# IMAT3904 – Lab2

## Creating the program

Clone the lab2 repro using the link here

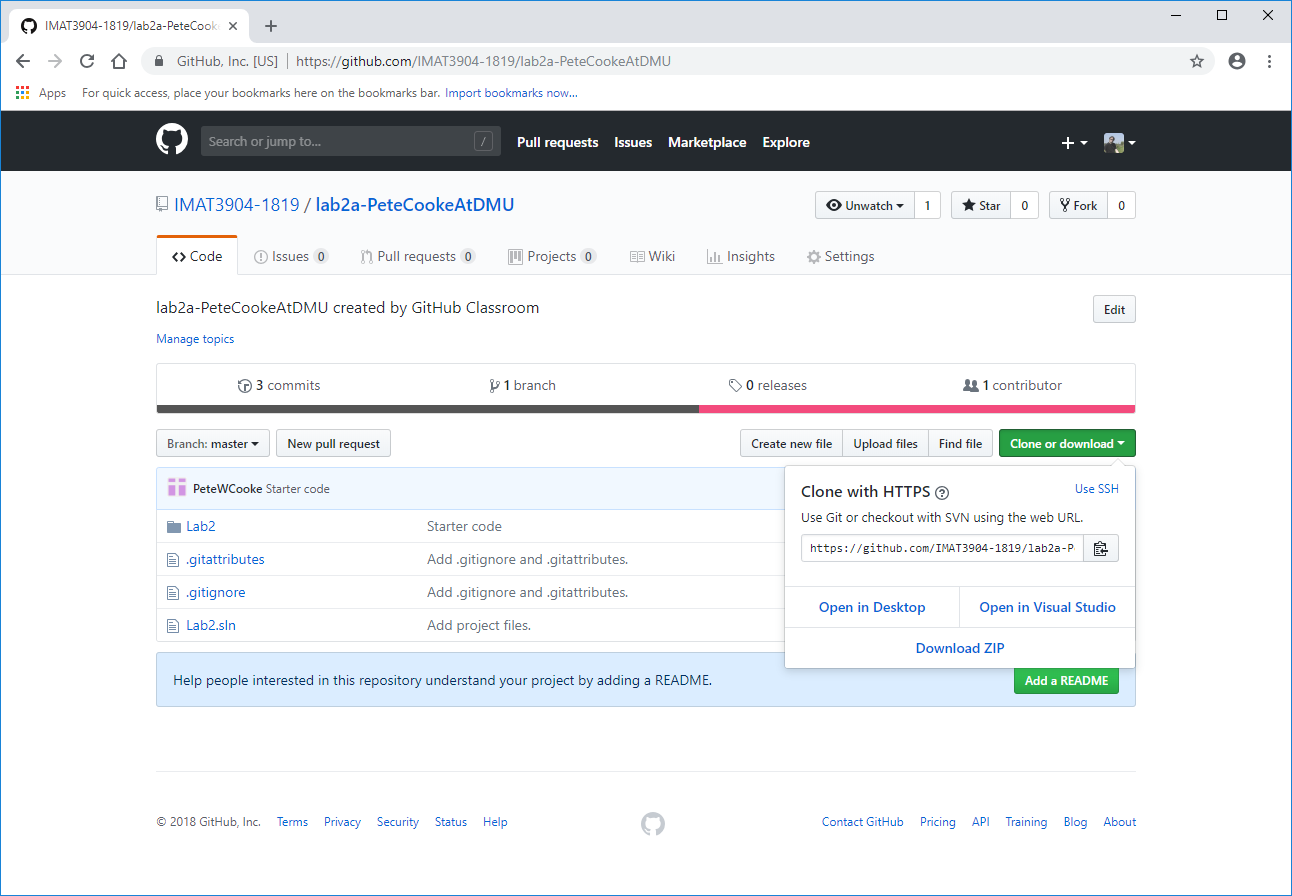
<https://classroom.github.com/a/YtO9S4LC>

(Old link was <https://classroom.github.com/a/aXxBJvmq> - missing .sln)

This is a simple 32bit Hello World console program, the only function of the clone is to create your repro on github classroom.

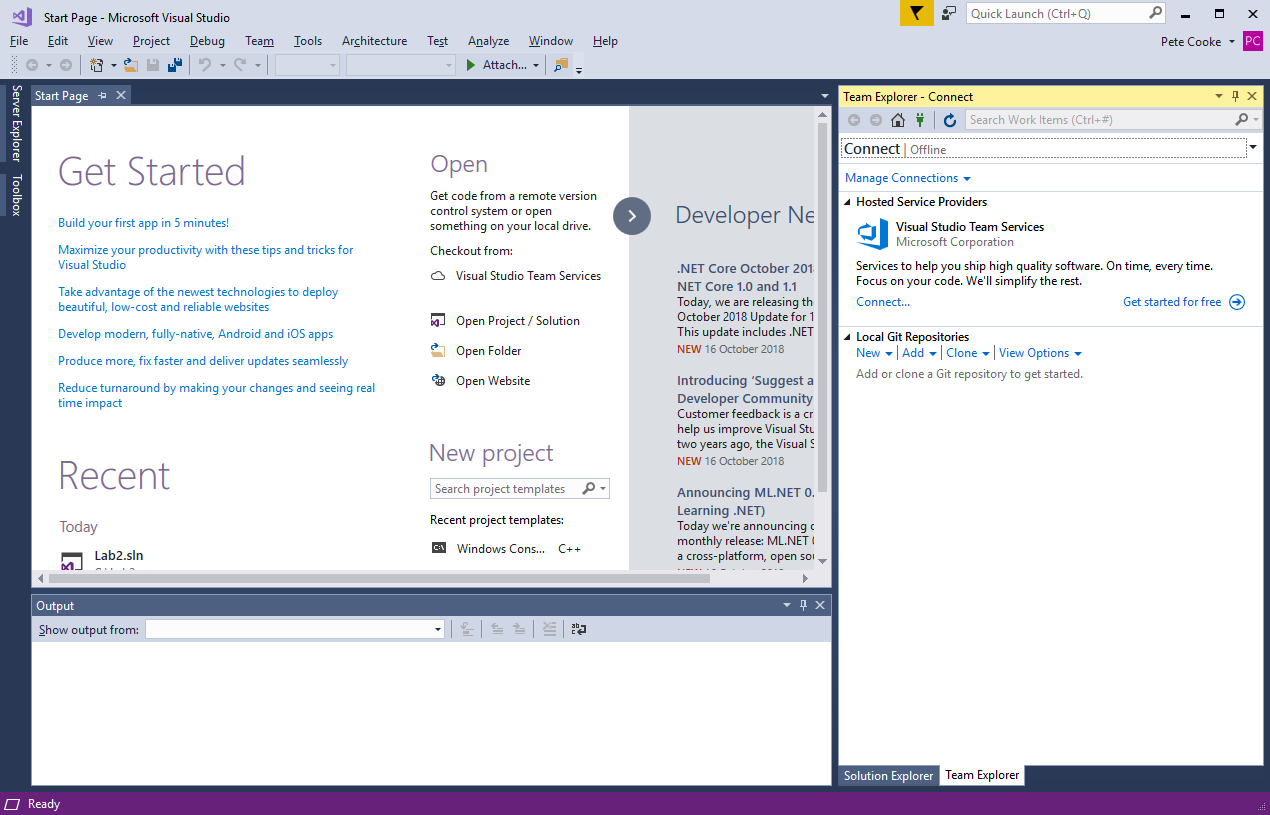
Paste this link into Chrome (remember IE doesn’t work with GITHUB) and follow the instructions to create your copy of the repro in Github Classrom.

Now view the repository online and click Clone or Download



Grab the URL <https://github.com/IMAT3904-1819/lab2a-????.git> (down’t download the .zip as you won’t be connected to the repository).

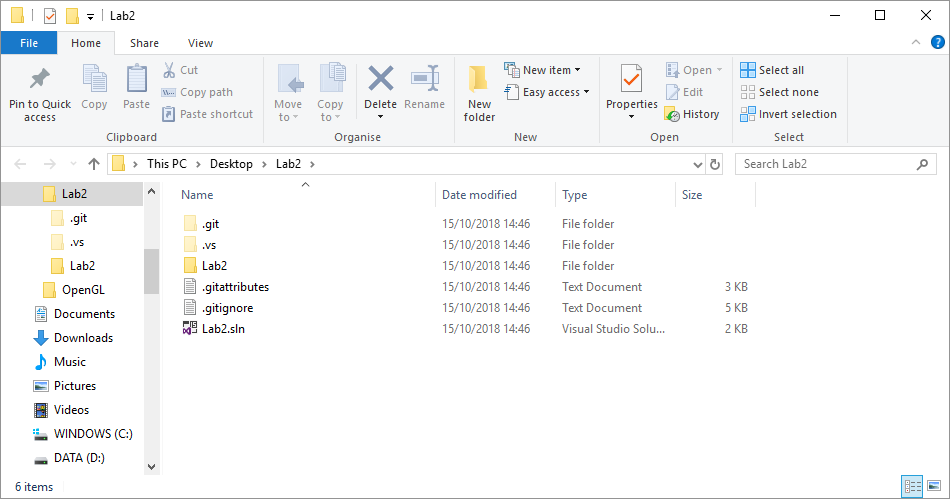
Launch VS2017 and go to team explorer.



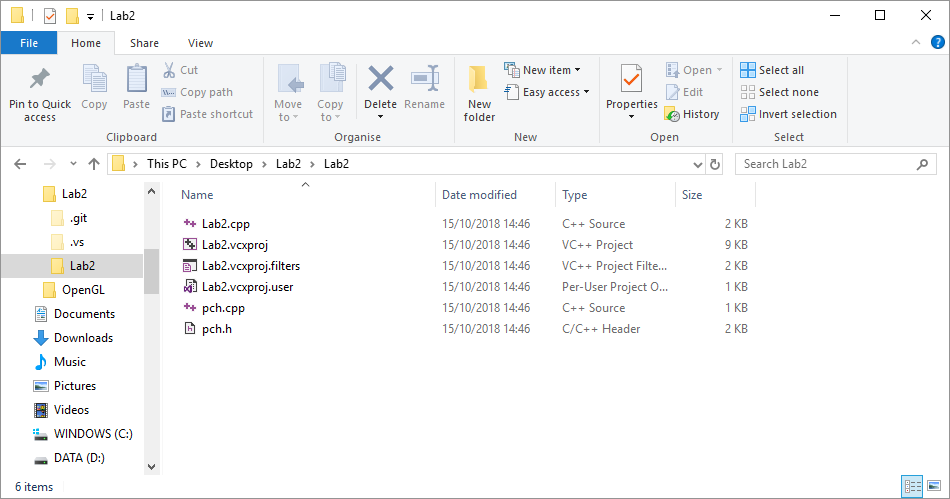
In the Connect menu click clone. Copy the github classroom URL in, choose a folder to store the cloned repo locally and click clone. Finally double click the .sln file to open the solution.

## Examining Visual Studio’s Folder Structure

Right click on the solution and select Open Folder in File Explorer, you should see a directory structure similar to the one below



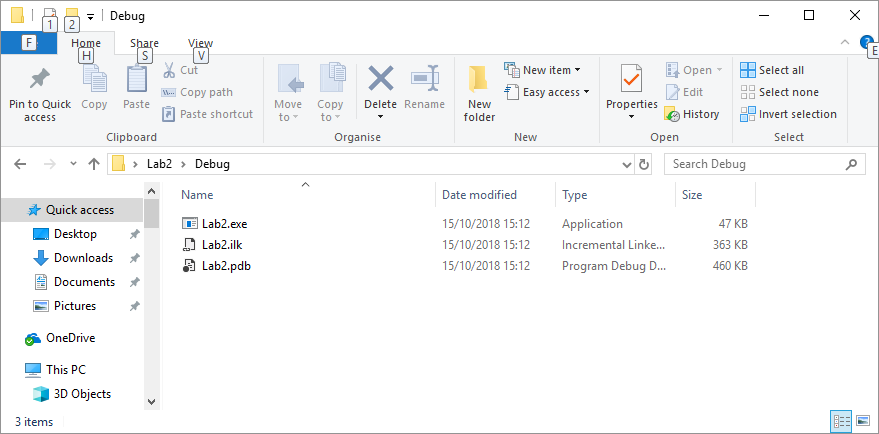
If we open up the Lab2 sub-folder we can see the source, include and project files…



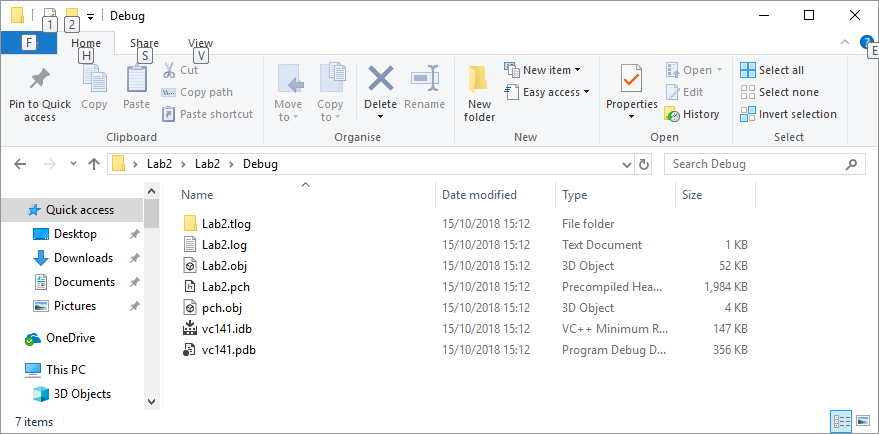
Switch back to Visual Studio and look at the files in Solution Explorer in the source and header folders and you can see that these folders in Solution Explorer are virtual and don’t reflect the actual structure on disk.

## Where VS Stores executables and Intermediate files

Make sure the project solution configuration is Debug and the platform is x86 (32 bit) and build the project. You should find that VS has created two Debug folders. In the project’s root directory is a Debug folder with three files, the executable, and linker file and the .pbd file that contains debug information.



In the Lab2 sub-folder is another Debug folder with various files including a .log file and two .obj files, one for each compiled .cpp file.



Switch to a 32bit Release version, build the project and note where the .exe and intermediate files are stored now.

Finally switch to x64 for a 64bit executable and build Debug and Release versions to see the Directories vs creates and where the .exe and intermediate files are stored now (note the x64 folders created).

Now switch back to a x86 Debug configuration for the rest of the lab.

## Adding more structure for larger projects

For a larger project is makes sense to add more structure than VS provides automatically.

Switch to the Lab2 sub-folder in explorer and create src, include, and lib folders. In the project root directory create a bin folder to hold the .exe

We need to move the cpp and .h files into there respective new folders. Unfortunately VS doesn’t provide an option to do this via the UI, we will need to edit the vs config file, an XML text file, ourselves.

First close the visual studio project.

Move the .cpp files to the src folder and the .h file to the include folder using Explorer

Now we need to let VS know about these changes. Open the file Lab2.vcxprj in the Lab2 sub-folder in Notepad++, find the references to the cpp and h files and change the path to add src\ or include\ as shown below.

<ItemGroup>

<ClInclude Include="stdafx.h" />

<ClInclude Include="targetver.h" />

</ItemGroup>

<ItemGroup>

<ClCompile Include="Lab2.cpp" />

<ClCompile Include="stdafx.cpp">

<PrecompiledHeader Condition="'$(Configuration)|$(Platform)'=='Debug|Win32'">Create</PrecompiledHeader>

<PrecompiledHeader Condition="'$(Configuration)|$(Platform)'=='Debug|x64'">Create</PrecompiledHeader>

<PrecompiledHeader Condition="'$(Configuration)|$(Platform)'=='Release|Win32'">Create</PrecompiledHeader>

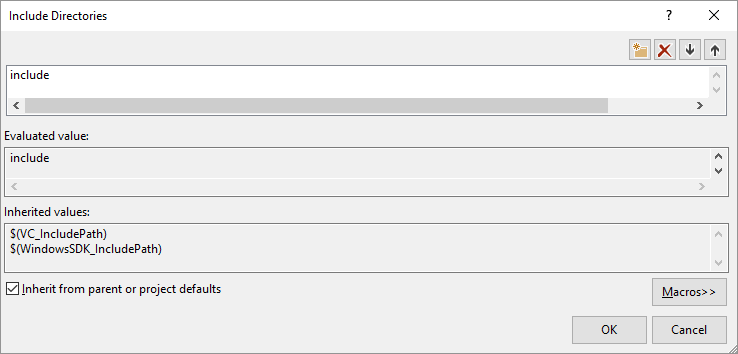
<PrecompiledHeader Condition="'$(Configuration)|$(Platform)'=='Release|x64'">Create</PrecompiledHeader>

</ClCompile>

</ItemGroup>

Restart Visual Studio and all the files should be visible. If your code had precompiled headers then the project won’t build because the code can’t find the pch.h header file (in the include subfolder).

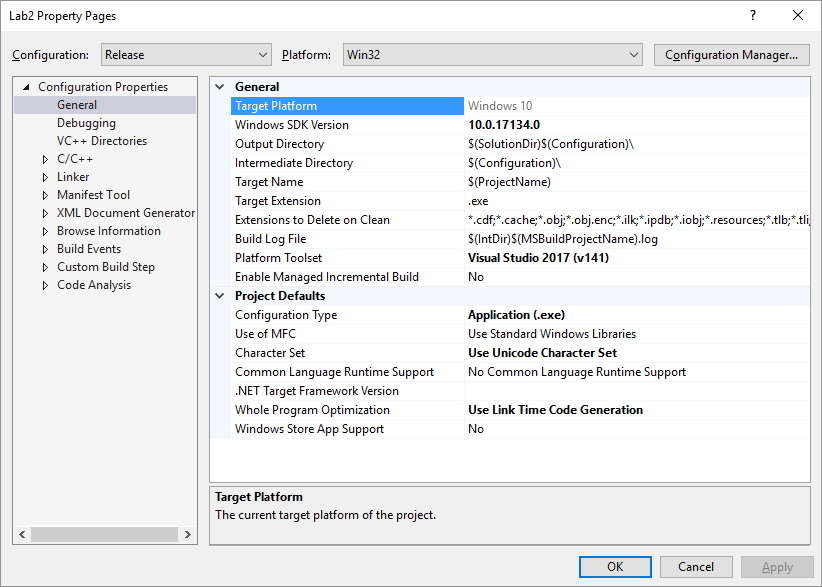
Go to Project Properties and under VC++ Directories add the include path to both Debug and Release configurations. Do the same for the scr and lib paths.



The Project should now build and run again.

## Looking at Project Directories and VS Macros

Open the included C++ project in Visual Studio and view the project properties. Look at the projects Configuration Properties->General



What is the value of $(SolutionDir)$(Configuration)\ for the Output Directory? Change this definition so that executables are written into the bin folder.

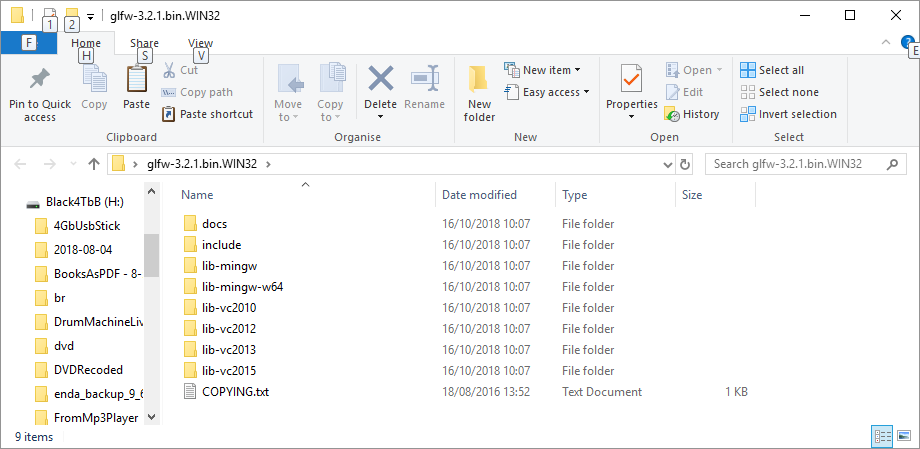
$(Platform) is another macro that can be used to differentiate between Win32 and x64 compiled programs. Change the Output Directory again for all combinations of Debug/Release and Win32/x64 and compile them to ensure they are indeed written to the correct folders. (Note: You can change all configurations at the same time by changing the Configuration and Platform drop-down boxes at the top.)

What is the meaning of the intermediate directory and where is it? What do the Target Name and Target Extension Lines mean? Change the intermediate directory to point to a ‘temp’ directory with separate places for each platform and configuration in the solution directory.

Under Configuration Properties->VC++ Directories, add an entry for Library Directories for of your own lib folders.

## Building a GLFW OpenGL project

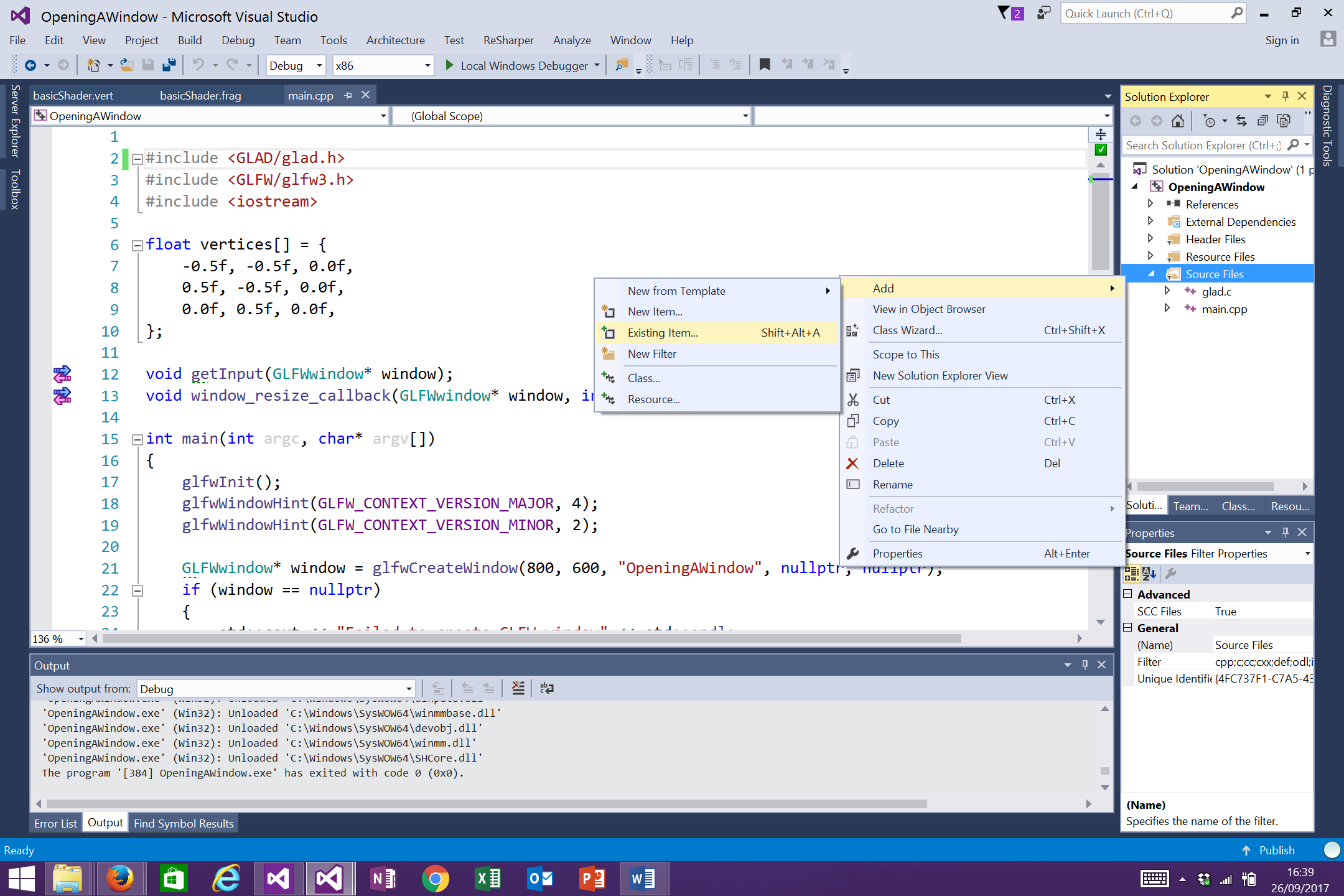
Download the pre-compiled binary version of the openGL framework GLFW from <http://www.glfw.org/download.html> . Note here that GLFW is open source and thus if you download the source code you are free to modify it and compile it yourself, just like any other project. To save time, the makers of GLFW have already compiled the library in the Win32 and x64 settings using a variety of different compilers – This is common courtesy and you will see this often. Download and unzip either version for now (or both). Work with 32 bits for now.

Inside the downloaded folder, you will see that the library has been compiled with the mingw compiler as well as multiple different visual studio compilers. This is because there are small differences amongst all the versions so you have to match the correct version to the compiler you are currently using. As of 15/10 there is no vc-2017 folder, but vc-2015 seems to work fine. There is only one version of the include folder as these are just header files and are the same for all versions. Go ahead and copy glfw.h into your own include/glfw folder and the appropriate glfw.lib into your lib folder.

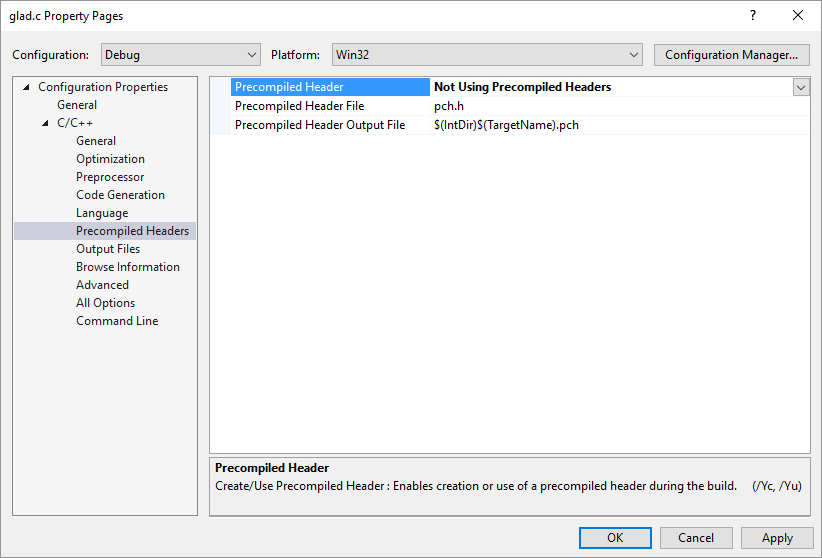
Link the required libraries under Project Properties -> Linker -> Input. The opengl32.lib is needed as well as glfw3.lib

We need some way of loading the correct opengl functions for our project. The modern shader approach to opengl requires the linking of a number of functions. Fortunately, this process has been simplified through a number of header libraries such as GLAD. Goto <http://glad.dav1d.de/> and ensure the gl version is set to 4.3 and the Profile to ‘Core’ (Note: you can get the latest and greatest graphics techniques with newer versions, but these won’t run on older machines and we won’t be using them anyway). At the bottom, ensure ‘Generate a loader’ is ticked and click ‘Generate’ to download the files.

Unzip the downloaded zip to a temporary folder. Now put the downloaded include folders in your own include folder (copy the whole Glad subfolder) and just copy the .c file to the project’s src folder. Right click on source folder and choose, Add-> Existing Item to add the file to the project.



You will need to turn off precompiled headers for the glad.c file as it is not cpp. Right click on the file and go to Properties ->c++ -> Precompiled headers and set ‘Not using precompiled headers’.



Copy (or type!) the following code into your lab2.cpp file to open up a simple opengl window and draw a green screen. If there are errors then it is likely you have not placed the libraries and includes in the correct places.

#include <GLAD/glad.h>

#include <GLFW/glfw3.h>

#include <iostream>

int main(int argc, char\* argv[])

{

// initialise a window and let GLFW know that it should target opengl version 4.3

glfwInit();

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 4);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

GLFWwindow\* window = glfwCreateWindow(800, 600, "OpeningAWindow", nullptr, nullptr);

if (window == nullptr)

{

std::cout << "Failed to create GLFW window" << std::endl;

glfwTerminate();

return -1;

}

// make this new window our current context, THEN try to initialise GLAD function ptrs

glfwMakeContextCurrent(window);

if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

{

std::cout << "Failed to initialise GLAD" << std::endl;

return -1;

}

// until we receive a message to close the program

while (!glfwWindowShouldClose(window))

{

// clear the screen to a green colour

glClearColor(0.2f, 0.5f, 0.3f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

// swap buffers i.e. draw to screen

glfwSwapBuffers(window);

glfwPollEvents();

}

// cleanup

glfwTerminate();

return 0;

}

Change the colour of the screen background. Commit the changes on your local git repo and then push the changes to the online repository.

If you receive the message LINK : warning LNK4098: defaultlib 'MSVCRT' conflicts with use of other libs; use /NODEFAULTLIB:library

Then go to Project Properties -> Linker -> Command Line and add

/NODEFAULTLIB:MSVCRT.lib

