives you a sense of the command line version of git
3) Go to https://guides.github.com/ and go through the following tutorials
    a. Understanding the GitHub flow
    b. Hello World
    c. Forking Projects
4) Go to https://docs.github.com/en/desktop and go through the following tutorials
    a. Installing and Configuring with GitHub Desktop
    b. Contributing and Collaborating with GitHub Desktop

Install MinGW C/C++ Compiler Tools

MinGW is an environment for using the GCC compiler tools on Windows

1) Go to https://sourceforge.net/projects/mingw-w64/ then click on the Files tab, then click to download the MinGW-W64 Online Installer
2) Run the mingw-w64-install.exe. Change the architecture to x86_64

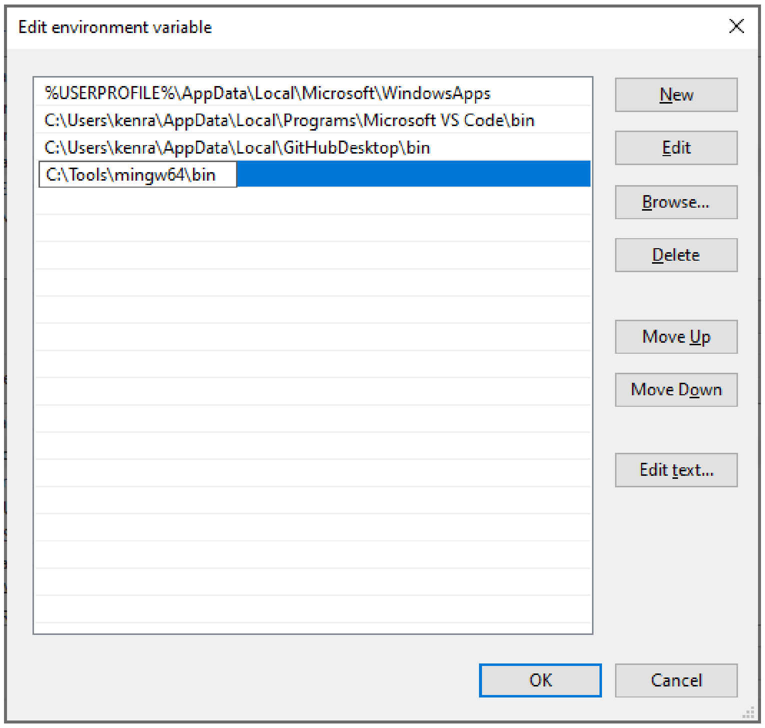

Installing MinGW-W64

Settings

Specify setup settings.

Version	8.1.0
Architecture	x86_64
Threads	posix
Exception	seh
Build revision	0

3) Change the destination folder to be C:\Tools, click Next and finish the installation
4) The compiler tools will be in the folder C:\Tools\mingw64\bin
5) In order for the command line to see the new tools, we need to add the path to the tools to the environment variable PATH. Enter envir in the Windows search path and click on the Environment Variables control panel
6) Click the Environment Variables… button and edit the user variable for PATH to add the new path to the MinGW bin folder


Edit environment variable

%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\kenra\AppData\Local\Programs\Microsoft VS Code\bin
C:\Users\kenra\AppData\Local\GitHubDesktop\bin
C:\Tools\mingw64\bin

7) Click OK to close out all the windows
8) Open a command line prompt and make sure you can run the gcc and g++ apps from any folder

```
gcc --version
g++ --version
```

<u>Install CodeLite IDE</u>

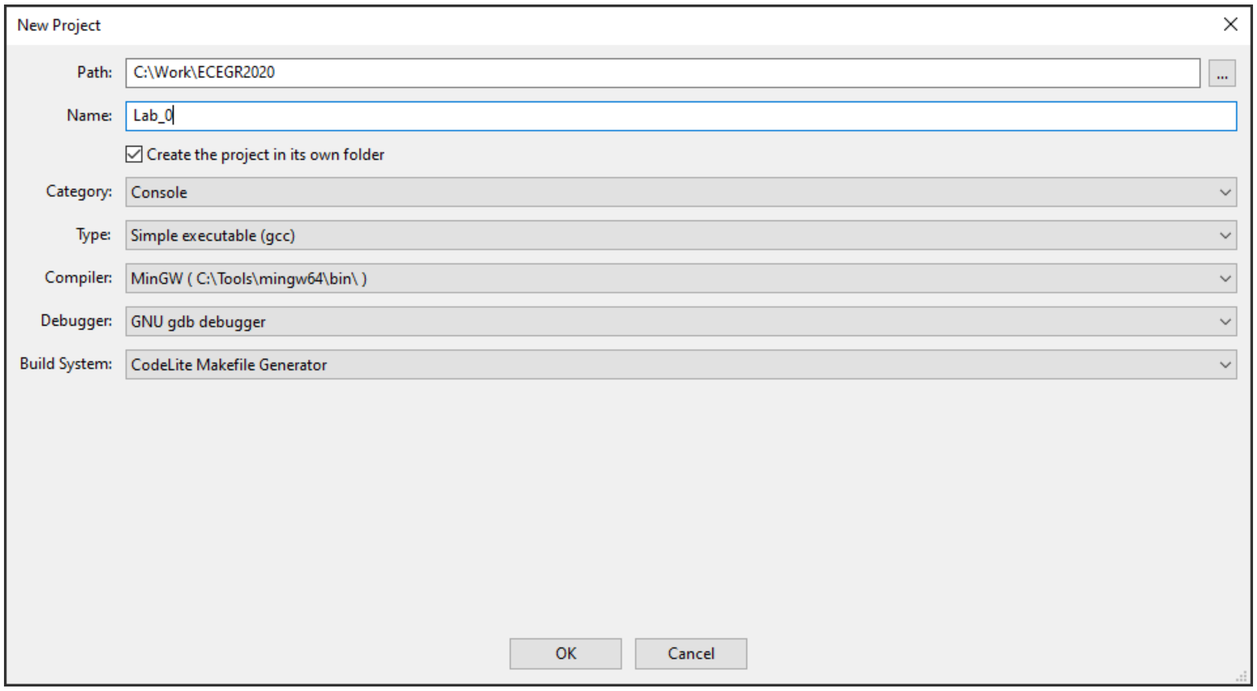
CodeLite is a free Integrated Development Environment (IDE) which makes editing and debugging code easier. You can use any editor that you like as well as run the compiler from the command line to build code. But an IDE will pull all the separate tools together along with the ability to debug the source code within the editor

1) Download CodeLite from https://codelite.org/ run the setup
2) Launch CodeLite once the install completes
3) Decline downloading a newer version of CodeLite
4) Click Next in the Setup Wizard
5) Select C/C++ Development for the Development Profile
6) Click Scan to setup compilers. The install of MinGW with the path should be listed. Highlight MinGW and click Next
7) Finish the install. CodeLite will restart
8) Again, decline the download of a newer version. You can turn that prompt off under Settings -> Misc -> Check for new version on startup (uncheck that option)
9) Create a new Workspace of type C++
10) Select a location for your workspace and name it ECEGR2020

Compile, Run, and Debug C program

The reason every computer language has a "Hello World" example is to ensure that the developer has all the tools installed and configured correctly to create, compile, run and debug a very simple program.

1) In CodeLite, select File -> New -> New project
2) Name the project Lab_0 and select the following settings:



New Project

Path: C:\Work\ECEGR2020

Name: Lab_0

☑ Create the project in its own folder

Category: Console

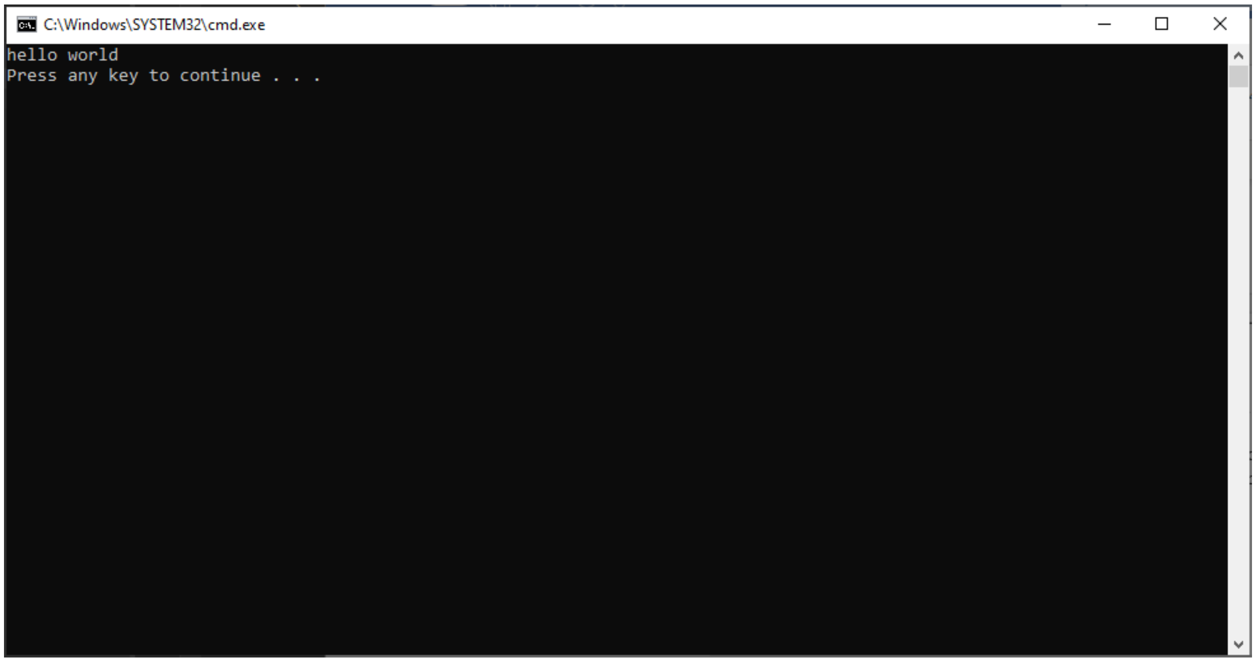
Type: Simple executable (gcc)

Compiler: MinGW (C:\Tools\mingw64\bin\ )

Debugger: GNU gdb debugger

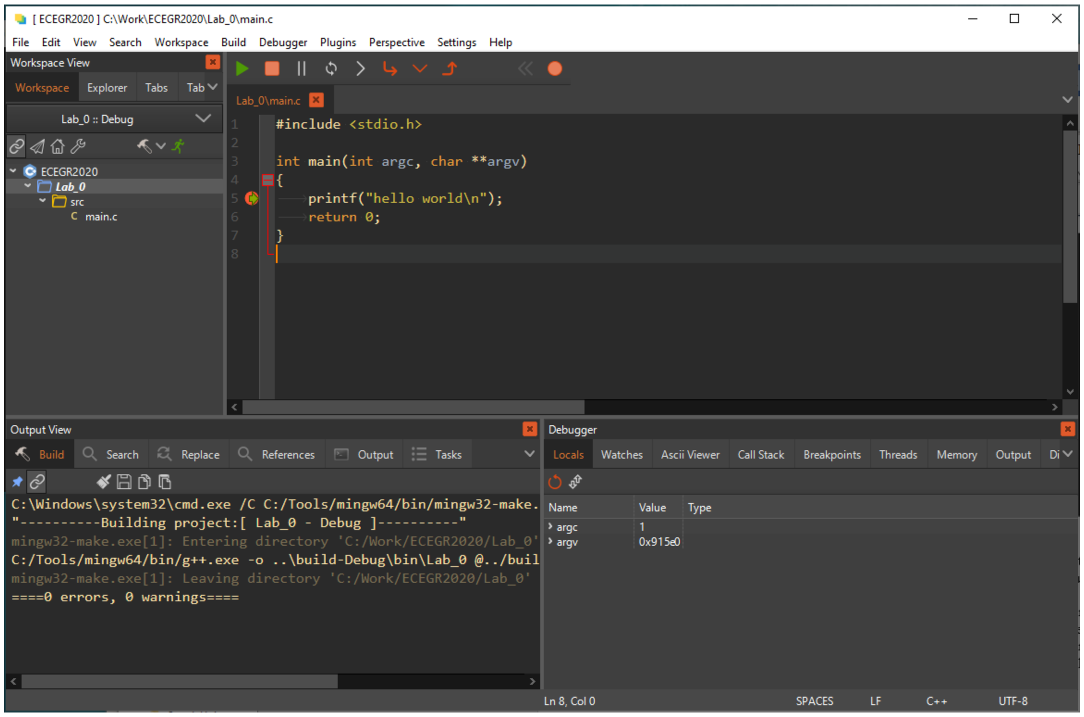
Build System: CodeLite Makefile Generator

3) In the Workspace View, open up the Lab_0 folder and src folder to get to main.c. Click on main.c to open the file in the editor. You should see the C source code for a simple Hello World app
4) Build the project by selecting Build -> Build Project. The Output View will open and display the build steps taken. There should be 0 errors and 0 warnings
5) Run the program by selecting Build -> Run. An option to Execute or Build and Execute may pop-up. Select Build and Execute and check the box to remember your selection. The Hello World app will open in a console window and show the following:



```
hello world
Press any key to continue . . .
```

6) Press a key to end the application
7) In the editor for main.c, click area right next to 5 for the printf() function to set a breakpoint. A red dot should appear. You can also place the editor cursor on the printf() line then select Debugger -> Toggle Breakpoint.
8) Select Debugger ->Start/Continue Debugger. An option to Execute or Build and Execute may pop-up. Select Build and Execute and check the box to remember your selection.
9) This time a console window will open but also the editor will show an arrow at line 5 and extra stuff will be displayed. We are now actively debugging the Hello World app and the debugger has stopped execution at line 5


[ECEGR2020] C:\Work\ECEGR2020\Lab_0\main.c
File Edit View Search Workspace Build Debugger Plugins Perspective Settings Help
Workspace View
Workspace Explorer Tabs Tab
Lab_0 :: Debug
ECEGR2020
Lab_0
src
main.c
Lab_0\main.c
Output View
Build Search Replace References Output Tasks
Debugger
Locals Watches Ascii Viewer Call Stack Breakpoints Threads Memory Output Di
Ln 8, Col 0 SPACES LF C++ UTF-8

```
1  #include <stdio.h>
2
3  int main(int argc, char **argv)
4  {
5      printf("hello world\n");
6      return 0;
7  }
8
```

```
C:\Windows\system32\cmd.exe /C C:/Tools/mingw64/bin/mingw32-make.
"----------Building project:[ Lab_0 - Debug ]----------"
mingw32-make.exe[1]: Entering directory 'C:/Work/ECEGR2020/Lab_0'
C:/Tools/mingw64/bin/g++.exe -o ..\build-Debug\bin\Lab_0 @../buil
mingw32-make.exe[1]: Leaving directory 'C:/Work/ECEGR2020/Lab_0'
====0 errors, 0 warnings====
```

Name	Value	Type
argc	1	
argv	0x915e0	

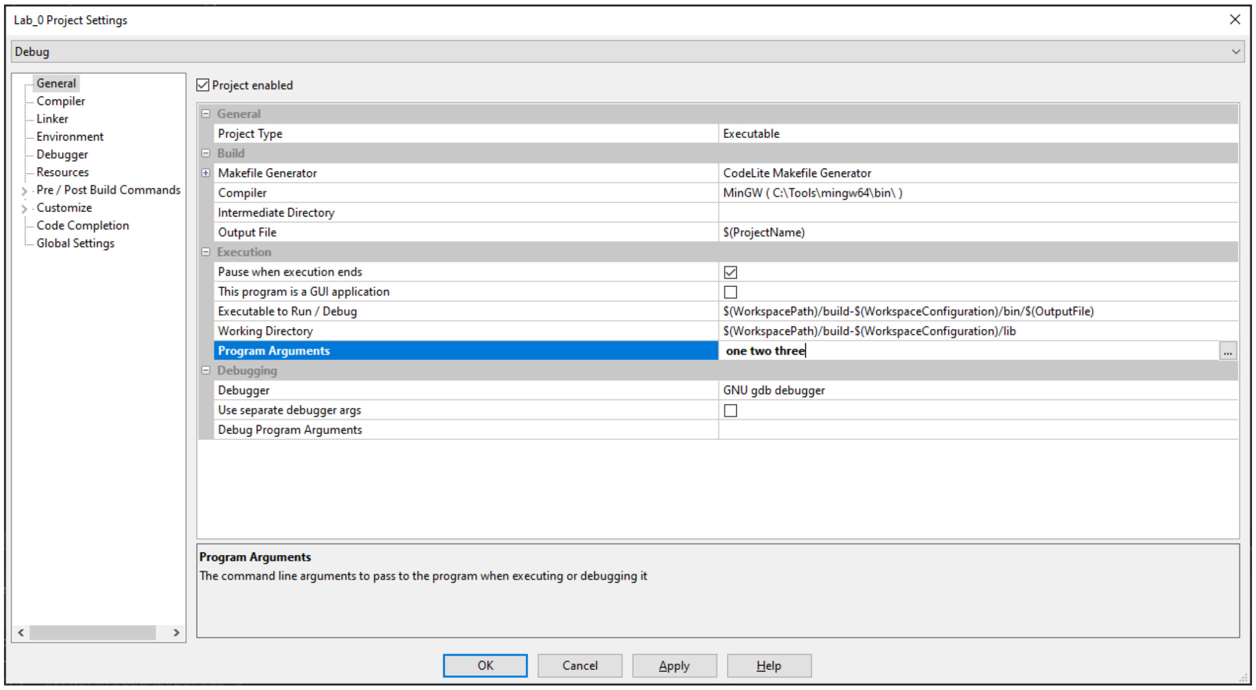
10) Look at the Debugger window and see that the variables argc and argv are display with values. Click on the arrow next to argv and see what is displayed
11) Hover the cursor over the buttons above main.c. These are the debugger execution buttons that help you control the debugger and the application. The green arrow will let the app run until it either ends or hits the next breakpoint. You can also single step with the Next button. Press that and notice what happens with the app's console output. The "hello world" is printed out
12) Click the red button to stop the debugger. This also stops the app as well
13) Change main.c to add the following source

```
#include <stdio.h>

int main(int argc, char **argv)
{
        printf("hello world\n");

        for(int i = 0; i < argc; i++ )
        {
                printf( "%d - %s\n", i, argv[i] );
        }
        return 0;
}
```

14) Run the debugger again with the breakpoint still active on line 5. Single step through the code with the Next button and notice how the debugger view of Locals changes with each line. Also note what is being output by the app in the console window. End the program
15) In the Workspace View, right click the Lab_0 project and select Settings. Under General, change the Program Arguments to add the following: one two three



☑ Project enabled

General	
Project Type	Executable
Build	
Makefile Generator	CodeLite Makefile Generator
Compiler	MinGW (C:\Tools\mingw64\bin\ )
Intermediate Directory	
Output File	$(ProjectName)
Execution	
Pause when execution ends	☑
This program is a GUI application	☐
Executable to Run / Debug	$(WorkspacePath)/build-$(WorkspaceConfiguration)/bin/$(OutputFile)
Working Directory	$(WorkspacePath)/build-$(WorkspaceConfiguration)/lib
Program Arguments	one two three
Debugging	
Debugger	GNU gdb debugger
Use separate debugger args	☐
Debug Program Arguments	

Program Arguments
The command line arguments to pass to the program when executing or debugging it

16) Press OK and debug the application again by single stepping through the for loop.Notice what changes in the debugger Locals view as well as the app output. End the application.
17) Debug the app one more time and single step to the for loop. Click on the Watches tab and click on the green + button to add the following variables to watch: i, argc, argv[i]. Notice the values of those variables and how they change as you continue single stepping. Watches are powerful in that you can not only look an individual variable values as you are debugging, but you can use more complex expressions like argv[i] which is using the value of i to index into the array of argument strings being passed to the application.

Save Work to GitHub Repository

Source control repositories allow for tracking history of source code changes and supports collaboration amongst developers on a common codebase. First we need to create a local git repository (git repo) then we'll push that repo to GitHub to store the source in the cloud.

1) Open a command prompt and cd to the location of the ECEGR2020 workspace folder

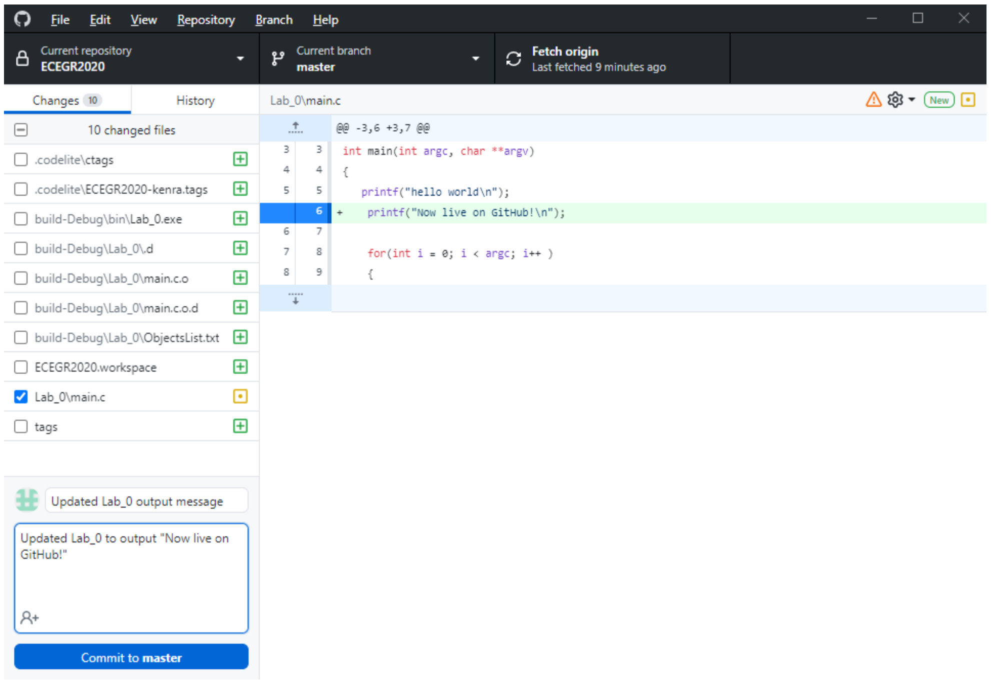
2)	git init	to initialize a new repo for all your ECEGR work
3)	git status	to see the current status of the repo. Nothing new is added
4)	git add Makefile	adds Makefile to repo
5)	git add Lab_0	adds Lab_0 folder and contents to repo
6)	git status	shows the files that are staged ready for commit to repo
7)	git commit	will open an editor. Add Lab #0 First Commit in editor, then exit. Commit will complete with the commit comment
8)	git status	

The local repo has been populated with the Lab_0 project. Now we need to push that repo to GitHub. We'll use GitHub Desktop for that

9) Open GitHub Desktop
10) Select File -> Add local repository…
11) Choose the path to the ECEGR2020 folder
12) Click on History tab and you'll see the commit of Lab_0 files and what was changed
13) Select Repository -> Push to publish the local repo to GitHub
14) Go to GitHub.com and log in. You should see your ECEGR2020 repro there

Lastly, you will make a change to the Lab_0 main.c, compile and test the change, commit the change to the local repo and push the change to GitHub

15) Add a line of code to printf("Now live on GitHub!\n");
16) Compile and debug the application to ensure the change works
17) In GitHub Desktop notice that the file Lab_0\main.c is showing as changed. Click on main.c and notice that the changes to the code are highlighted
18) Make sure only main.c is check marked and add a commit message and description. The better the commit message, the easier it is to see the history of a project on GitHub



10 changed files

- [ ] .codelite\ctags
- [ ] .codelite\ECEGR2020-kenra.tags
- [ ] build-Debug\bin\Lab_0.exe
- [ ] build-Debug\Lab_0\.d
- [ ] build-Debug\Lab_0\main.c.o
- [ ] build-Debug\Lab_0\main.c.o.d
- [ ] build-Debug\Lab_0\ObjectsList.txt
- [ ] ECEGR2020.workspace
- [x] Lab_0\main.c
- [ ] tags

```
@@ -3,6 +3,7 @@
 int main(int argc, char **argv)
 {
    printf("hello world\n");
+     printf("Now live on GitHub!\n");

     for(int i = 0; i < argc; i++ )
     {
```

Updated Lab_0 output message

Updated Lab_0 to output "Now live on GitHub!"

19) Commit the change to the master branch
20) Push the change to GitHub. Refresh the browser view of GitHub.com and see the history of main.c