Lab 1: Downloading IDE/Flashing Code
Due Date: September 8, 2023, at 5 pm

Prerequisites:

- Knowledge of how to follow directions
- Access to a computer and internet
- A course-approved laptop
- Knowledge of how to run an application installer
- Knowledge of arrays and “printf” statements
    - How to declare and initialize an array
    - How to populate elements in an array
    - How to create a string using arrays

Background information:
This lab will guide you to make changes to the source code, the code that will be executed by the microcontroller, and project to allow you to use “printf” on the embedded board. It is to be noted that this lab is mainly to get you (the student) familiar with the IDE and flashing code to the microcontroller. Print statements such as “printf” will not be allowed for debugging purposes. There may be some contexts where we want to use “printf” to print to the console, but those will be explicitly stated. The student will lose points if “printf” statements are used outside explicit direction. This also includes commented-out code; if the following is implemented in the code upon it being turned in:

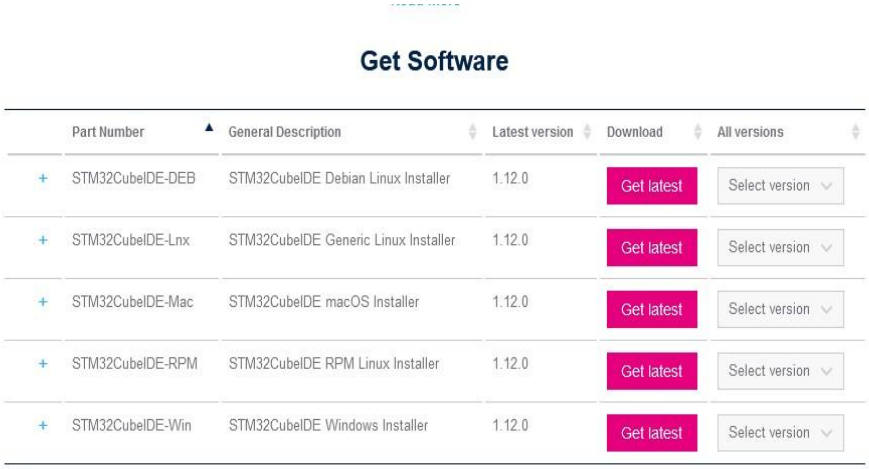
```
//printf("This register value is:   %d", registerToPrint);
```

Points will still be deducted from the lab grade.
Proper debug techniques will be taught in this class, and the expectation is that they get followed to the best of the student’s ability

Refer to the prelab for Coding Hierarchy.

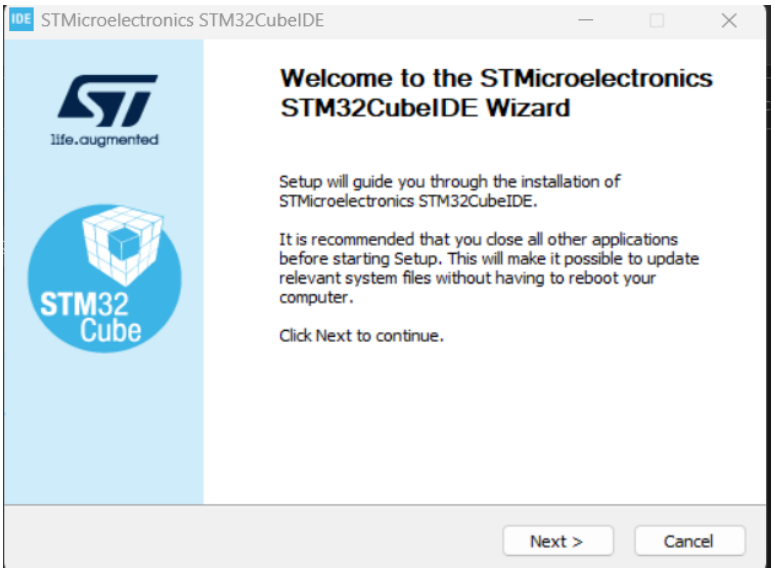
Lab instruction:

1. Download the IDE
    a. Go to https://www.st.com/en/development-tools/stm32cubeide.html
    b. Go to the ‘Get Software’ section

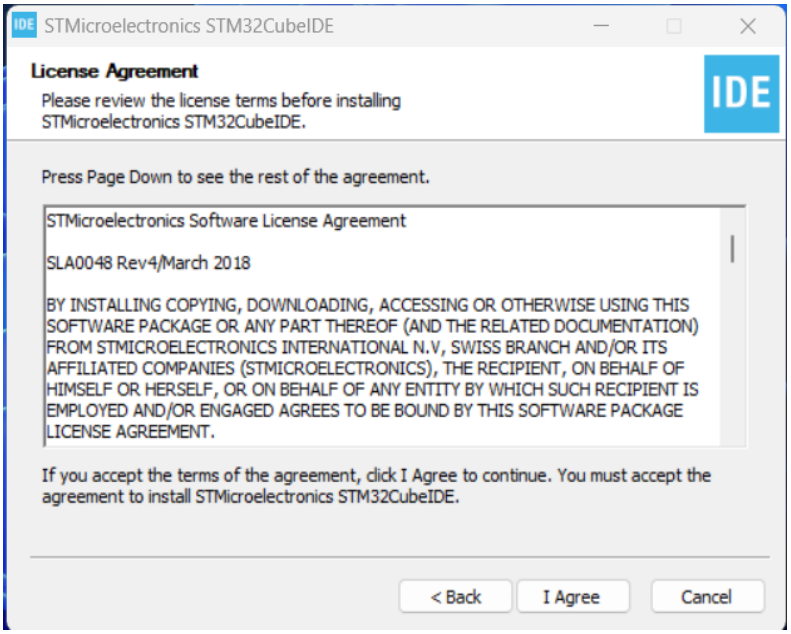



i. Select “Get Lastest”
ii. It should then ask for information to request download, enter your information (use your school email address)
iii. Within a few minutes, you should get an email from STM, open it and click the download link
iv. This should bring you back to the site, where you will need to go back to the ‘Get Software’ section and select ‘Download Latest’
v. The installer file should download

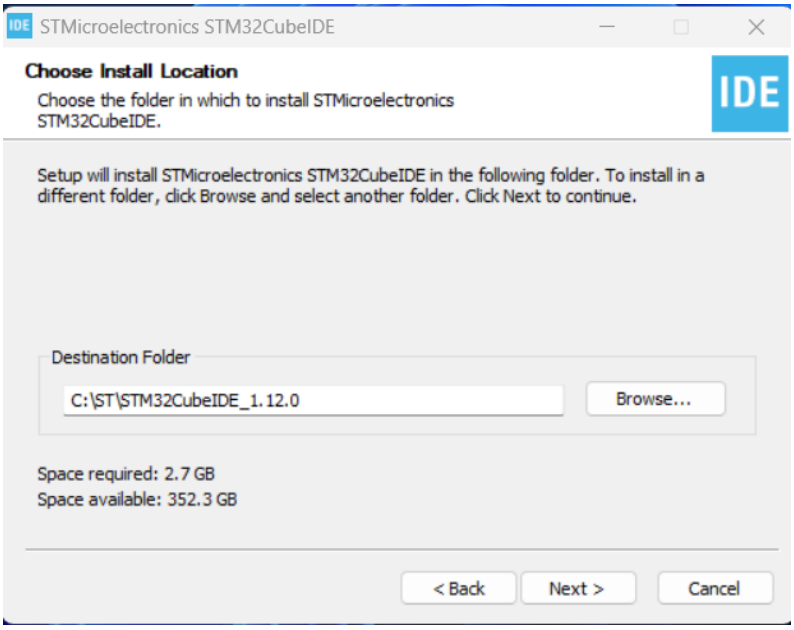
c. Download using the Installer
   i. For Windows:
      1. Double click the installer, you may get asked if you want to allow this program to make changes to your PC, select “yes”.
      2. The following window should then come up



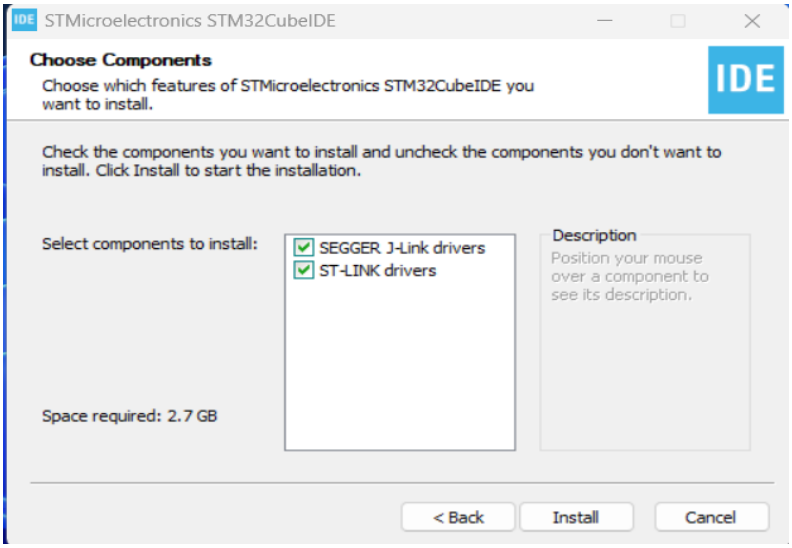

      3. Click Next
      4. Read the license agreement and Click Accept



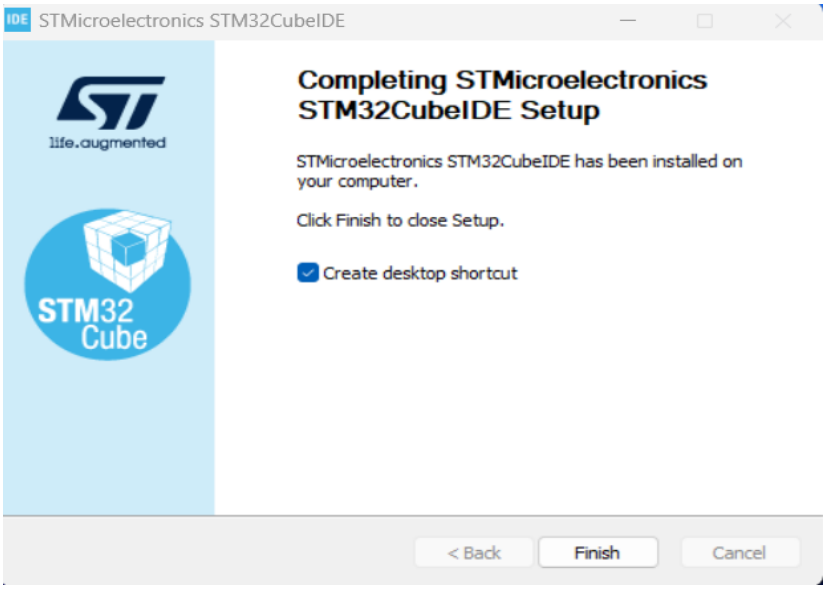

5. Choose the install location, the default is usually sufficient




6. Install the Segger J-Link Drivers and ST-Link Drivers
    a. *These will allow our PC to be able to communicate with relevant components on the board.*
    b. *This option **may not** be shown on Macs but you will still have them installed I believe*

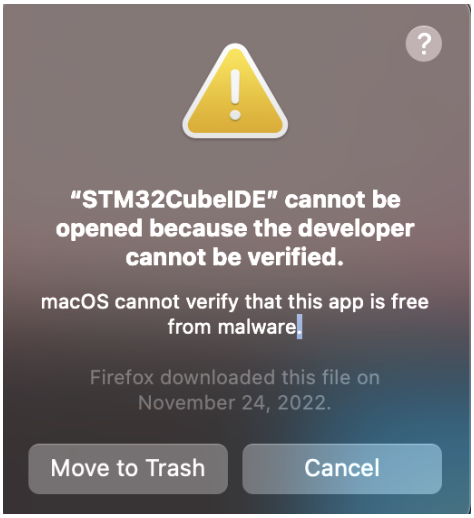



7. Create Desktop shortcut

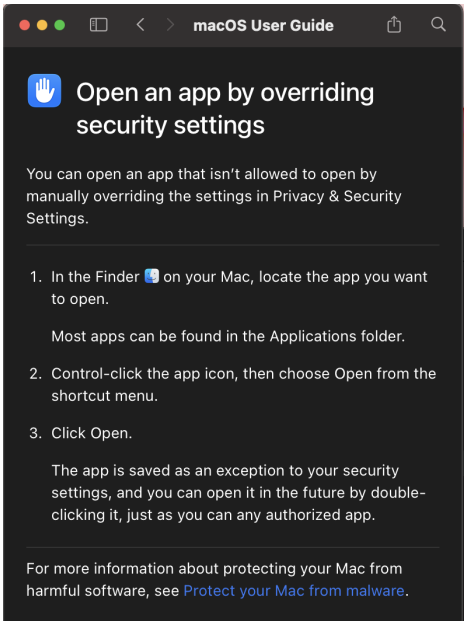



ii. For Mac users:

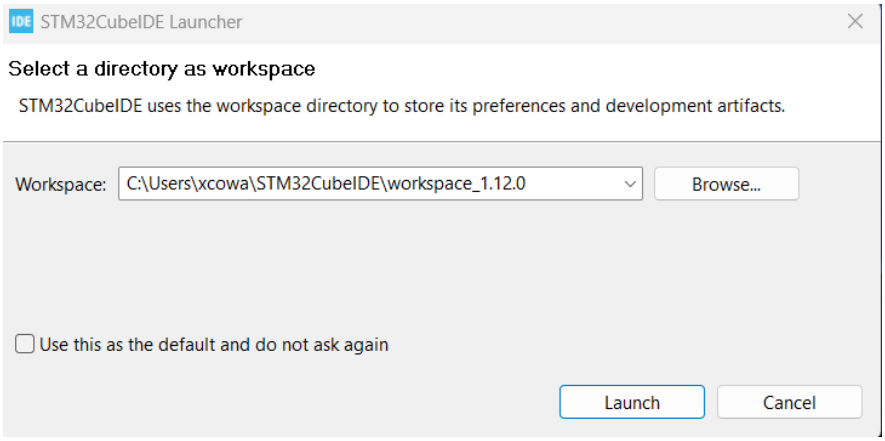
1. The process is pretty much the same for Windows, but the initial download may be a bit different due to the possibility that Apple cannot verify the “author” of the installer.
2. The “error message”:



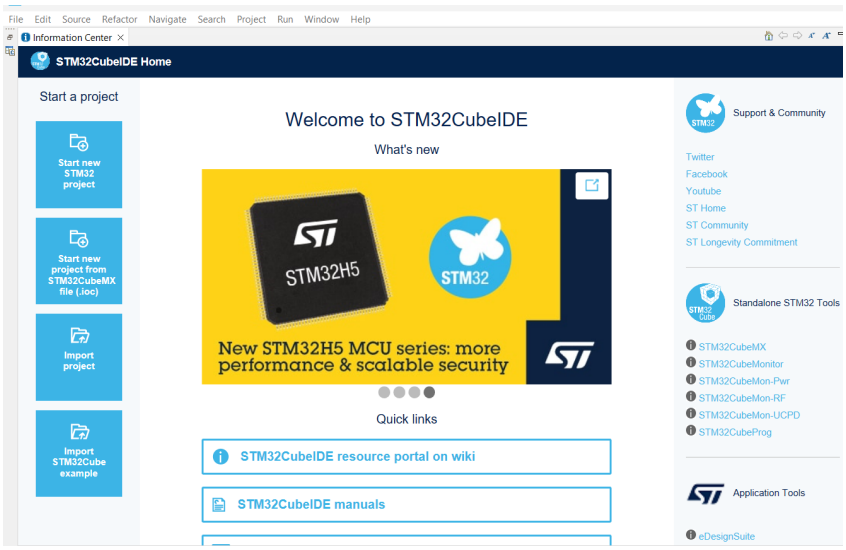

3. The Solution:



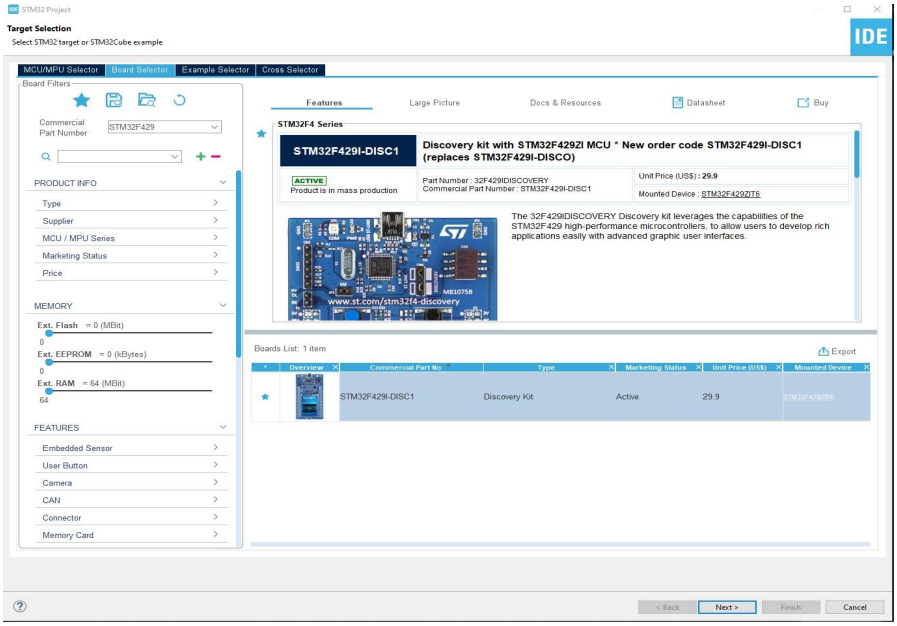

d. Open the STM32CubeIDE
e. It will ask for a "workspace"; choose whichever workspace you would like your projects to be in



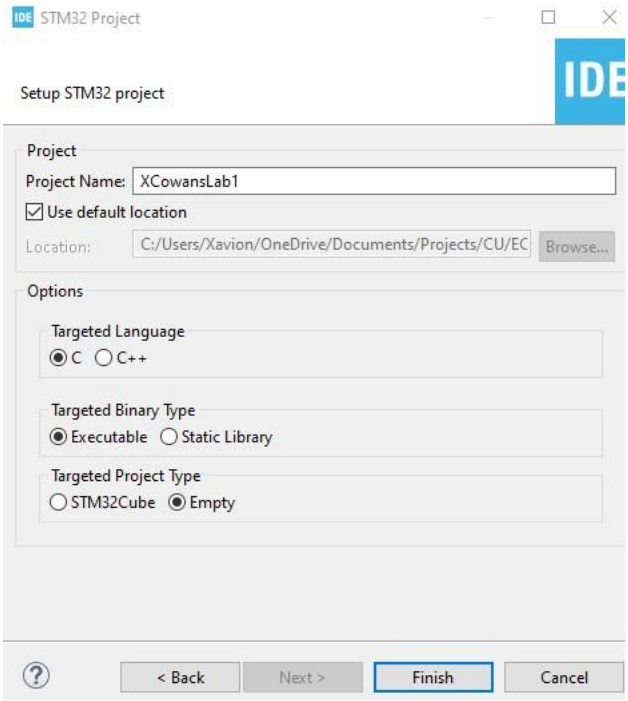

2. Create a new project
    a. You should be at a screen that looks like this:




    b. Select “Start new STM32 project” or go to File>New>STM32 Project
    c. In the Target Selection window:
        i. Go to **board selector**
        ii. type “STM32F429i” in the commercial part ID search box
        iii. Select the STM32F429I-Disc board and select “Next”

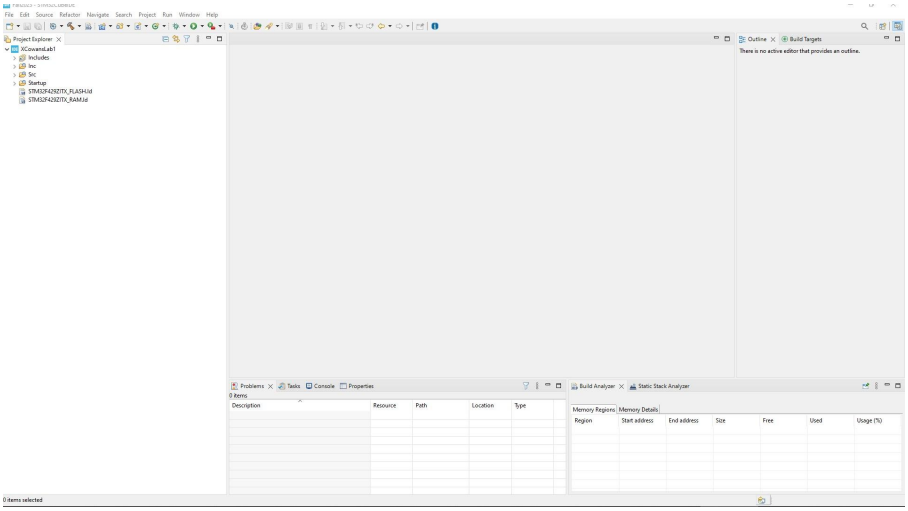



d. In the setup STM32 project:
    i. Name your project ***FirstInitialLastNameLabLabNumber***
        1. For example, XCowansLab1
    ii. Targeted Language should be C
    iii. Targeted Binary Type should be Executable
        1. *This means that the microcontroller can execute this code. Alternatively, code can be read only and it will get placed into a different section of memory. This is something you will learn about in later courses.*
    iv. The target Project type should be “Empty”
        1. Failure to select empty will lead you to a path of confusion and despair.
    v. It should look similar to this:




e. Select “Finish”

f. You should now see a screen very similar to this:



g. Open Src/main.c

   i. *This is going to be the entry point of our program. The CPU will begin execution by beginning execution of int main().*

3. Suppress warning regarding FPU (*Floating Point Unit*) usage in main.c:

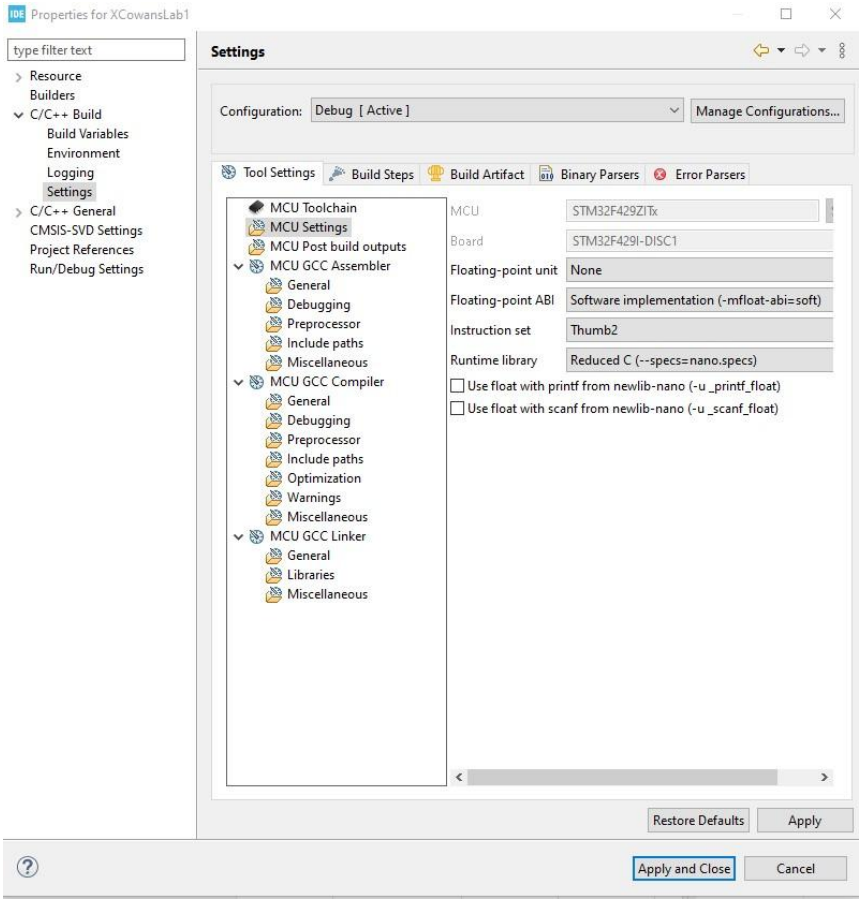
   a. Do NOT remove the lines as a method of “suppressing the warnings”

      i. While that technically “solves” the problem, it creates potential issues later down the road if/when we decide to use the FPU or become reliant on it

   b. Go to Project>Properties>C/C++ Build > Settings > MCU Settings

      i. Change Floating-point unit to “None”

ii. Change Floating-point ABI to “Software implementation”
    1. *ABI is a specification on which rules to follow.*
    2. *Changing this to “Software implementation” says floating point ABI will be supported by software and the compiler does need need to generate FPU (Floating point unit instructions)*
    3. *Here is a link that explains that and more: https://embeddedartistry.com/blog/2017/10/11/demystifying-arm-floating-point-compiler-options/*




iii. Click “Apply and Close”

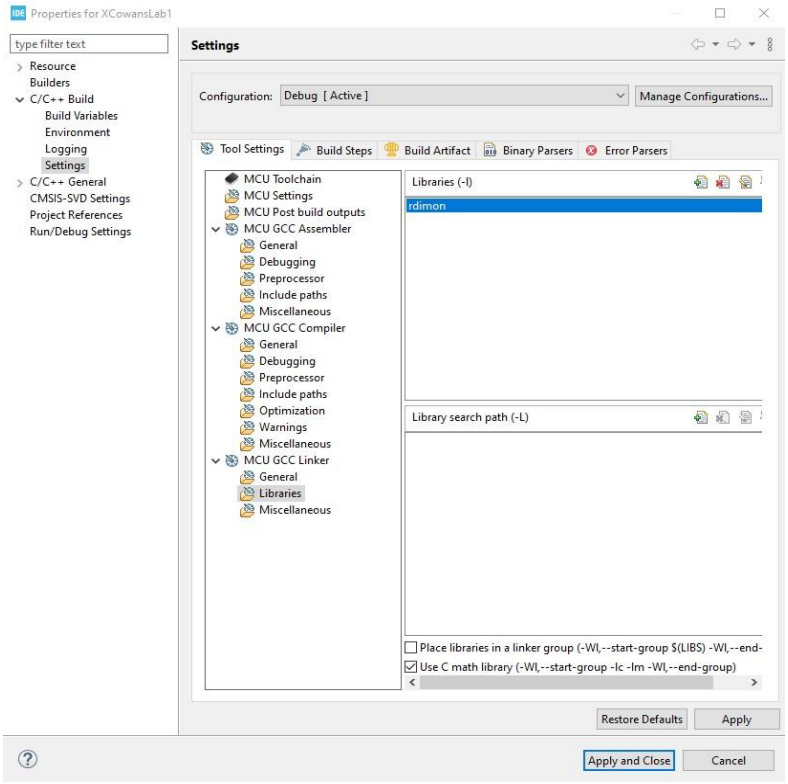
*iv. You should now see grayed-out code in main.c, this means that that section of code will not be compiled. Review compile switches in C if needed*

4. Configure project for printf functionality
    a. Delete Src/syscalls.c (Right-click the file and go to delete)
    b. Add the following to main.c:

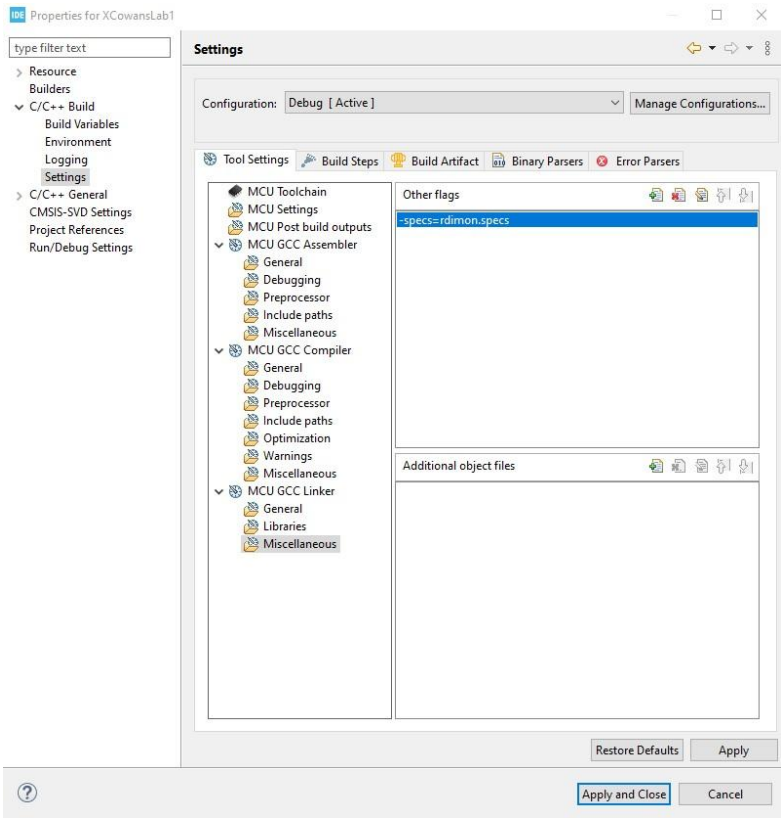
```
#include <stdio.h>
extern void initialise_monitor_handles(void);
```

    c. Add a function call to initialise_monitor_handles() as the first thing to get executed within main

d. Update Linker configuration
    i. In Project > Properties > C/C++ Build > Settings > MCU GCC Linker > Libraries, add “rdimon”
        1. *This verifies the proper libraries and functions get linked when the code is compiled. Refresh on the C compilation process if needed*



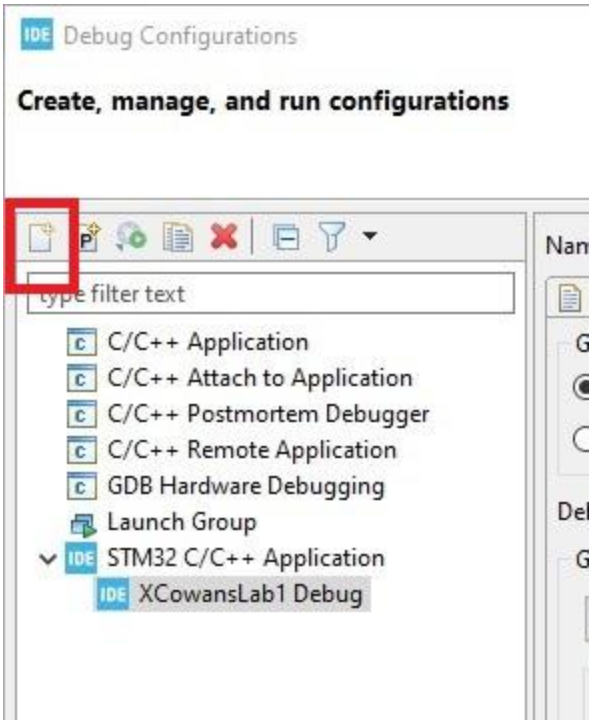

    ii. In the Miscellaneous section, add the flag -specs=rdimon.specs” in the other flags section
        1. *Compiler flags are used to tell the compiler to do or consider certain things. You will learn more about this in later courses*




iii. Click Apply and Close

e. Build the project
   i. Just click the hammer icon in the toolbar
      1. Alternatively, you can go to project > build all
      2. Alternatively, CTRL + B
   ii. The project should build with 0 errors and 0 warnings
      1. *This means that the code was able to be successfully compiled, and machine code was able to be generated. This is good because then this means this can be flashed to the board!*

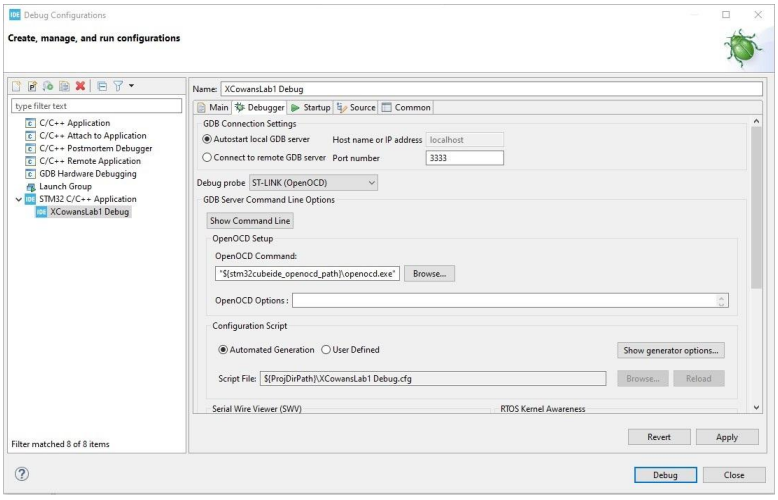
f. Create/Update OpenOCD debug configuration
   i. Select Run > Debug Configurations
   ii. Select STM32 C/.. and click the “New launch configurations”




1. Alternatively, you can also right-click STM32 C/C++ Application and select “New Configuration”

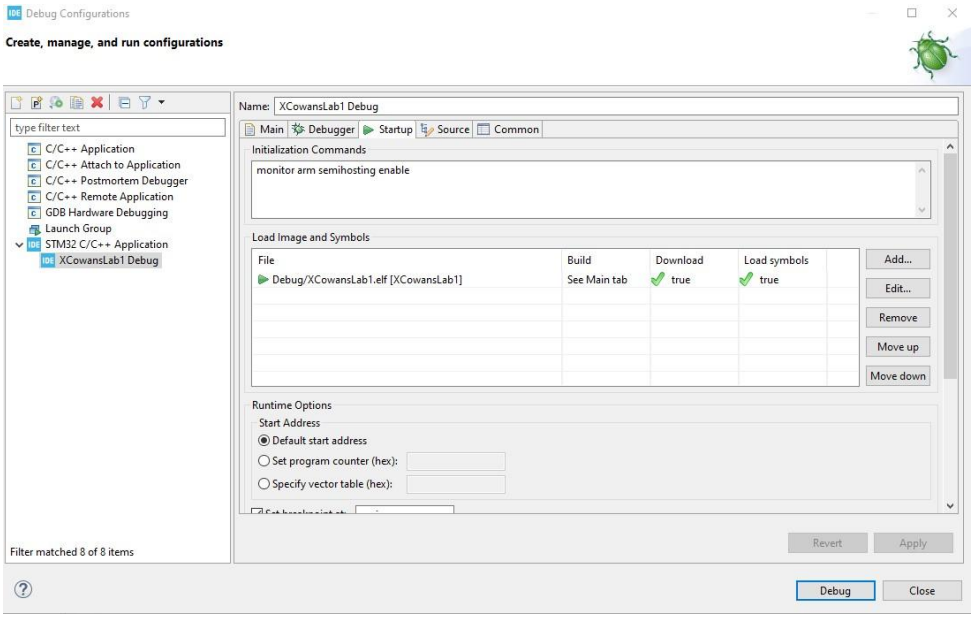
iii. In the Debugger tab, change “Debug Probe” to ST-Link (OpenOCD)

1. *This communicates with the board on which debug module we want to use. Many boards, like this one, only have one debug probe available*




iv. In the “Startup” tab, in the Initialization Commands, add “monitor arm semihosting enable”

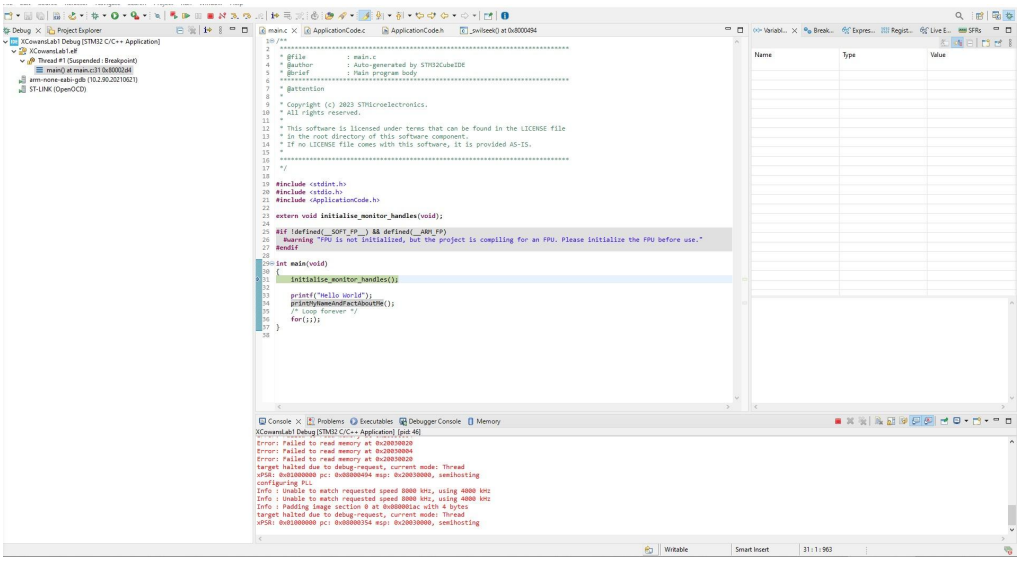
1. *This tells the debug session (GDB) to use semihosting*




v. Click Apply
vi. Click Close

5. Add the application code
    a. Using the IDE create an ApplicationCode.* fileset
        i. Create an ApplicationCode.c in the Src directory
            1. Right-click the Src directory > New > Source File
            2. In the Source File field, enter “ApplicationCode.c”
            3. Verify the template is still “Default C source template”
            4. Click Finish
        ii. Create an ApplicationCode.h in the Inc directory
            1. Right-click the Inc directory > New > Header File
            2. In the Header File field, enter “ApplicationCode.h”
            3. Verify the template is still “Default C header template”
            4. Click Finish
    b. Include the ApplicationCode header in the ApplicationCode.c
    c. Create a function prototype in ApplicationCode.h
        i. Name the function “printMyNameAndFactAboutMe”
            1. This function does not take in parameters and won’t return anything
            2. *Usually, we will not dictate what you name your variables or functions, but sometimes we will require things to be a specific name*
    d. Create the function definition in ApplicationCode.c
        i. In the function “printMyNameAndFactAboutMe”
            1. Create a local array with your name
            2. Create a printf statement that states your name and 1 exciting thing about you
                a. This should be one print statement

        i. If needed, refresh on printf and how to print strings stored in variables.
    b. Don’t forget to terminate the string ;)
6. Add the appropriate code to main (before the forever loop)
    a. *We are adding this BEFORE the for loop so it only gets executed ONCE rather than every iteration fo the for-loop*
    b. Add a printf statement that says “Hello World”
        i. Don’t forget to terminate the string (Refresh on C strings if needed).
    c. Then add a function call to print your name and the fun fact
        i. Note - you may want to include some headers, or you will get errors or warnings
            1. Hint: Which standard library does C use for printf?
            2. Another Hint: how do we tell the compiler where “printMyNameAndFactAboutMe” is and what it should do?
7. Review the debugging tutorial if you haven't already
    a. There will be questions on your quiz related to the content presented in video
8. Build your code
9. Plug in your board (if it wasn’t already plugged in)
10. Download/Flash your code and debug using the debug configuration you created earlier
    a. Run > Debug Configurations
    b. Select the debug configuration you want
        i. The most recent debug configuration you created should suffice
    c. Click Debug
    d. It will ask you to switch perspectives, click the check box regarding it remembering your decision, and select “switch”
    e. A lot of red text should fill the console window, and your screen should look like this:



    f. Select the play icon 
    g. Verify your print statements are present.

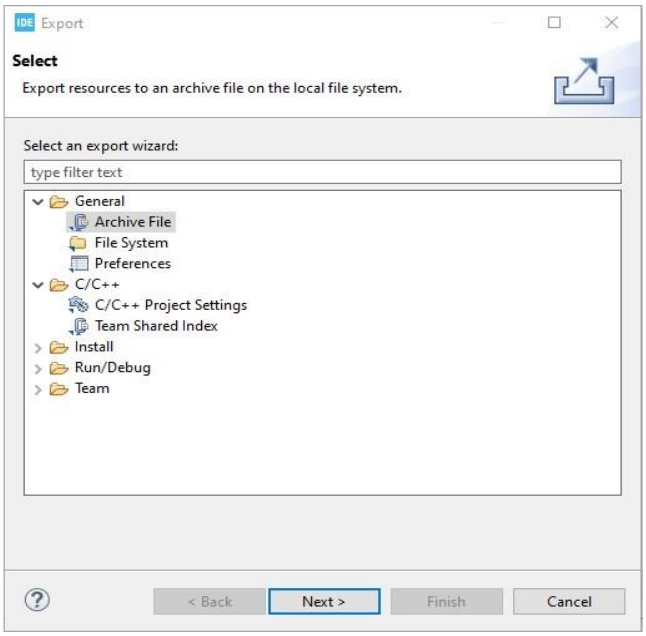
i. If nothing is printing, verify you terminated your strings when using printf..

h. Take a screenshot of your printed statement and turn it in with the post lab questions (Quiz 1)!

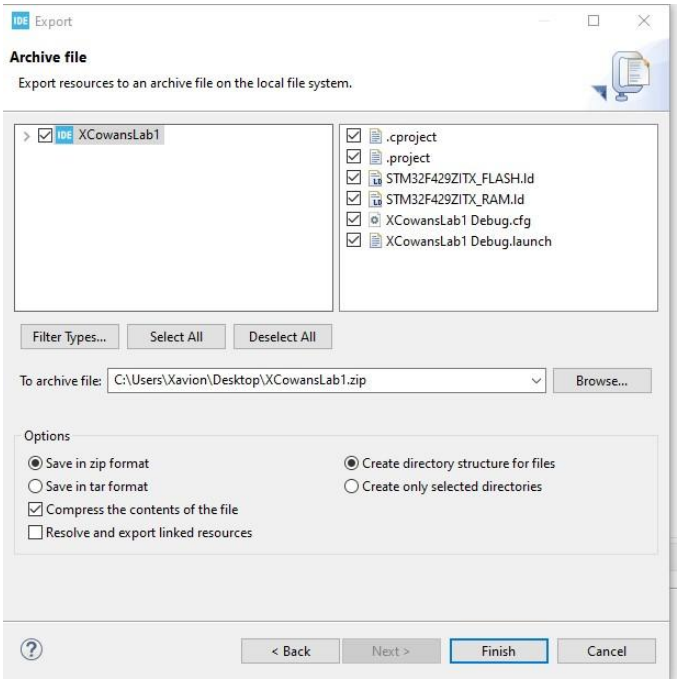
i. Restart the debug section and play around with some debugging concepts
   i. Verify you watched the debug tricks and tips video on Canvas
   ii. Set a breakpoint in code
   iii. Use the step-over functionality
   iv. Use the step-through functionality

j. Hit the red square to exit the debug session

11. Export the project
    a. Right-click the project and go to Export > Select General > Archive File
       *i. Archive file can also be seen as a zip file*




    b. Make sure the project and the contents in the right box are selected.
       i. Provide a path and filename for the export, this should be the same name as your lab and follow proper naming conventions
       ii. Make sure to save it as a zip and Select ‘Compress the contents of the file’
       iii. Select ‘Create directory structure for file’




iv. Click finish

12. Join the Slack workspace. Go to the announcement section of Canvas to see the URL needed to join the Slack link
    a. Update your name to your preferred name
    b. Update your picture to a somewhat recent (and somewhat professional) picture of you
13. Turn in the project (zip file) into Canvas by the due date.

Acceptance Criteria:

- The code compiles
- The console shows the student's name and a fun fact about them
- Student joined the slack channel with a work-appropriate photo of them

Grading Rubric:

This lab will be worth 100 points. The breakdown of grading will be given below.

1. Code Compilation (20 points)
    a. Full credit - Code Compiles with 0 errors and 0 code warnings
    b. Partial Credit - Code contains warnings (-10 points for each warning)
    c. No credit - Code does not compile (Student has at least one error)
2. Code Standards and Hierarchy (20 points)
    a. Proper naming of functions/files
    b. Proper layering of files
    c. For each Coding Standard violation, 5 points will get subtracted from the 20 points possible
3. Code functionality (30 points)
    a. Does the code print the student's name and a fact about them?

4. Class Involvement (10 points)
    a. Did the student join the Slack with an appropriate picture of them?
5. Project Exporting (10 points)
    a. Was the project exported right with the appropriate naming?
6. Lab Attendance (10 points)
    a. Did the student attend lab sessions appropriately and consistently