June 13, 2019

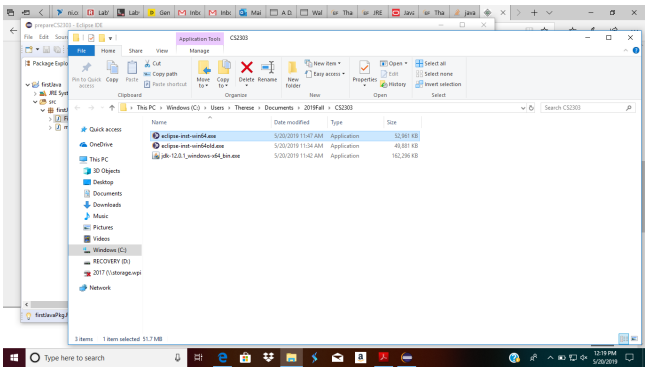

Figure 1: The JDK installer after downloading it, accompanied by two Eclipse installers.

0.1 Java Development Kit (JDK)

The latest release of Java seems to be jdk-12.0.1. I downloaded it from `https://www.oracle.com/technetwork/java/javase/downloads/jdk12-downloads-5295953.html` into C:/Documents/2019Fall/CS2301.

0.2 Eclipse

I needed to use the Eclipse installer found at `https://www.eclipse.org/downloads/download.php?file=/oomph/products/latest/eclipse-inst-win64.exe\&mirror\_id=492`. That is why there are two installers shown in the directory.

In it I selected to use jdk-12-0.1, which was not the default. The directory at this point appears as in Figure 1.

0.3 Running a first Java program

Needed to learn about modules. See Figure 2

0.4 C and C++ Infrastructure

Need to support C and C++, so, can we get C compiler?

Looking at `https://www.cs.odu.edu/~zeil/cs250PreTest/latest/Public/installingACompiler/`.

Choosing the MinGW version. going to `http://www.mingw.org/`. Don't forget MSYS. There find a downloads page at `https://osdn.net/projects/mingw/releases/`. Scroll to the download file, see Figure 3.

It defaults to going to C:/MinGW.

Be sure to select what you need, see Figure 4.

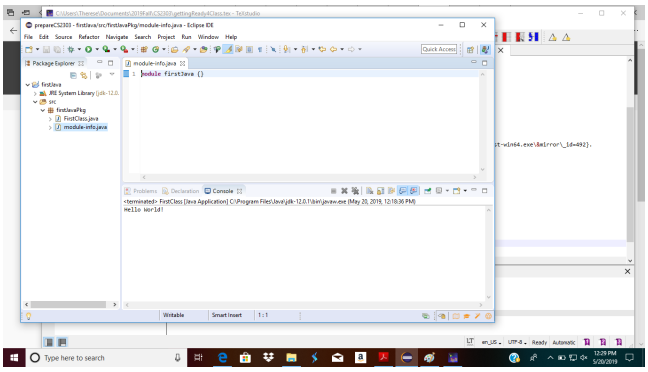

Figure 2: Showing a running Eclipse, with a first Java project.

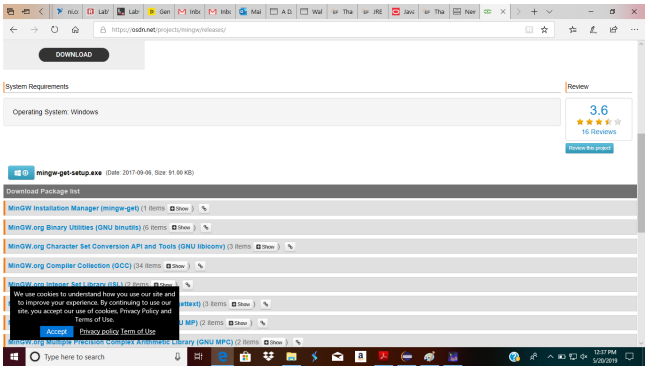

Figure 3: We need C/C++, which we are getting via MinGW. See the download executable on the left, you might have to scroll down to find it.

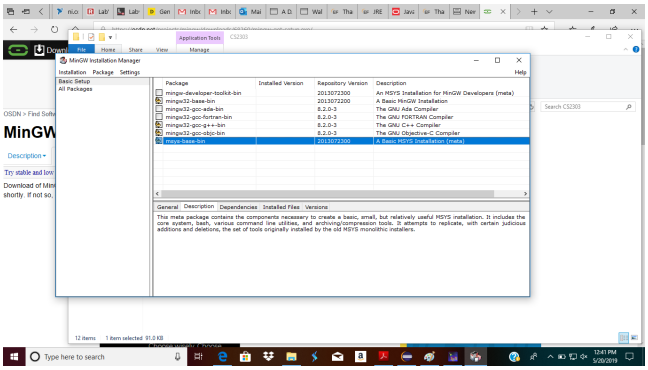

Figure 4: We have selected the components we want: C, C++, basic and MSYS.

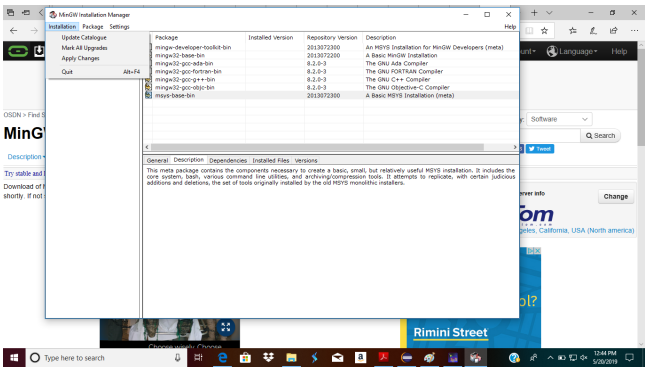

Figure 5: We need to choose ApplyChanges.

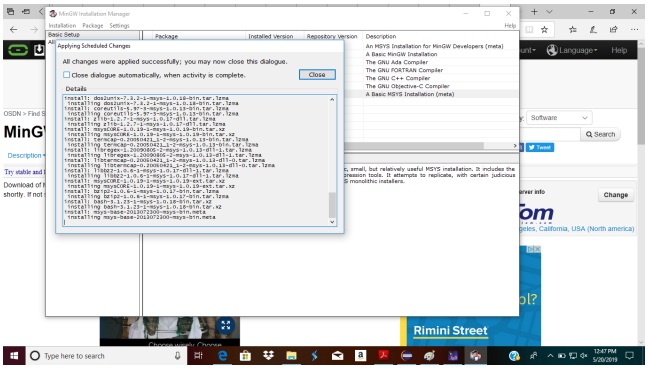

Figure 6: We can see the tool applying those changes, i.e., adding our needed capabilities.

Do we need to say "Apply Changes?", see Figure 5.

Yes, we do. When it is done, it should show as in Figure 6.

Next, we should be able to get Eclipse CDT, a C and C++ development environment, into Eclipse. We will look on Eclipse Marketplace for it. See Figure 7.

See Figure 8.

We will choose to install, and we will confirm, see Figure 9.

Don't worry about information messages (such as it is not available), be of good heart and continue. Accept any license request.

Allow restarting for changes.

Let's see how we did. We can try to make a C project. See Figure 10.

You should find C project as a choice, see Figure 11.

Choose the Hello,World package. See Figure 12.

It should show you a Hello, World c file in the editor. You can edit the print out, and put your name on the file. See Figure 13.

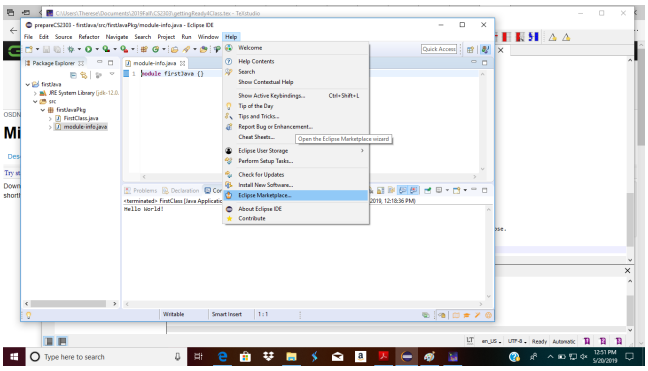

Figure 7: We found Eclipse Markeplace under the Help Menu using the Eclipse Marketplace menu item.

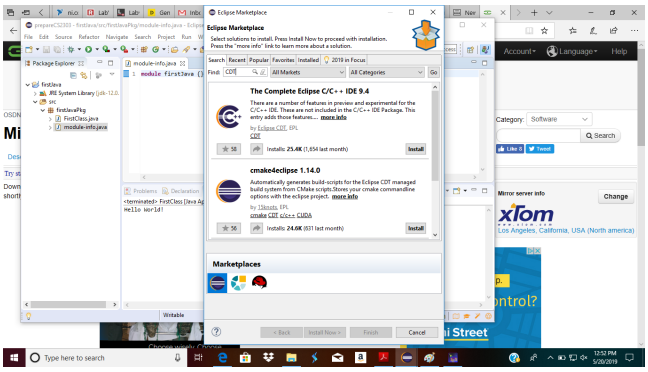

Figure 8: We entered CDT in the search window near the top, and obtained a list of related software. We click install in the area of the application we want.

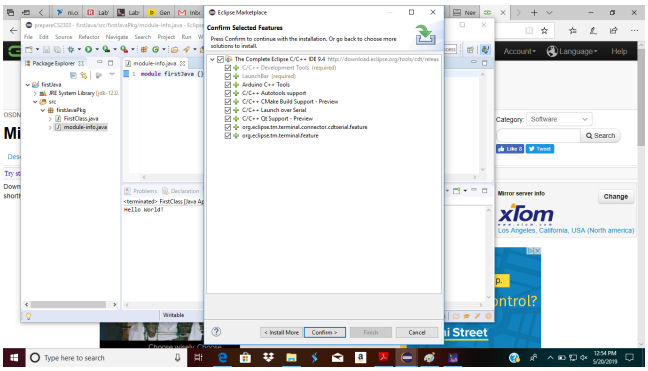

Figure 9: Eclipse shows us a list of items that make up what we just chose, and we confirm.

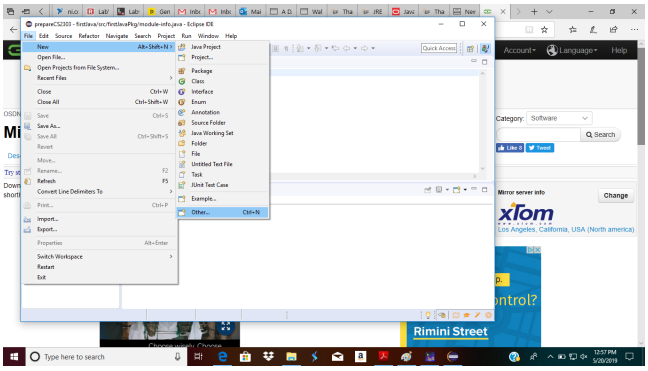

Figure 10: We use the newly installed capability to make a C project, starting with File/New/Other.

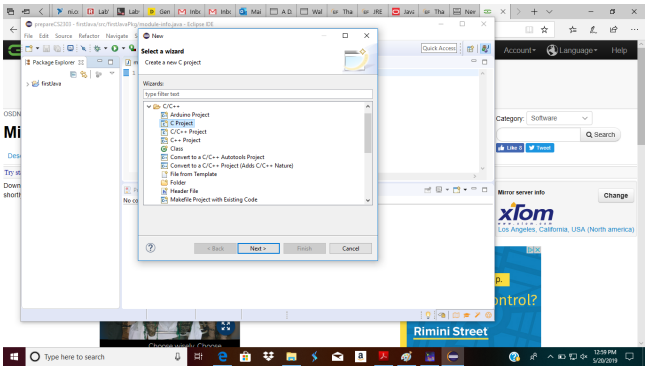

Figure 11: We choose C project within the C/C++ category.

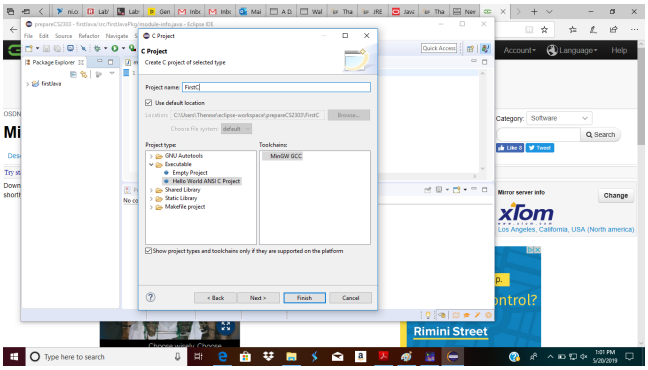

Figure 12: Be sure to choose Hello World. Note that the MinGW C is present as a toolchain. If you have multiple tool chains, and MinGW is one, you might wish to select it.

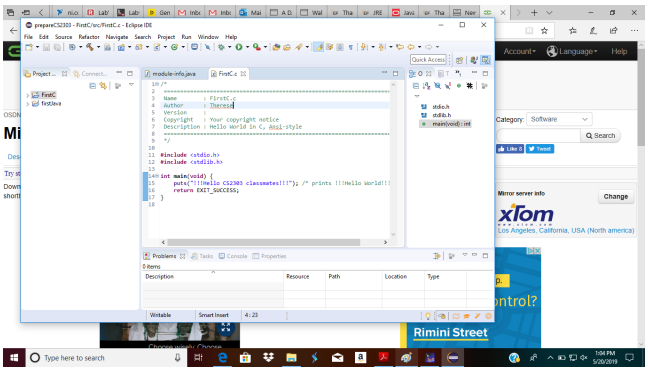

Figure 13: We can edit the print statement in our new project.

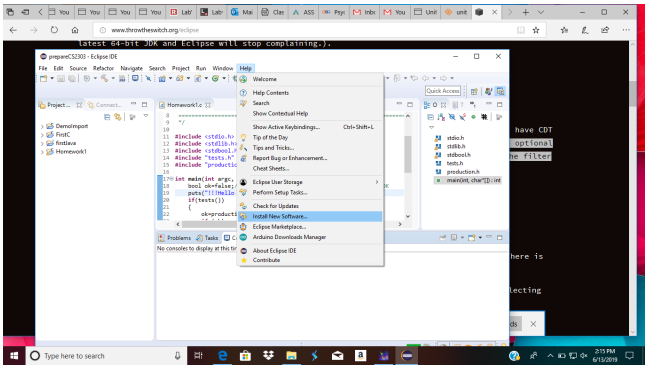

Figure 14:

The Unity test framework is at: `http://www.throwtheswitch.org/unity`.

Use Help-¿Install Software-¿CDT, that is, click help, select install new software, enter CDT in the text box, and observe the results, under CDT Optional Features. Check the checkboxes, so that the unit testing support items are included.

0.5 Run Code!

Next we want to compile, link and run this code. First we must build the code, then we can run it. For building the code, see Figure 18. Note that, build automatically is not enough.

When the build succeeds, look for console output as shown in Figure 19.

We will build a run configuration to support that process.

Under the run menu we can find the choice "Run configurations". There are also other ways to find that choice. See Figure 20.

We are making a run configuration for a C executable. Note in Figure 21,

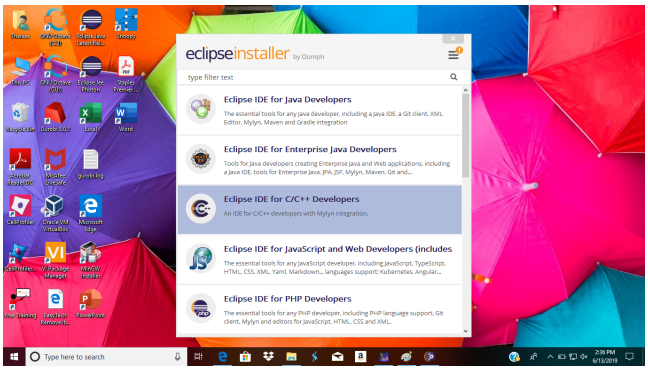

Figure 15: selecting the CDT version of Eclipse

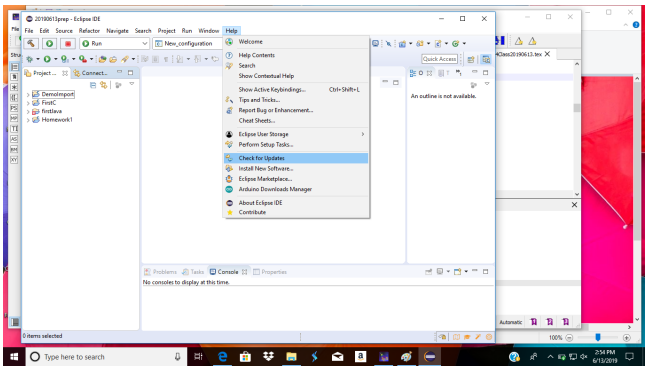

Figure 16: There were many. You can accept them all.

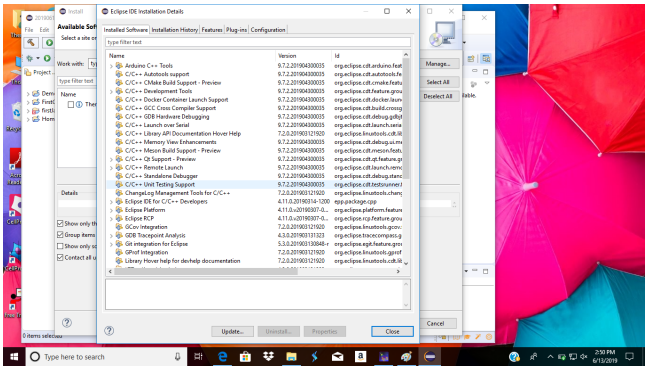

Figure 17:

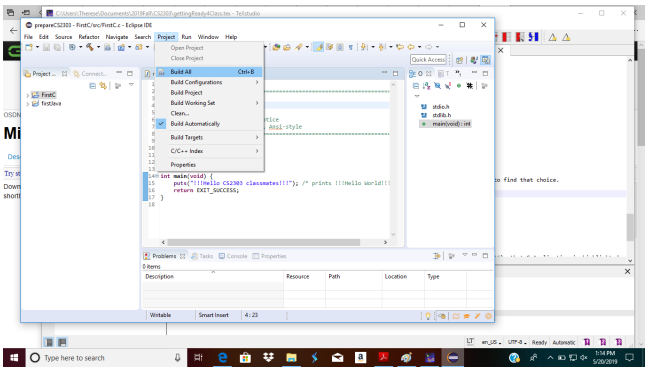

Figure 18: To show that our new code runs, we first build it.

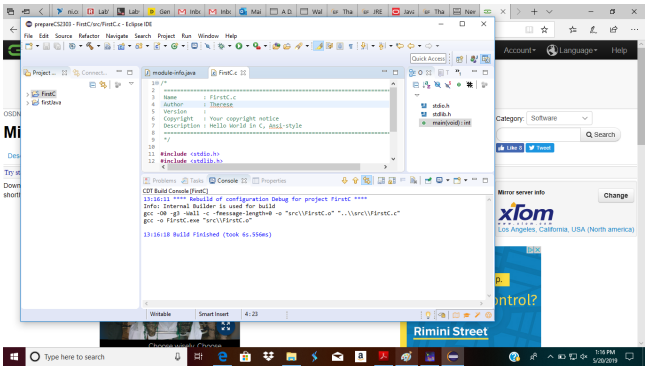

Figure 19: The results of building are shown in the console.

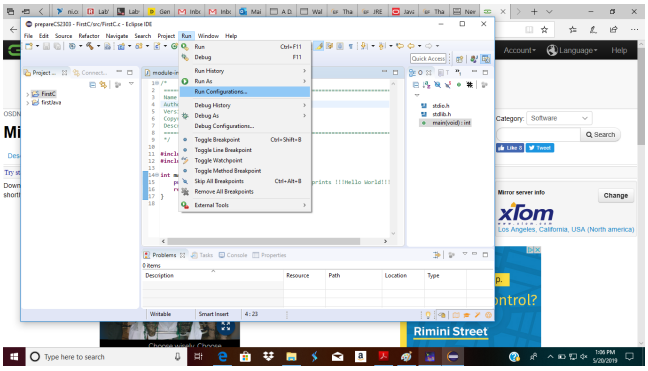

Figure 20: Next we want to create a run configuration. It will support execution of our executable we just built.

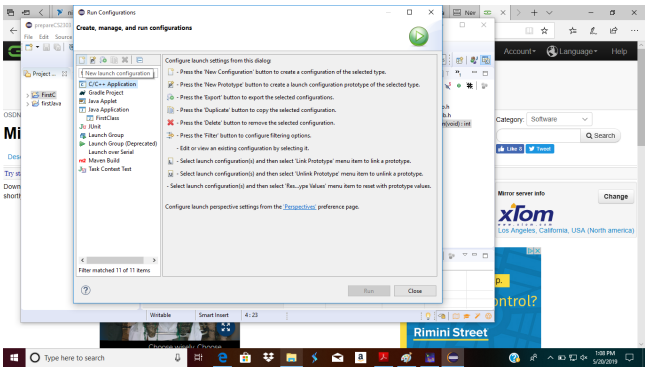

Figure 21: We select C/C++ Application and then click the icon for New.

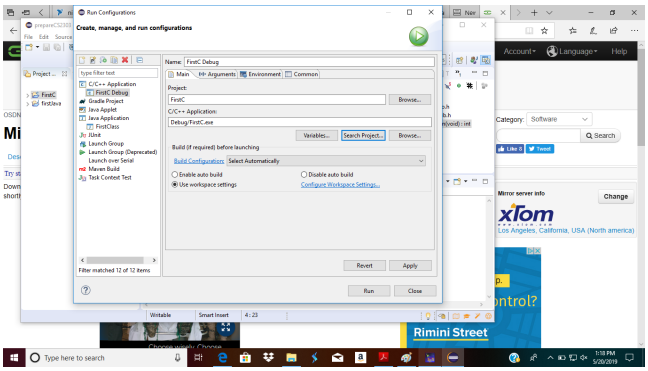

Figure 22: We see that the run configuration has been created: it appears under the C/C++ Application list on the left.

that C Application is highlighted, and the command icon for "New" is being hovered over. We will click it.

A successful result looks like Figure 22. Note that the name of the executable appears in the application name text field, having been automatically supplied.

After clicking run in the lower right of that run configuration, the file should execute. The console output should appear as in Figure 23.

In my case it was necessary to stop my security tool from rejecting the file. In case you have this problem, I solved it by running McAfee, selecting the cog on the upper right, going to real-time scanning, excluded files, and adding the executable to its list. I navigated to the executable at C:/Windows/Users/Therese/eclipse-workspace/prepareCS2303/FirstC/Debug/FirstC.exe.

After this exclusion, I was able to run the file through Eclipse.

It might be necessary to exclude each such executable.

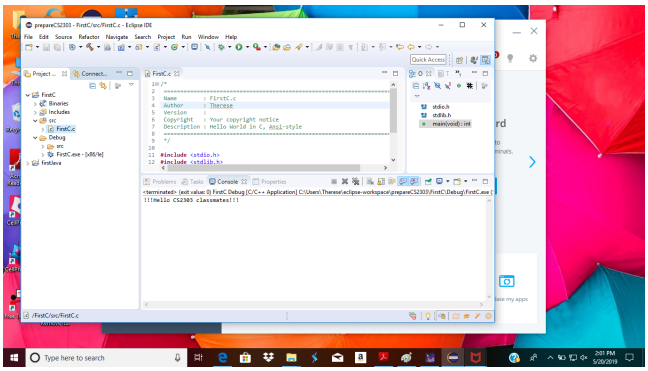

Figure 23: When we click the Run command button on the run configuration, the executable produces its output in the console window. Below terminated with exit value 0, it shows our message to CS2303 classmates.

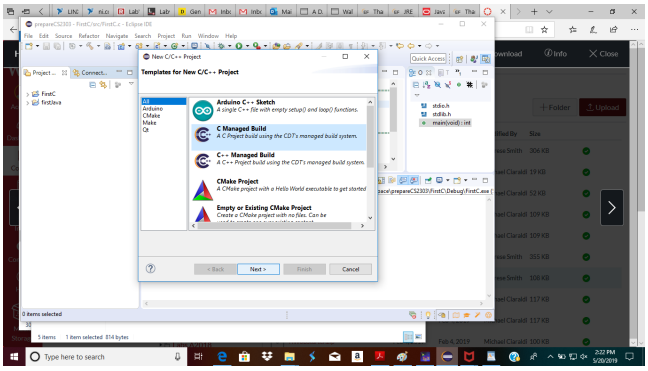

Figure 24: When making a new C project, we select managed build.

0.6 Homework 1

Let's get started on homework 1. There is a template you must use. Begin by making a new C project. See Figures 24 and 25.

The file should look familiar, see Figure 26,

but we will change it, by pasting in a different file into the editor, see Figure 27.

Now let's create the files that are referenced, without yet being available.

We need to create two new include (also called header) files, they have the suffix ".h". See Figure 28.

Assigning the name occurs in Figure 29.

After this has been done for both include files, the file explorer shows them. See Figure 30.

The code that belongs in tests.h is shown in Figure 31.

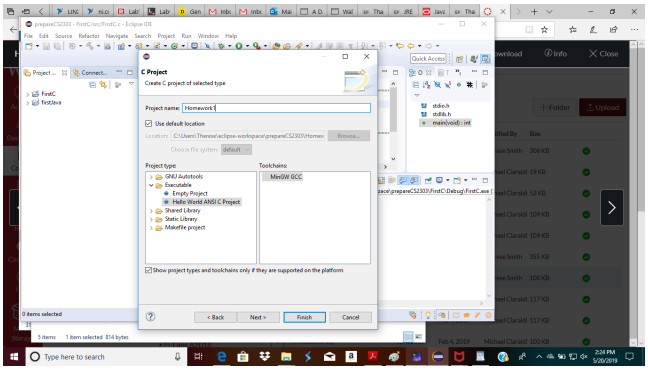

Figure 25: We invent a name for it, and we choose to start with a Hello, World project.

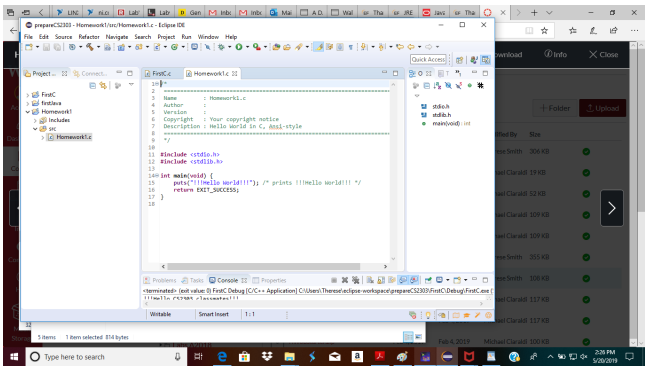

Figure 26: Here we are starting a homework project.

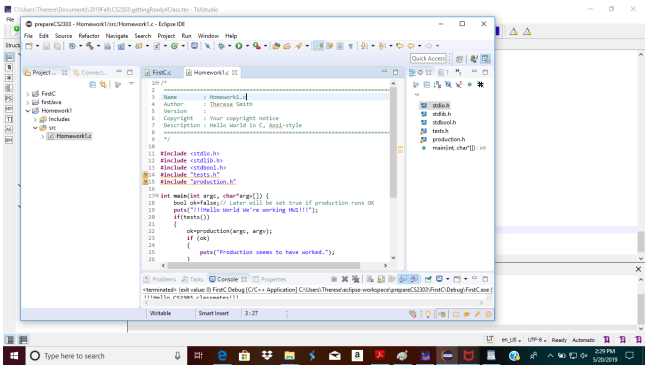

Figure 27: Notice in editor lines 14 and 15 that two files are referenced, while as yet unknown.

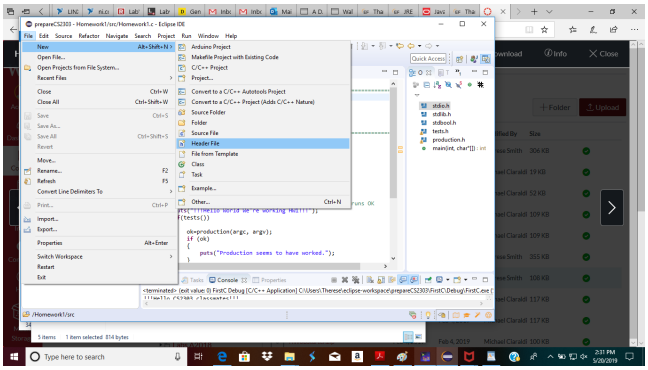

Figure 28: We can get a new header file, using File/New/Header file.

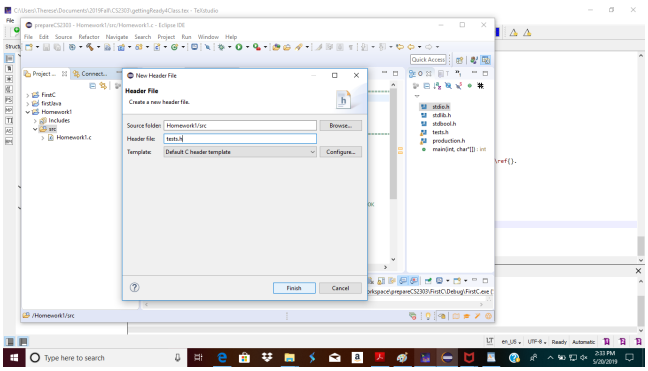

Figure 29: We need to invent a name for it, which must end ".h".

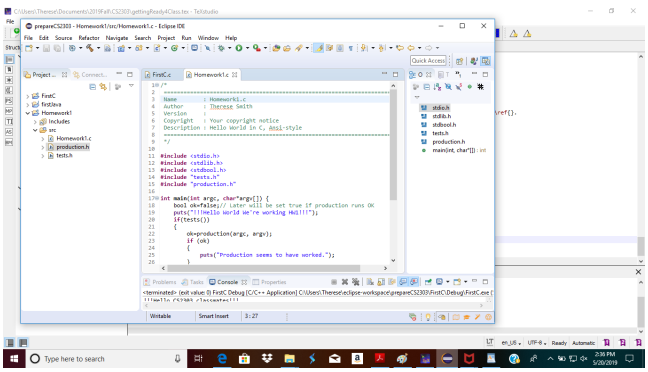

Figure 30: We can see the two new header files in the Project Explorer pane.

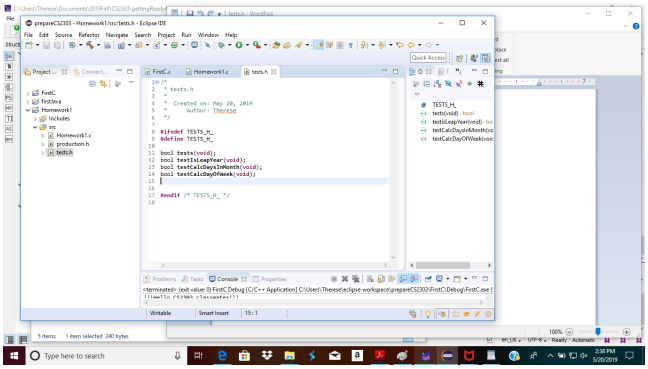

Figure 31: Header files are an appropriate place for function prototypes, seen here.

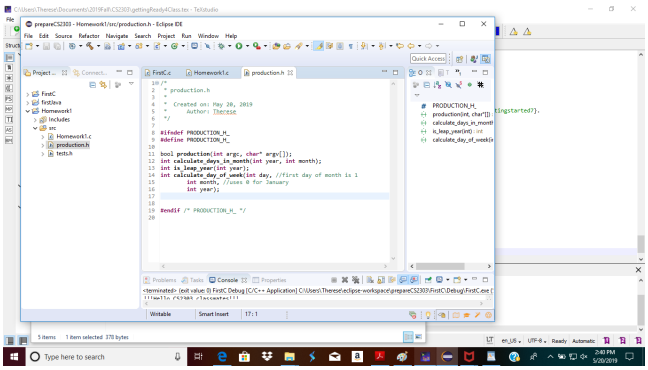

Figure 32: Function prototypes for the production.c file.

The code that belongs in production.h is shown in Figure 32.

There are also two more source code (.c) files that we want, tests.c and production.c. They are created similarly to the way we created the two header files.

These files, namely:

1. The HWx file that includes main, and invokes tests, and upon success, invokes production
2. the tests.c file that contains the function “tests”, and the test functions it invokes
3. the tests.h file that contains the function prototypes for tests.c
4. the production.c file that contains the functions being tested
5. the production.h file that contains the function prototypes for production.c

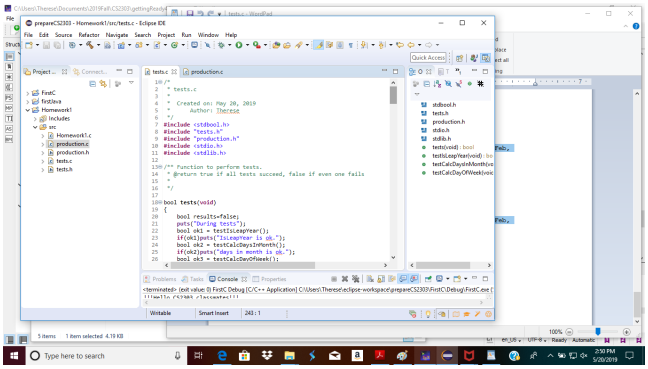

Figure 33: A ".c" file, with code for running tests.

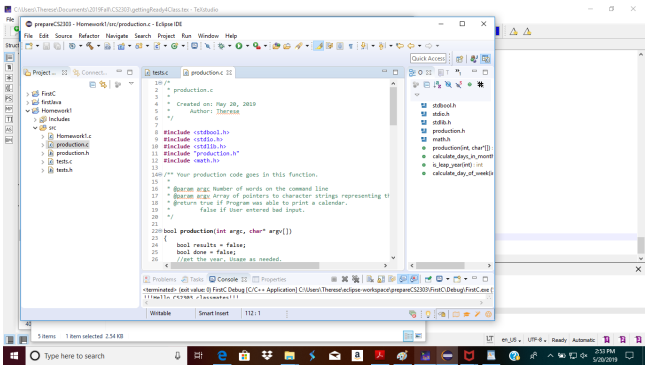

Figure 34: A ".c" file for code to be tested. Note by looking in the Project Explorer, that an executable has not yet been built.

must be used for every coding assignment in this term. The particular tests and test cases, and the particular production functions can be different in every assignment. The HWx file will be the same every time, though its name might change. Its main function implements the idea of having a suite of regression tests, and invoking them, prior to permitting the invocation of the production code. The entirety of the regression suite and the production code will be your deliverable for each homework assignment.

I will give you the starter code for these files. When you have it, the top part of the tests.c file should look like Figure 33.

The top of the production.c file should look like Figure 34.

After running Project/BuildAll, the ProjectExplorer pane shows the existence of the binary, that we need for the run configuration. See Figure 35.

We create a new run configuration, as before, see Figures 36 and 37.

After hitting the run command in the lower right of the run configuration, we should obtain console output as shown in Figure 38.

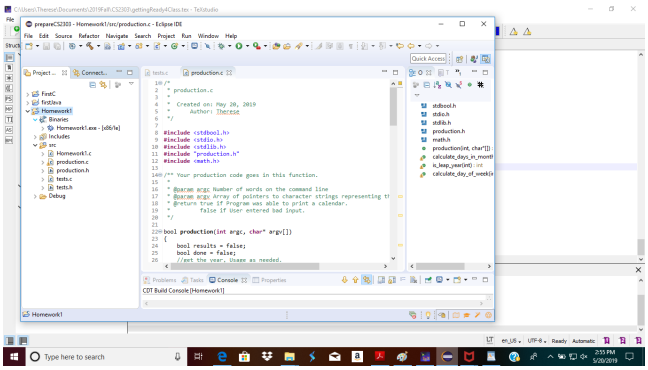

Figure 35: Observe in the Project Explorer, that an executable has been built.

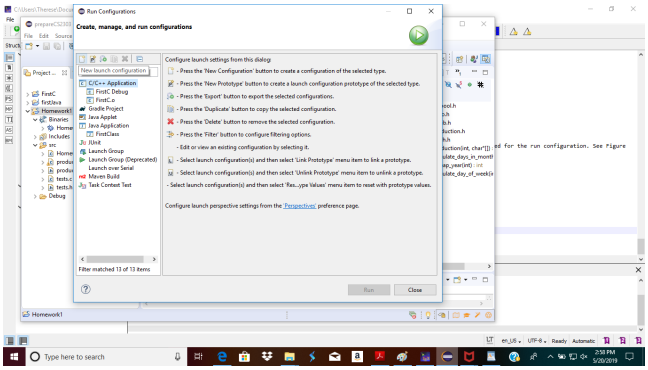

Figure 36: We again create a run configuration. One way to reach this panel is the Run menu, the Run Configurations menu item.

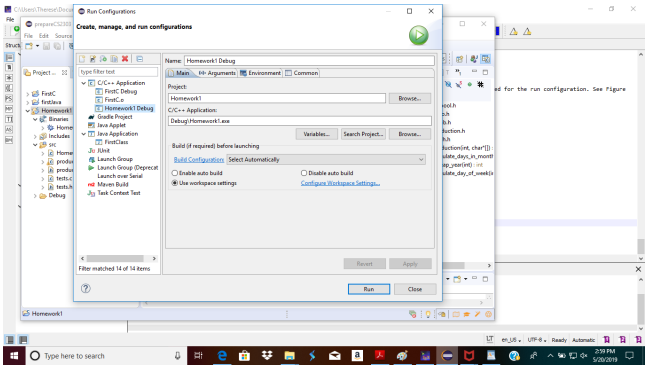

Figure 37: For this simple program, once the run configuration is made, we can execute by clicking Run.

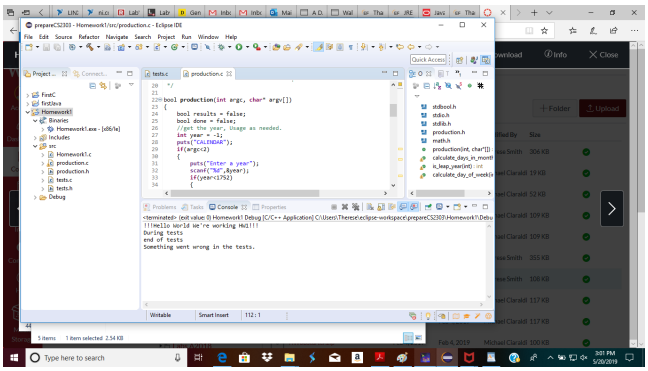

Figure 38: The program's output appears in the console.

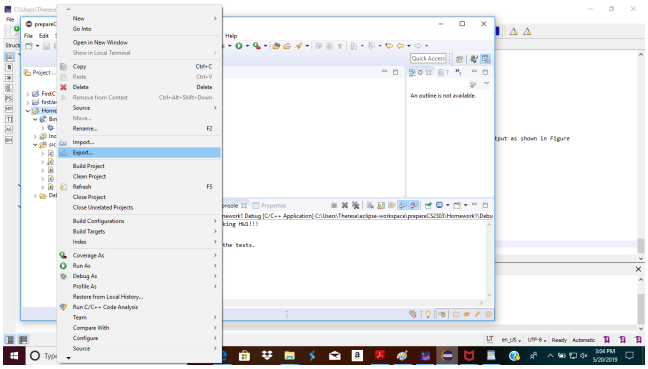

Figure 39: A context menu, including the export menu item, can be obtained by right-clicking the project name in the Project Explorer pane.

0.7 Exporting a Project

For submitting homework, you must export your project. Let's go through this process. We'll export the project just seen, called "Homework1".

One way to obtain the export menu item is to right-click the project. See Figure 39.

We want to export to an archive, which is found under General. See Figure 40.

Place the archive in your directory tree, preferably somewhere you will remember. See Figure 41, for an example.

Don't forget to click "Finish".

You should be able to see the product in your file system, for example, see Figure 42.

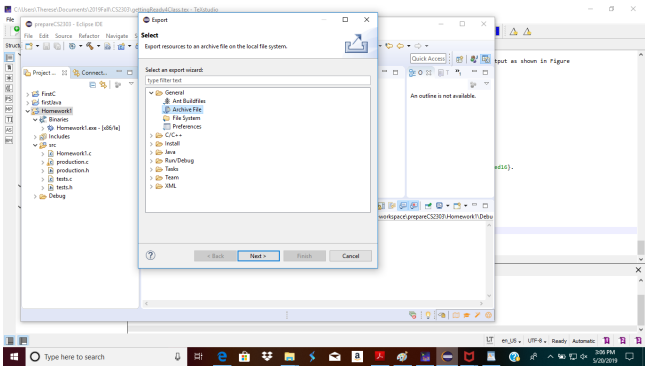

Figure 40: We're choosing to export to an archive.

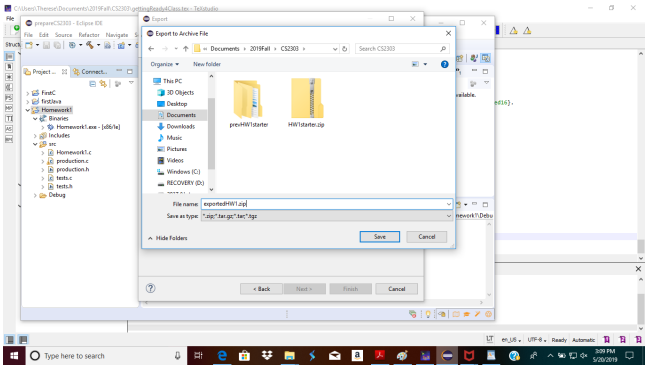

Figure 41: We've browsed to a place to put the archive, and also given the archive a name.

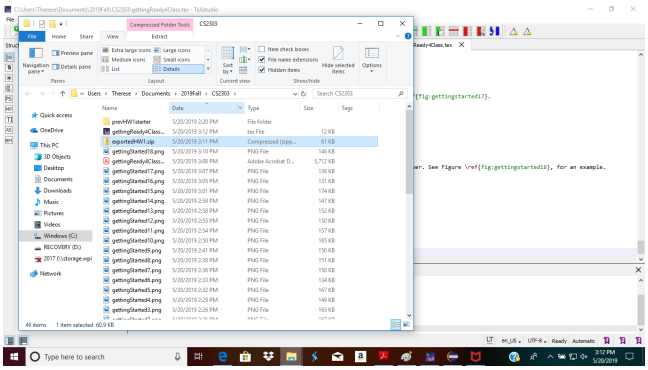

Figure 42: In the file system, we can see the exported archive.

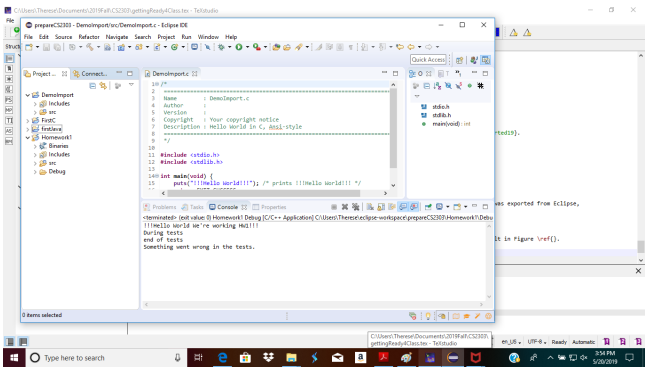

Figure 43: We've made a new project called DemoImport.

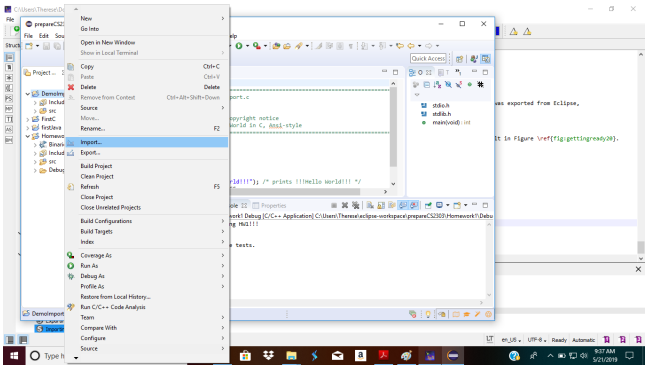

Figure 44: Right click on the project was used to bring up the context menu. Import is the menu item desired.

0.8 Importing a Project

You will be given starter code for use in homework, and the form of this code will be a zip file that was exported from Eclipse, as you just saw.

In this section, we show a way to import such a file into Eclipse.

I made a new C project called DemoImport, in the same way we made a C project previously. See the result in Figure 23.

Now I will import the zip file just made, into that project. See Figure 44.

It's an archive. See Figure 45. We click next.

We arrive at a panel with a browse button, which we click, see Figure 46 for the panel that implements the browse.

After selecting (use the command button) and executing the browse, we return to the panel that allowed us to browse, with our selection populating the upper and left parts of the panel, see Figure 47.

Now we are ready to import, by clicking Finish. We can see the result in

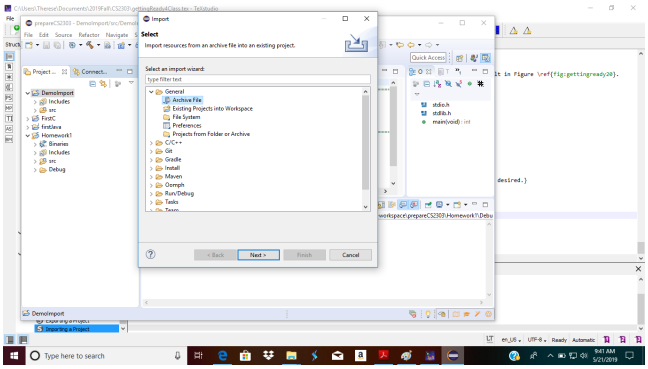

Figure 45: We're choosing to import from an archive (contents will be extracted).

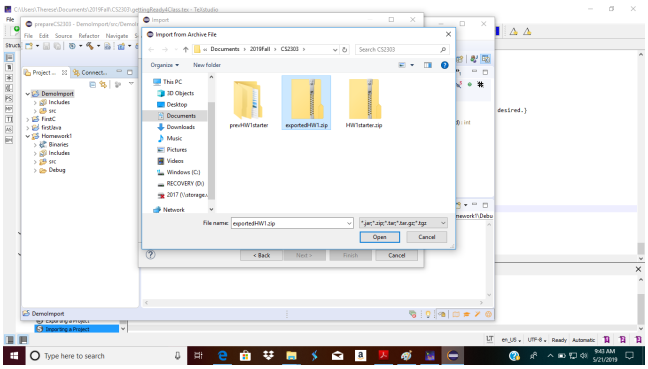

Figure 46: We browse to the specific archive we want.

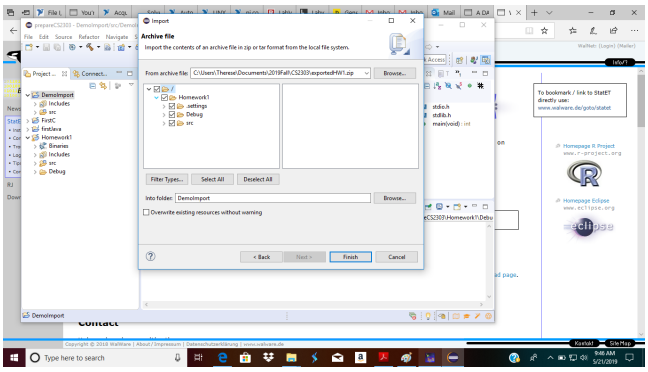

Figure 47: When we browse, the target appears in the text area at the top. Below on the left, we can explore the contents of the archive, and select items within it, for import.

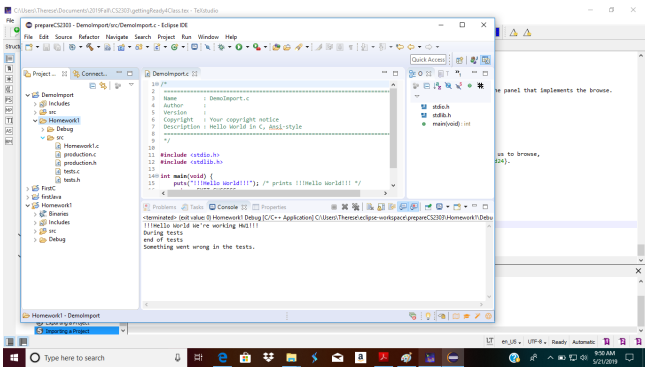

Figure 48: The results of the import are visible in the Project Explorer pane.

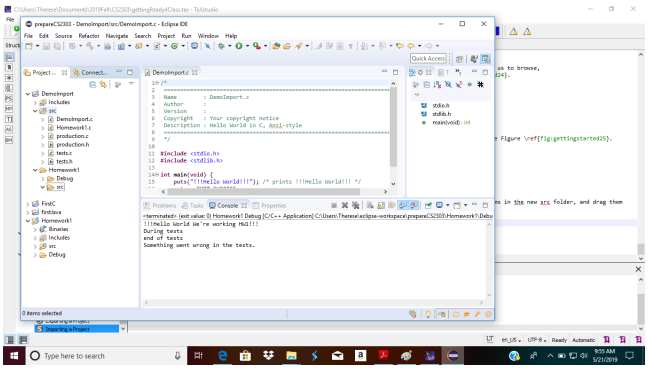

Figure 49: See, in the Project Explorer pane, that the desired files have been selected/dragged to the folder where we want them to be.

the Project Explorer pane, see Figure 48.

The newly imported contents are not exactly where we would like them to be. We can select all the items in the new src folder, and drag them into the desired src folder (the one under the project DemoImport). See Figure 49.

The DemoImport.c file, created for us by the system, has a main method. The Homework1.c file also has a main method. We do not want two mains. We will change the name of the function in the DemoImport.c file to main2. See Figure 50. It probably would have been better to make the executable and run configuration in the DemoImport file before importing.

Let's see whether we can build an executable, using Project/BuildAll. We can tell we succeeded by the appearance of the Binaries entry in the Project-Explorer. Let's see whether we can create a run configuration and run. We can tell we succeeded by seeing the console output. We can edit the newly imported file, to change that console output. See Figure 51.

Once we've saved the file, we can build and run again. See Figure 52 for the

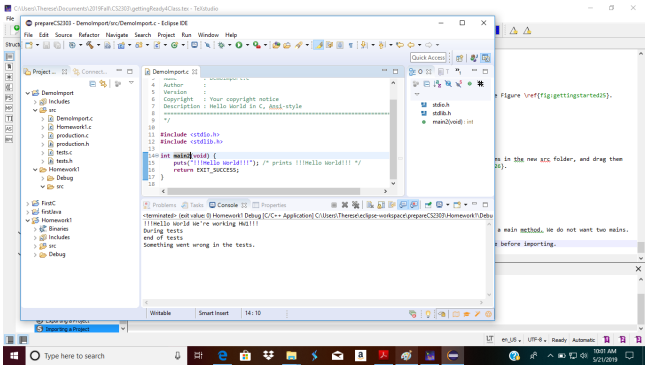

Figure 50: We do not want two main functions. We edit the name of the one we do not want.

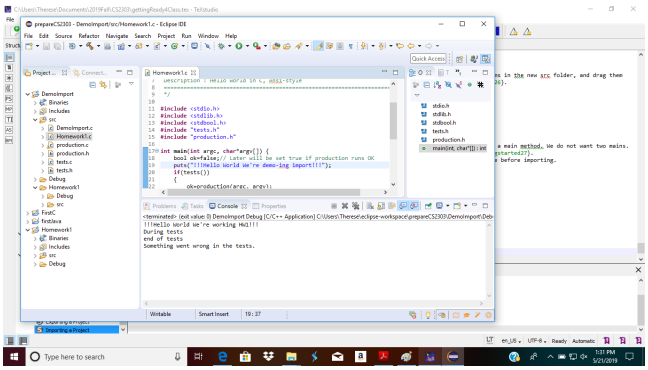

Figure 51: We edit the puts statement in the main function that we do want.

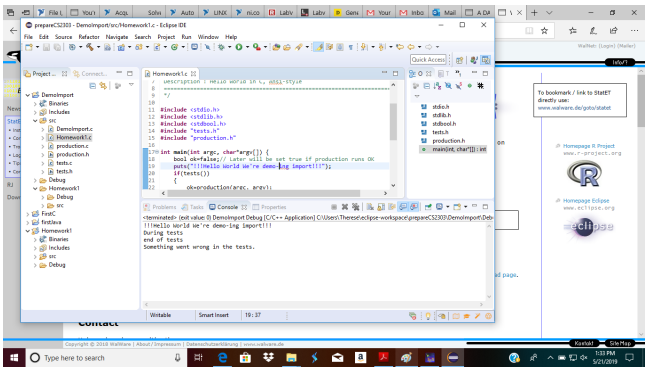

Figure 52: When we run the run configuration for this project, we see the revised output in the console.

console output showing the results of running the edited code.

Congratulations if you have arrived at this point. It seems you are well-underway for the learning in CS2303.