|  |  |  |
| --- | --- | --- |
|  | **Rochester Institute of Technology**  **Golisano College of Computing and Information Sciences**  **School of Interactive Games and Media**  **2145 Golisano Hall – (585) 475-7680** |  |

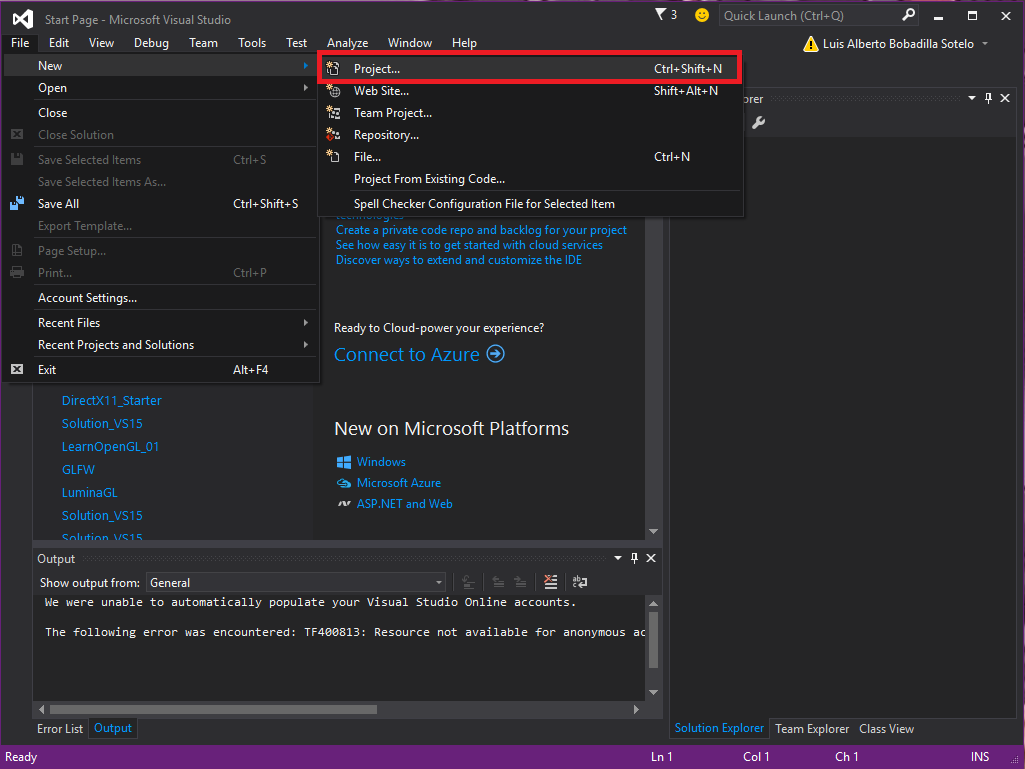
**Data Structures & Algorithms for Games & Simulation II**

**IGME 309**

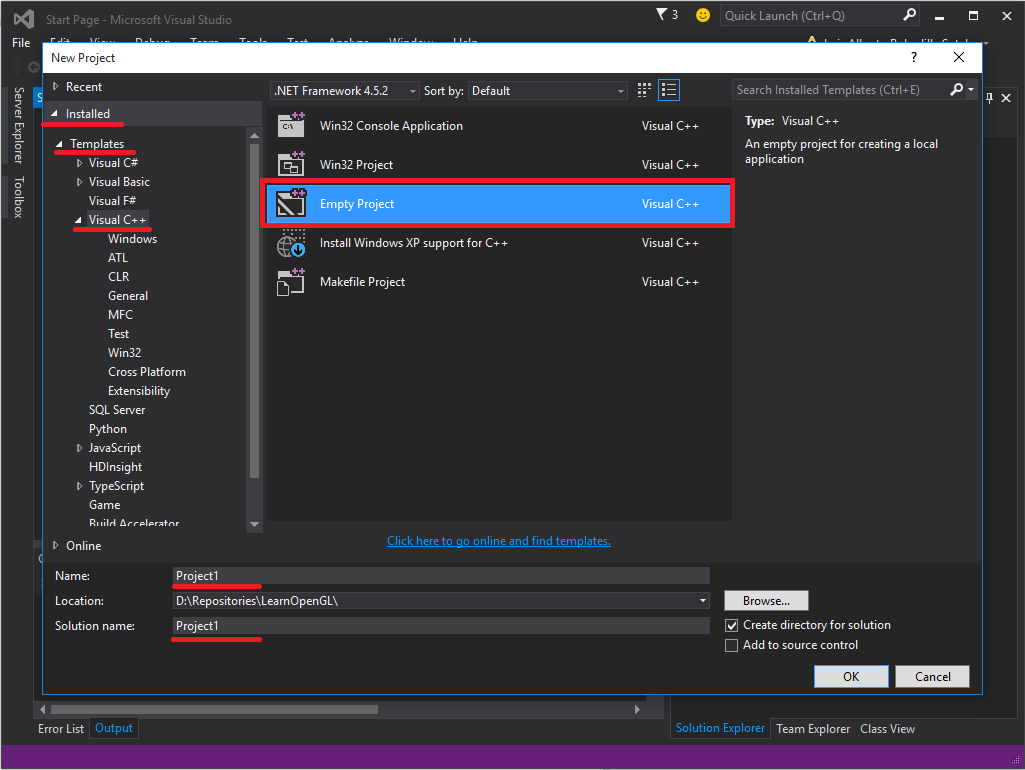
**E02: Visual Studio 2015 GLFW Configuration**

0) Instruction points 1 through 24 are the same as E01, you can just use an existing solution or start from scratch (if you do reuse a working solution please skip all the way to page 20).

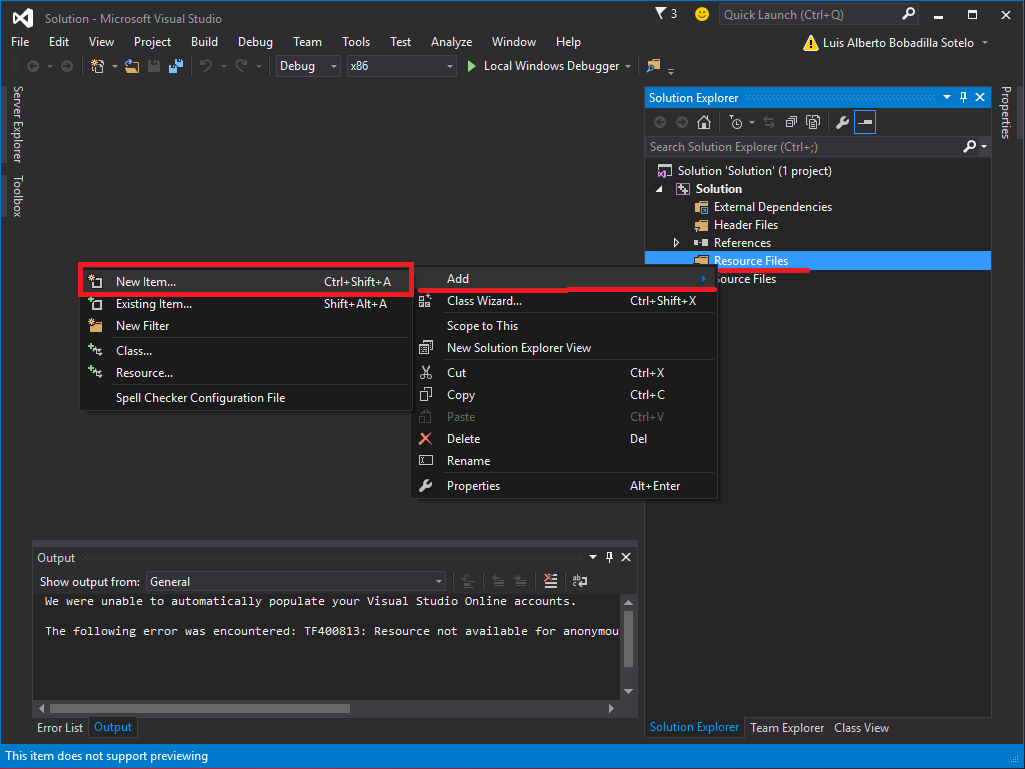
1) Open a New Visual Studio 2015 Window and create a new Project (FILE/New/Project…)



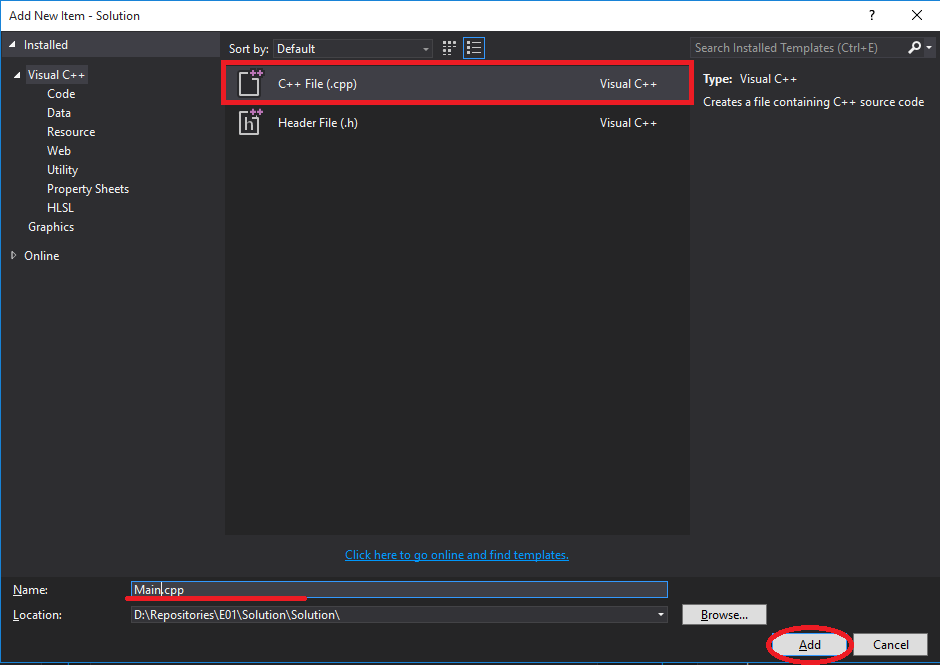
2) Create a new C++ Empty Project (Installed/Visual C++/ Empty Project), give it a name (like Solution) and a location and hit OK



3) Add a new C++ file IN THE SOURCE FOLDER (Source Files/Add/New Item…)



4) Name the file however you want (“Main” for the sake of this example) and hit OK



5) In Main.cpp add your main function and call your header folder, remember that #include “ ” and #include < > are different, the first refers to a file that is in one of the header folders local to your project and the second refers to the headers folder in Visual Studio installation directory (for this example just copy paste these lines)

/\*--------------------------------------------------------------------------------------------------

Comments about this program

--------------------------------------------------------------------------------------------------\*/

#include "Main.h"

int main()

{

std::cout << "Hello World\n";

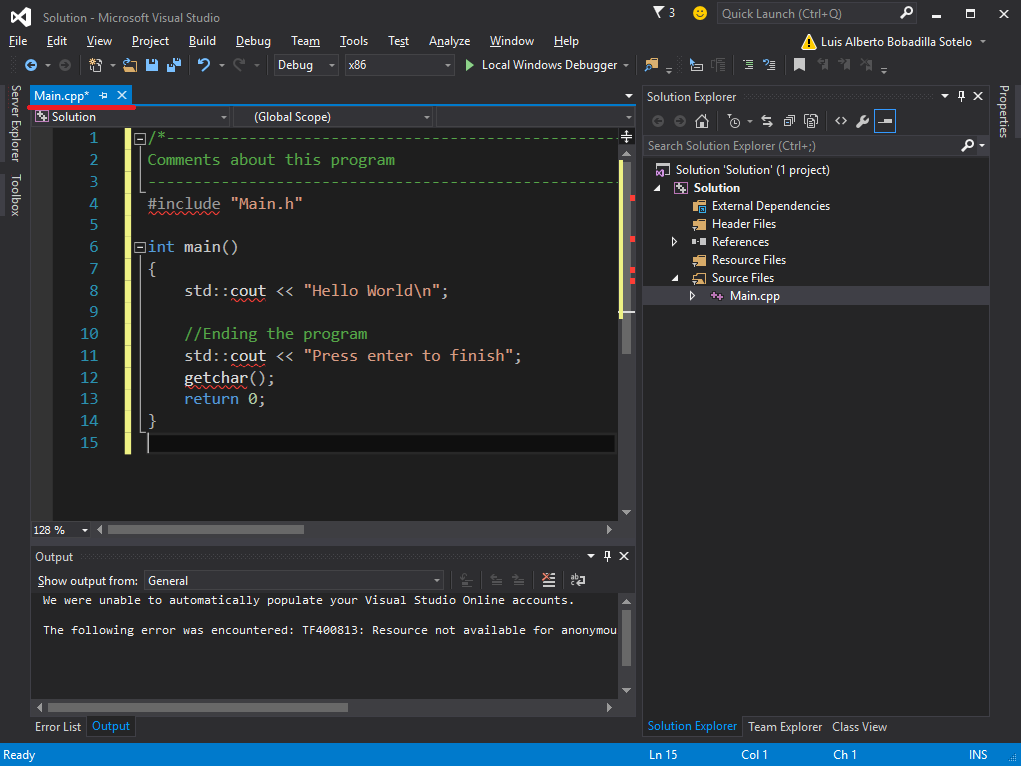
//Ending the program

std::cout << "Press enter to finish";

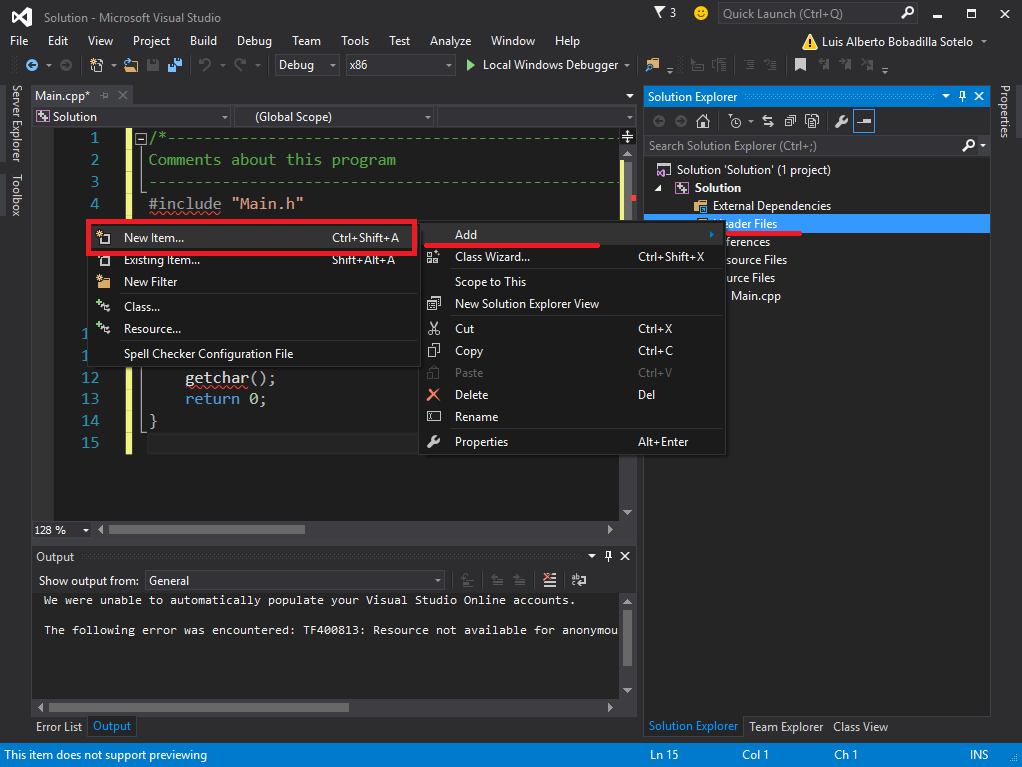
getchar();

return 0;

}

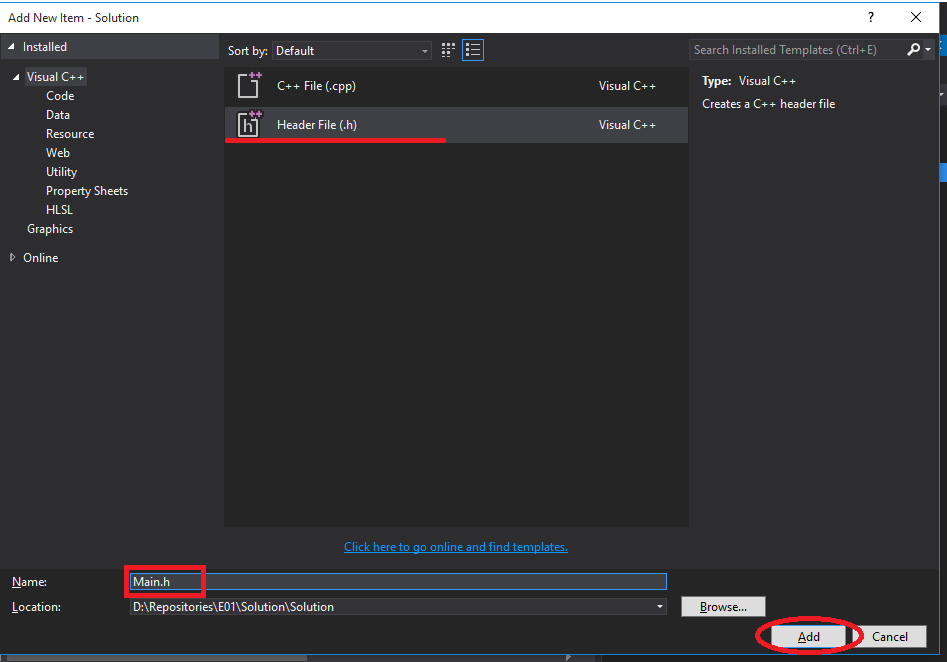


6) Add a new header file IN THE HEADERS FOLDER (Header Files/Add/New Item…)



7) Name the file however you want but remember to include the header from the cpp file

(Name it “Main” for the sake of this example) and hit OK



8) Create the protectors for your file, so the compiler don’t try to compile the file more than once. And include the headers that you will use in your project. (for this example just copy paste these lines, )

/\*--------------------------------------------------------------------------------------------------

This project was generated in 2013

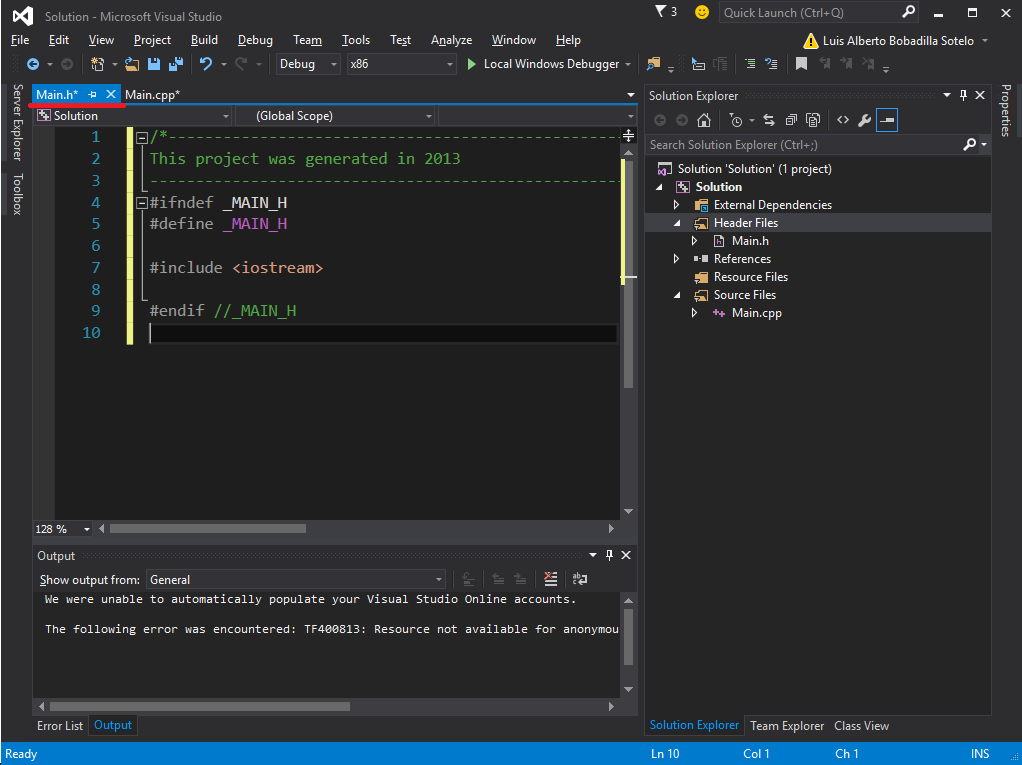
--------------------------------------------------------------------------------------------------\*/

#ifndef \_MAIN\_H

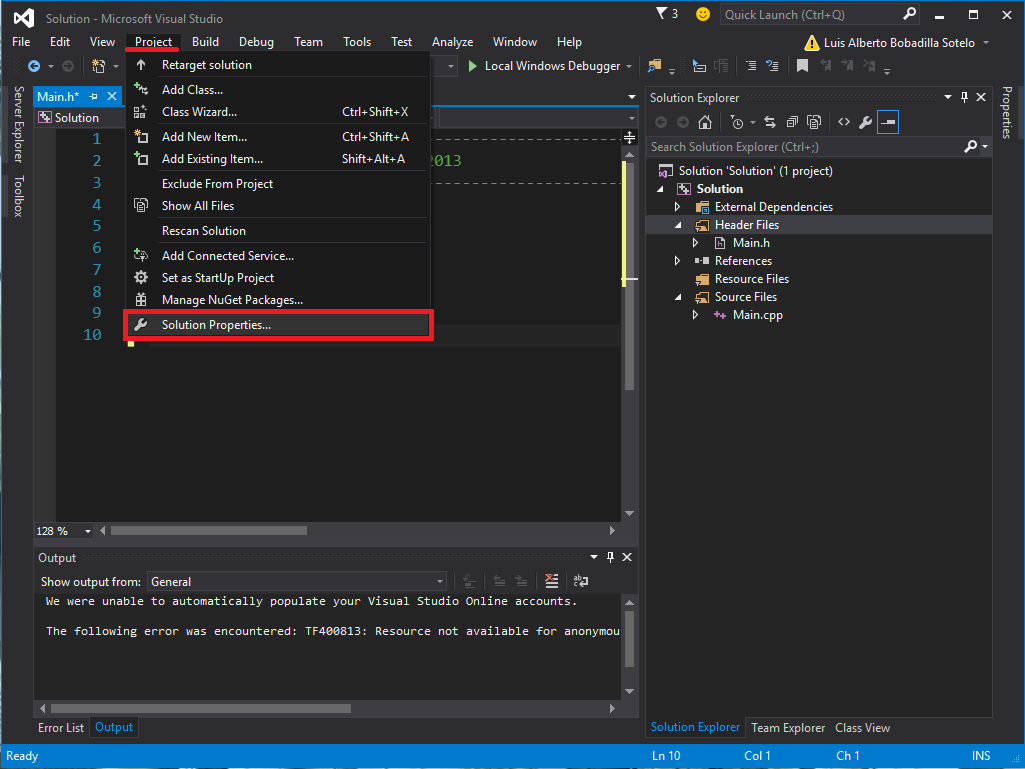
#define \_MAIN\_H

#include <iostream>

#endif //\_MAIN\_H



The difference with the #ifndef/#define/#endif and #pragma once is that #pragma is non-standard and it would just work right in most of the compilers but not in all, causing differences in the semantic, while #ifndef/#define/#endif is standard working in all compilers and its usually faster. If you don’t get what I tried to say just here… only stick with #ifndef/#define/#endif it’s faster in compilation anyway.

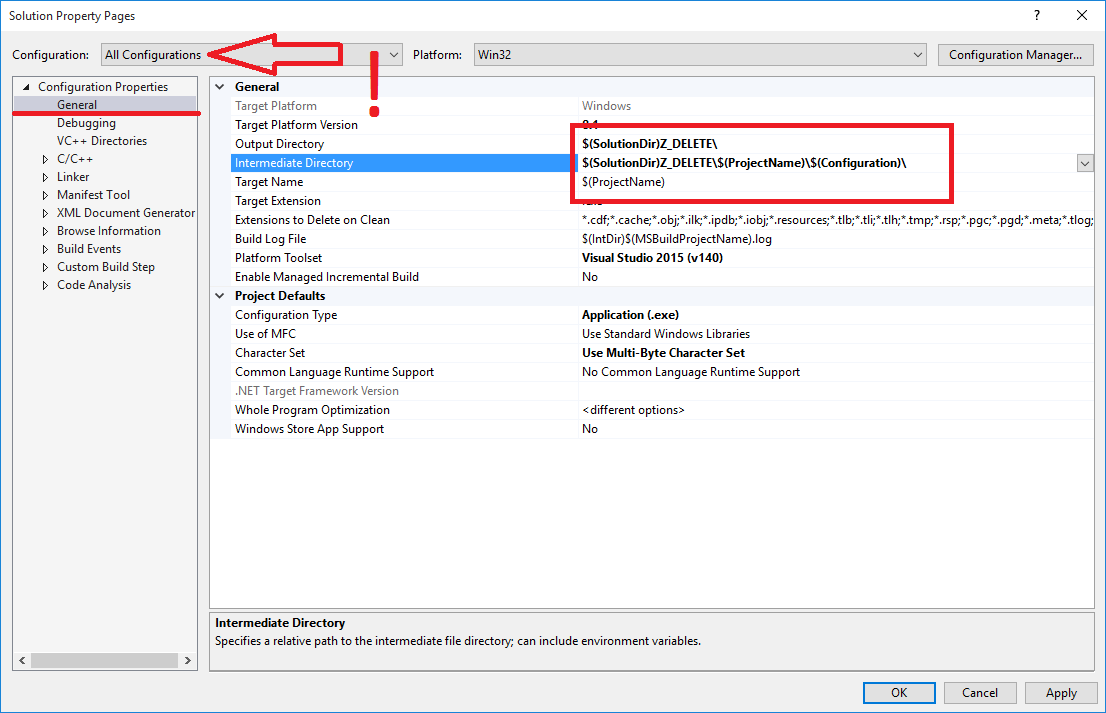
9) Go to your project properties. (PROJECT/[Solution] Properties…)

10) Click on Configuration Properties and make sure you are changing the settings for **all the configurations** go to General and change the Output Directory, the Intermediate Directory and the Target Name fields like follows and apply your changes:

Output Directory: $(SolutionDir)Z\_DELETE\

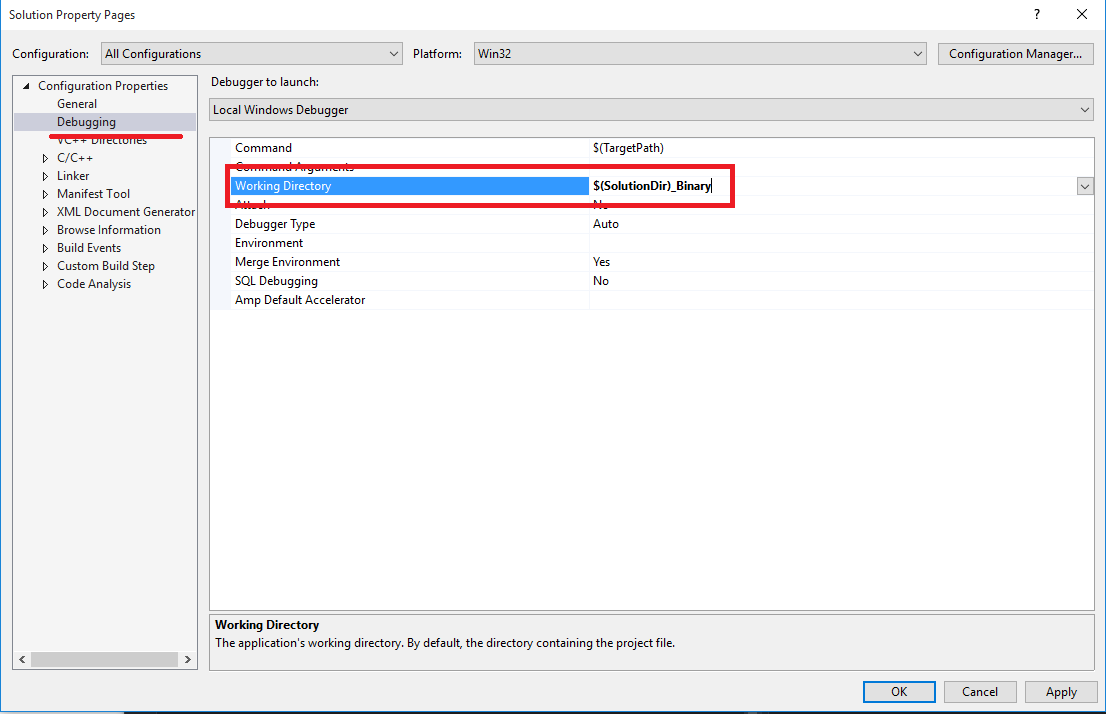
Intermediate Directory: $(SolutionDir)Z\_DELETE\$(ProjectName)\$(Configuration)\

Target Name: $(ProjectName)



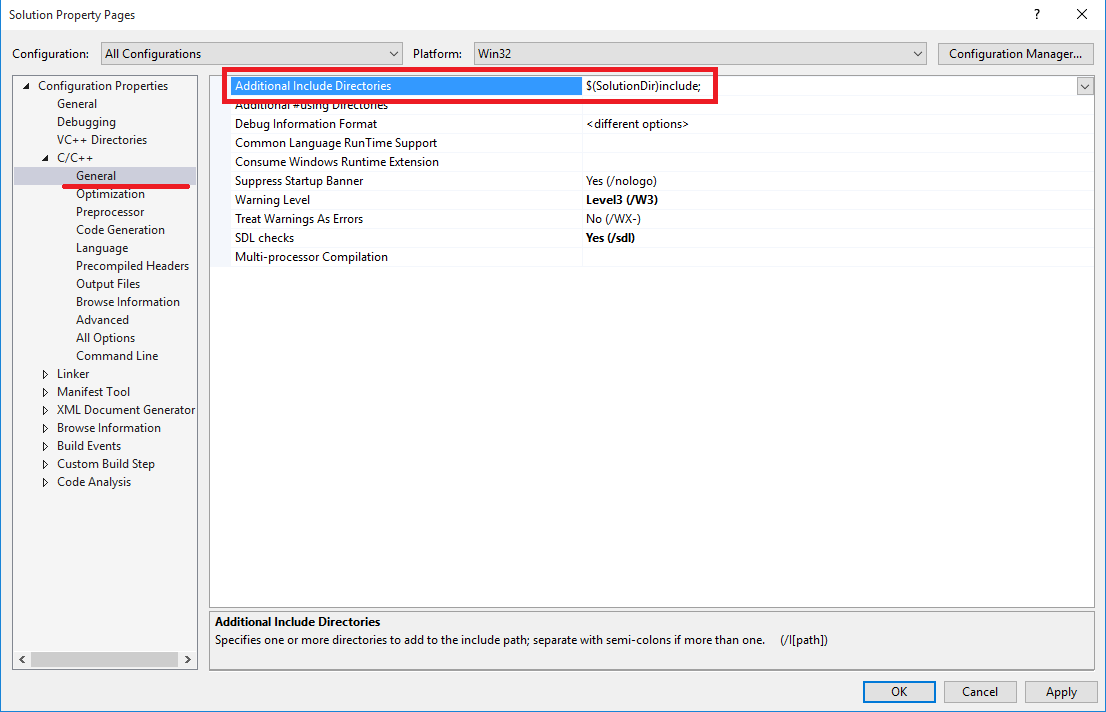
11) In Configuration Properties / Debugging / Working Directory change the field to:

$(SolutionDir)\_Binary

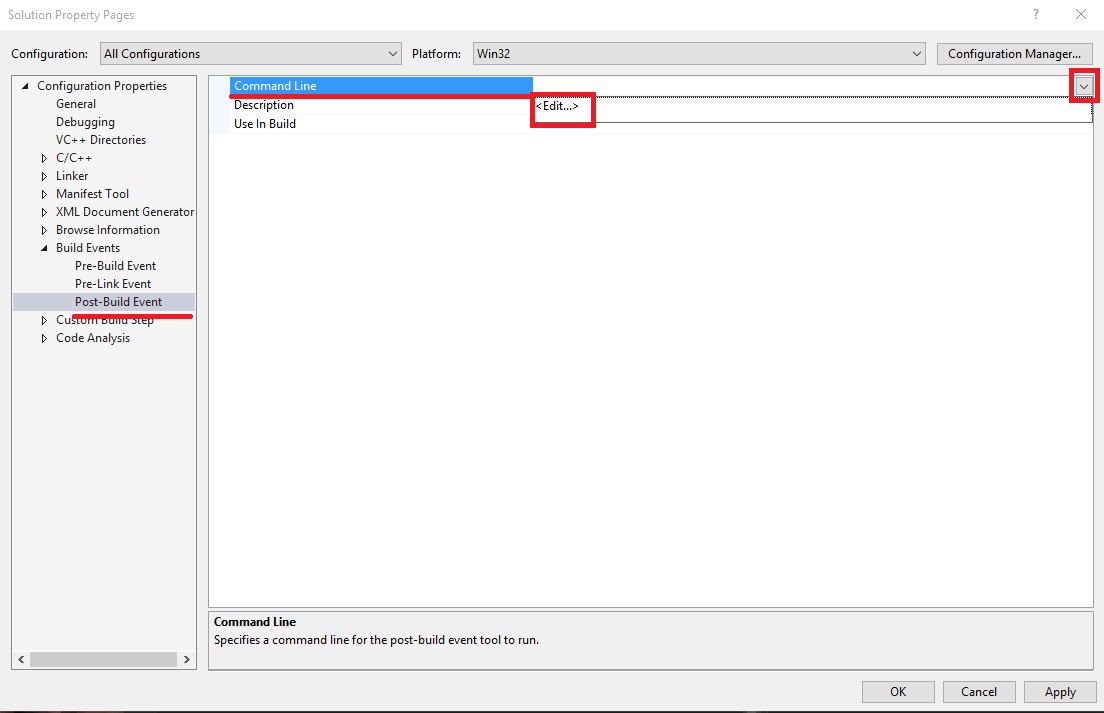


12) In C/C++ / General include the folder:

$(SolutionDir)include;



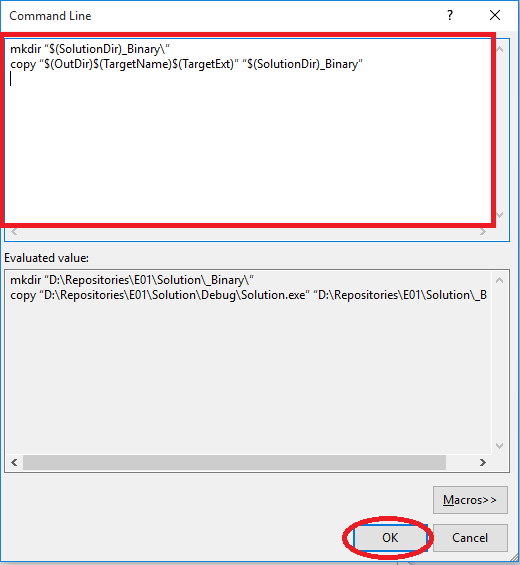
This way if you have more than one header file that is going to be used among various projects in your solution you don’t need to have extra copies of your headers.

13) In Build Events/ Post-Build Events/ Command Line click on the arrow and <edit…> 

Then add:

mkdir “$(SolutionDir)\_Binary\”

copy “$(OutDir)$(TargetName)$(TargetExt)” “$(SolutionDir)\_Binary”

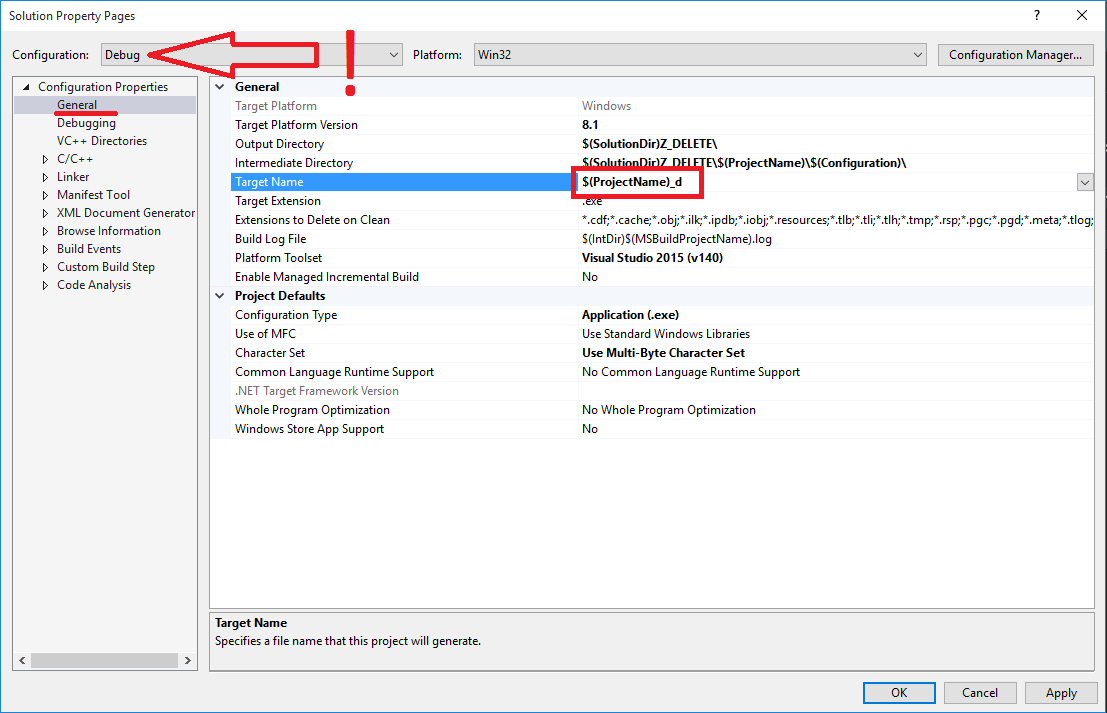


and in Buils Events/ Post-Build Events/ Description add:

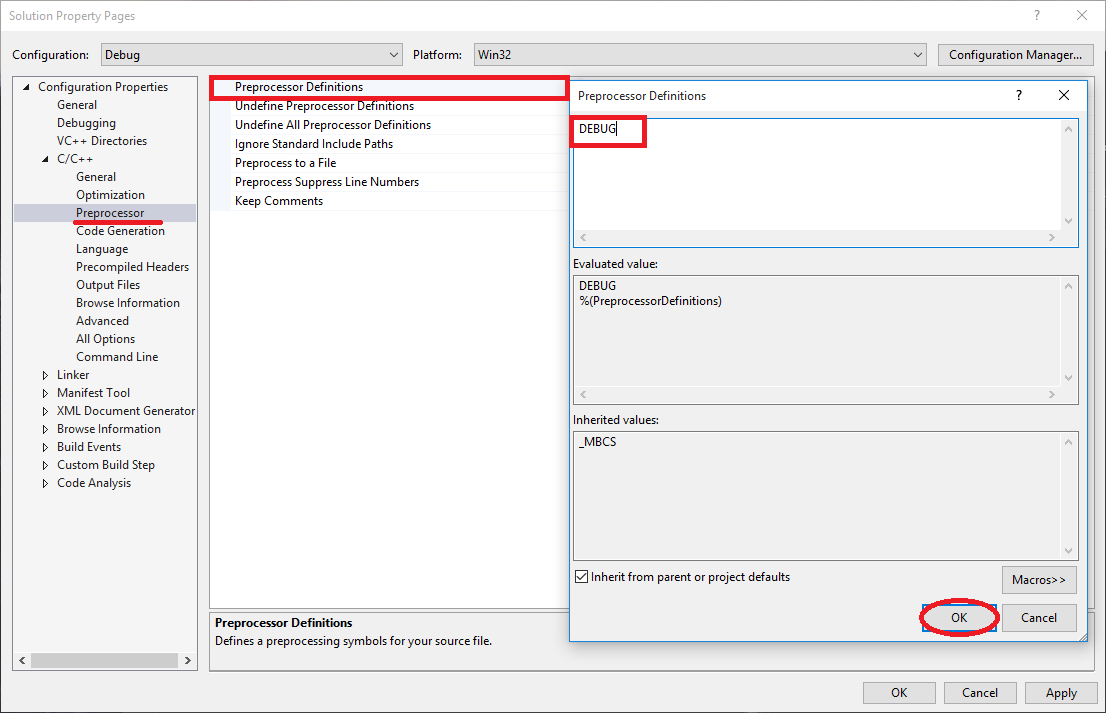
Copying binary…

***Apply your changes***

14) Change your configuration to **DEBUG** and in Configuration Properties / General add \_d at the end of your Target Name and apply the changes.



15) Still in Debug configuration in C/C++ / Preprocessor/ Preprocessor Definitions click on the arrow and add DEBUG

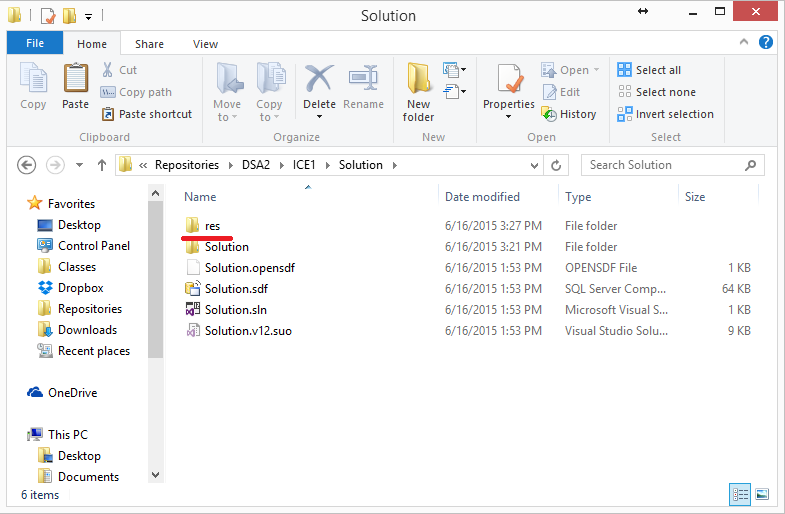


With this you are defining DEBUG as a keyword for your entire project.

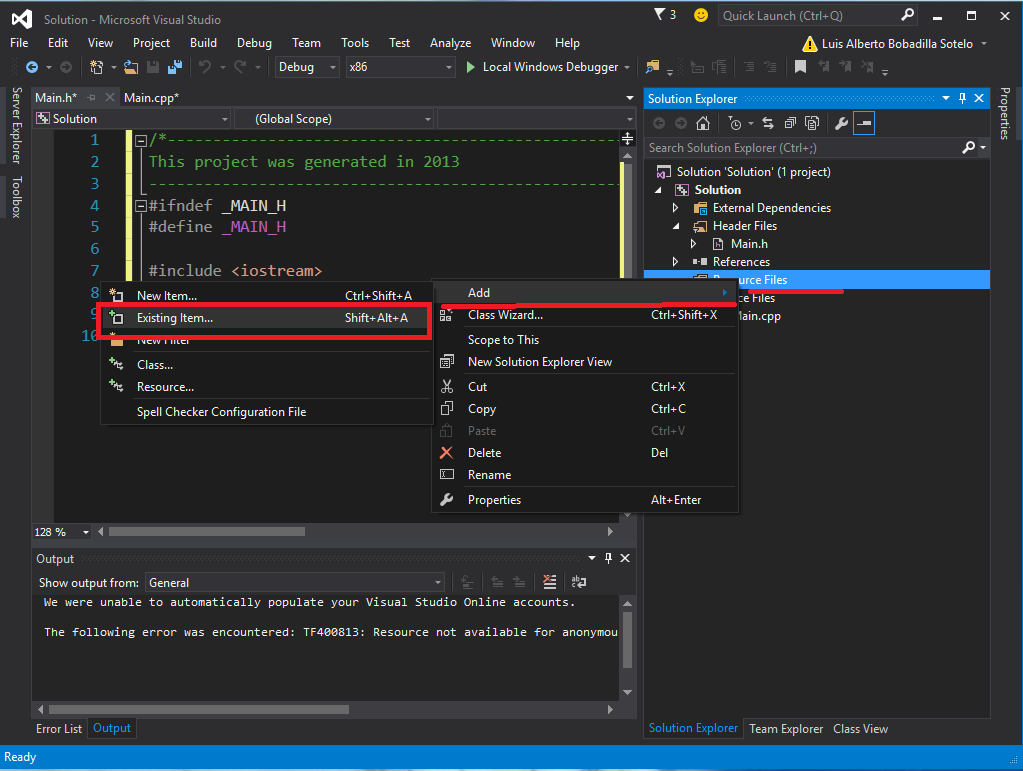
With this configuration you will add all the intermediary and binary files to a folder called Z\_DELETE at the same level as your solution and will copy the binary to a new folder called \_Binary.

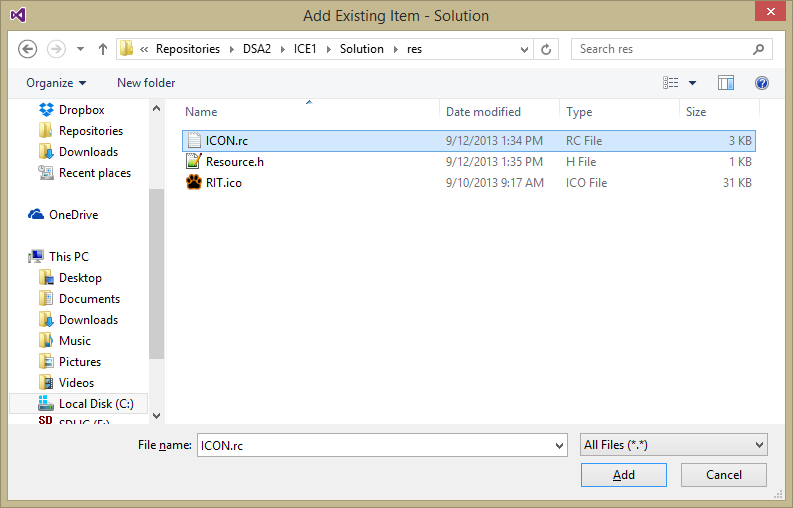
***Hit OK to close the configuration now***

16) Copy the provided res folder at the same level of your sln file:



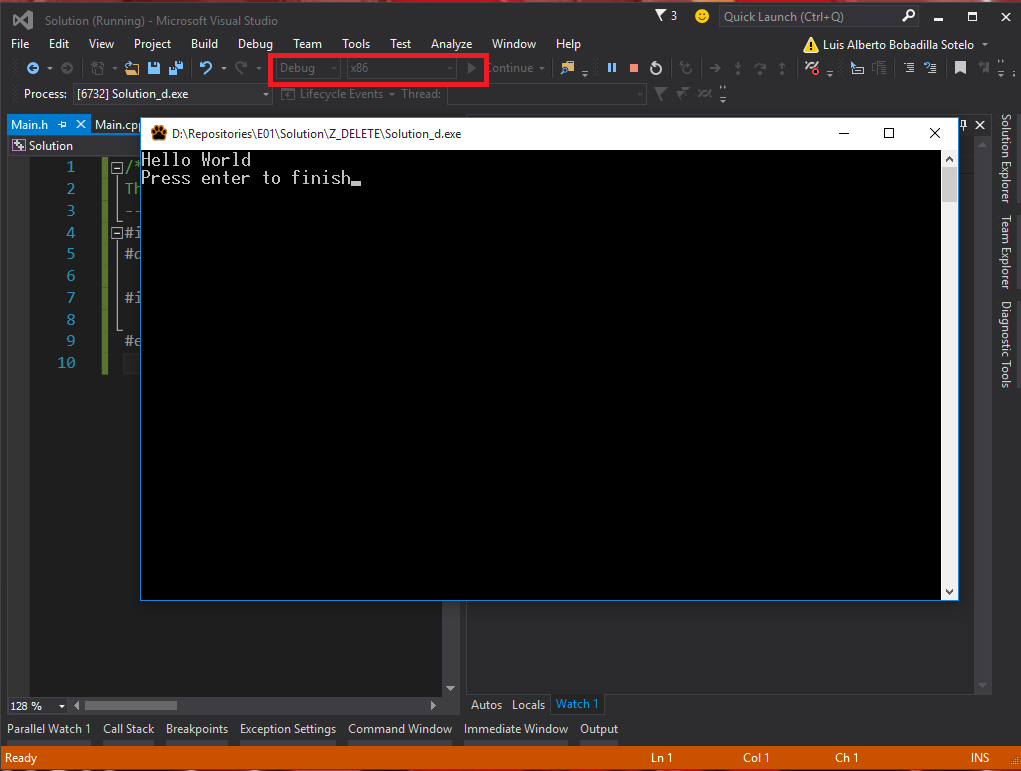
17) Add the ICON.rc into the RESOURCES FOLDER (right-click Resources Files/Add/Existing Item)





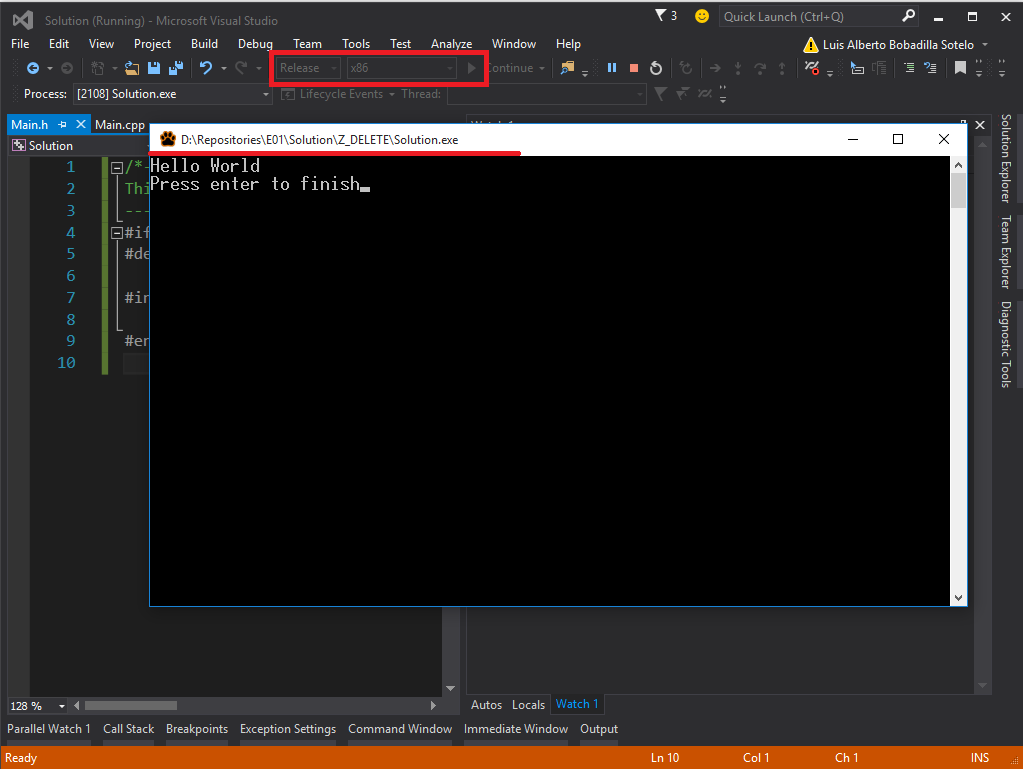
This will add an Icon to your application, if you ever want to add a different icon to your project you just need to specify said icon in the ICON.rc file accordingly (You can edit the file in Note Pad, or the icon directly in VS).

18) Run your project in Debug mode (with F5 only, not break-debugging it with Control+F5) and you should see something like this:



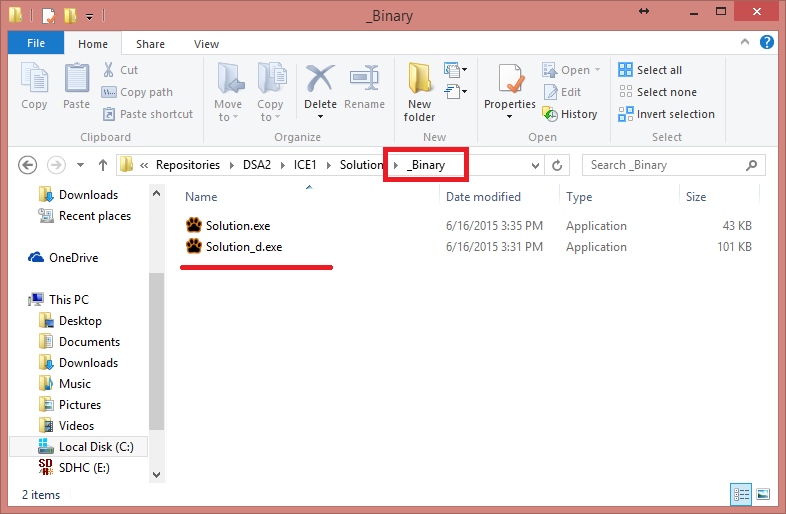
As you can see the name of the executable ends in \_d

19) Run your project in Release mode and you should see something like this:

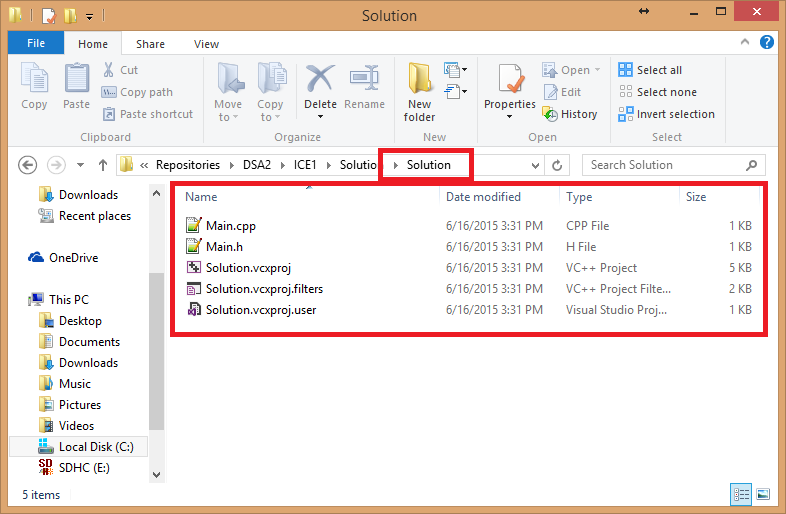


The executable should be missing the \_d now

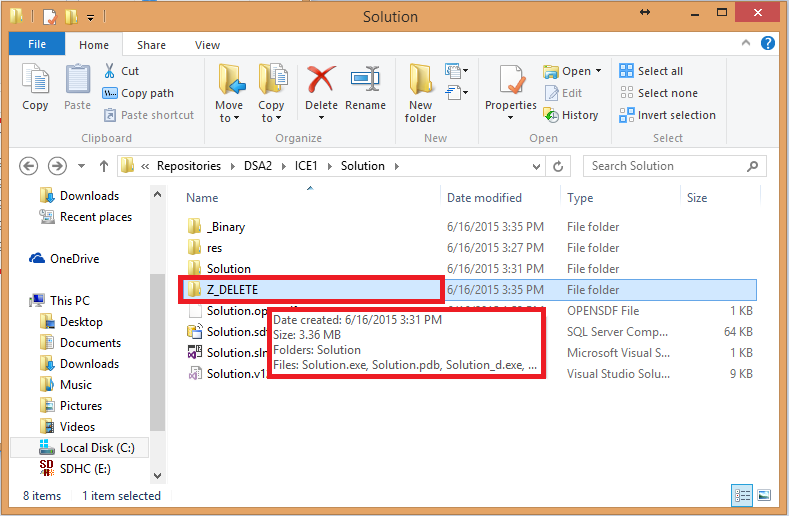
20) in your \_Binary folder you should only have two files:



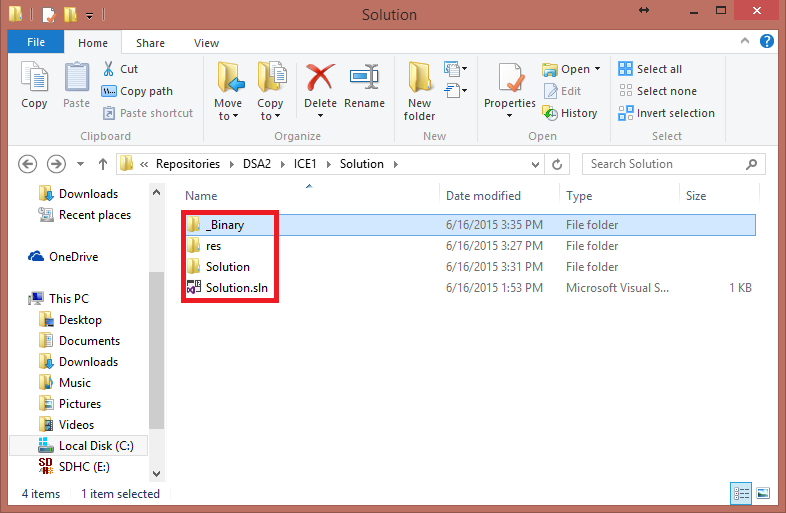
21) in your [Solution] Directory (where your c++s and headers are) you should have no subfolders:



22) In your main Solution folder you should have a Z\_DELETE folder (of about 3 MB)



23) Close visual studio and delete all the trash files:



24) You should only have a Solution.sln file and those 3 folders (and maybe a hidden .suo file do not worry about that file).

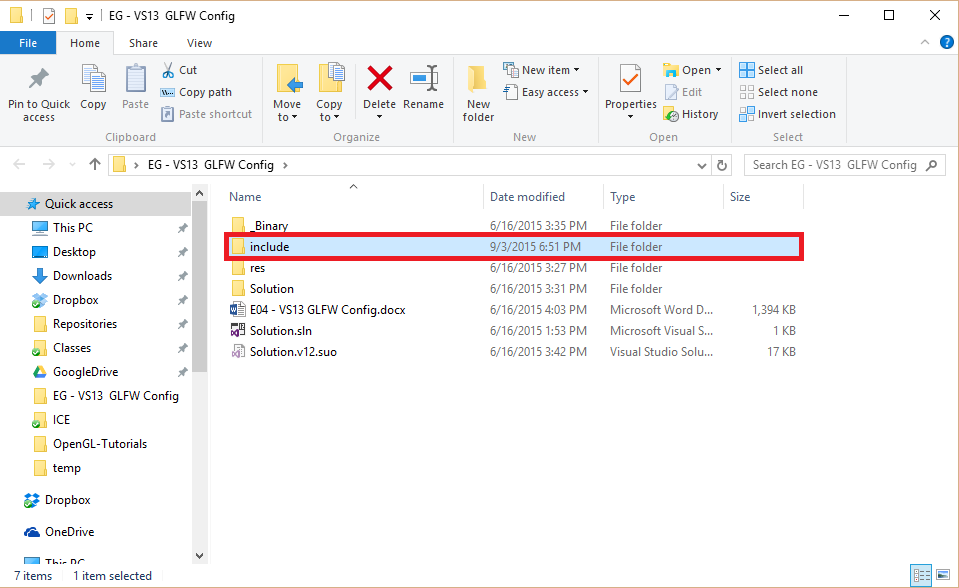
When you open an .sln file Visual studio is going to recreate the data base definition for your variables and functions (the .sdf file) this file is used inside your configuration for your machine only if you share this project the new computer will get rid of this file and create a new tailored one, thus making it completely useless for sharing. It also uses A LOT of space.

Inside of your Z\_DELETE folder visual studio will create the final binary and all of the intermediary files that are necessary in order to make those, again all these files are recreated by Visual Studio as needed, thus you are completely safe by just deleting this folder before you zip your project and share it.

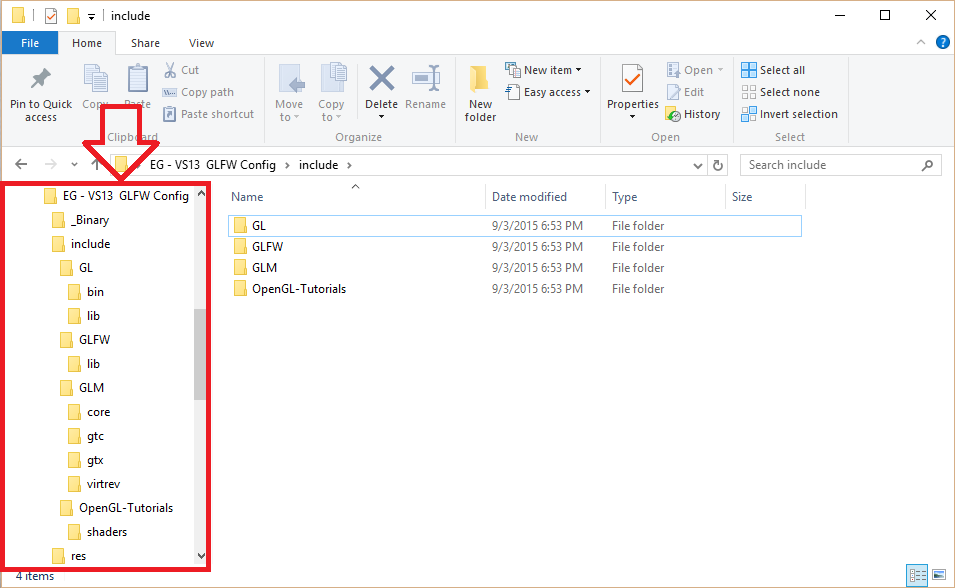
A zipped version of this project is going to use as much as 3 MB without all the “trash files” a project that has no “trash files” is roughly around 100KB, this is why GETTING RID OF ALL THE TRASH FILES IS A REQUIREMENT FOR ANY SUBMISSION FOR THIS CLASS.

NEW INSTRUCTIONS

25) In the root of your solution create an include folder:



26) Inside of that folder unzip the provided library files in such a way they follow the folder structure on the left:



27) in your main.h include the following after <iostream>:

// Include standard headers

#include <stdio.h>

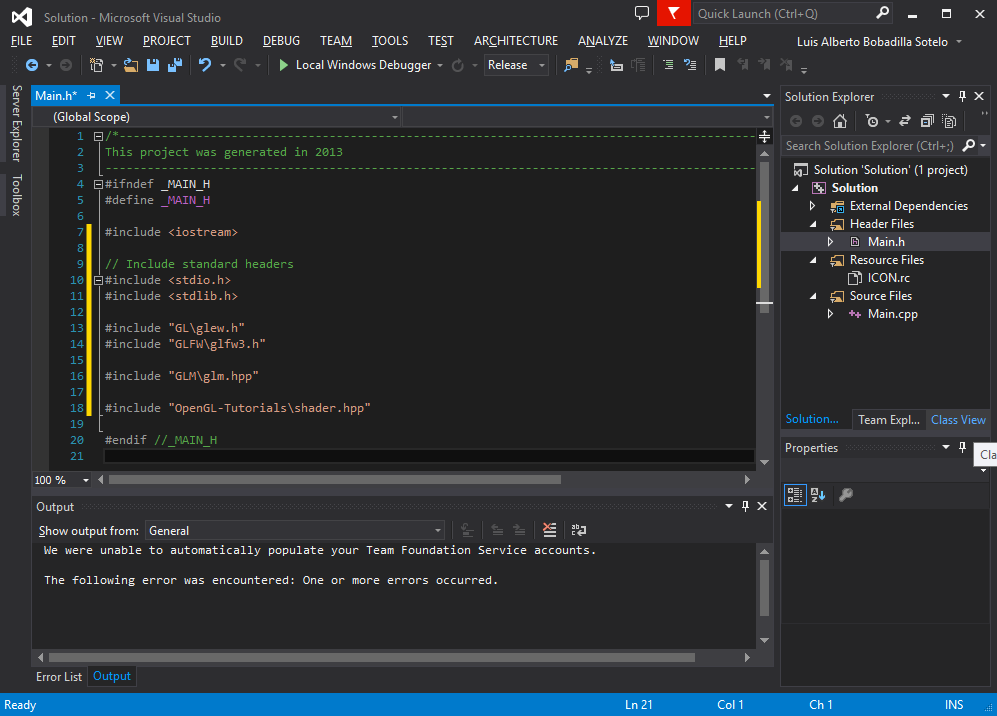
#include <stdlib.h>

#include "GL\glew.h"

#include "GLFW\glfw3.h"

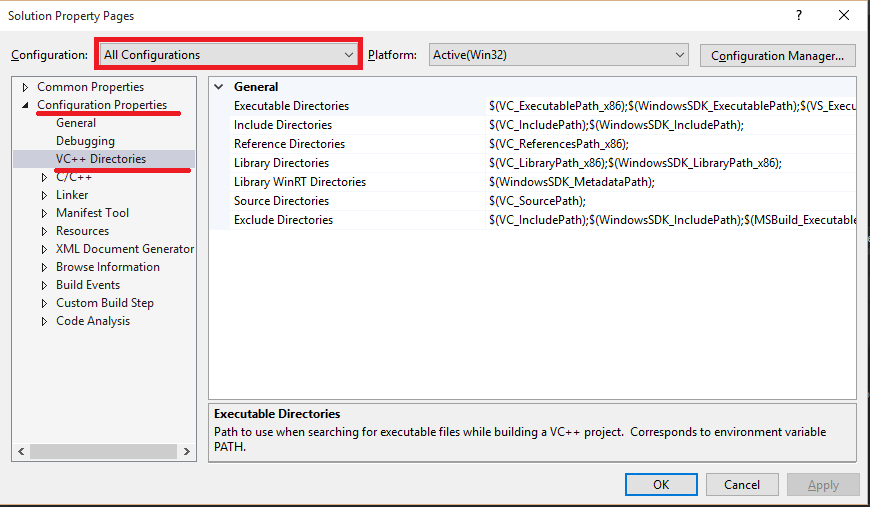
#include "GLM\glm.hpp"

#include "OpenGL-Tutorials\shader.hpp"



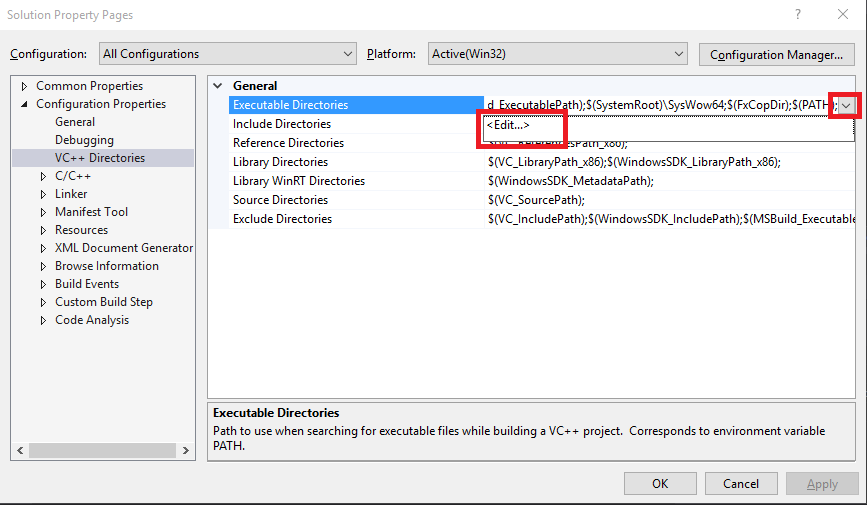
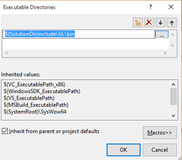
With that you are telling the compiler where the libraries are going to live under your project, the issue is they are relative routes right now, and the compiler ignores the real location of these files, we will fix that next.

28) Go to the properties of your project again (make sure you are under “all configurations” settings) and look for the option VC++ Directories under Configuration Properties:



29 a) making sure you edit the properties and not just pasting this addresses (using the arrows) add the following folders:

$(SolutionDir)include\GL\bin

29 b) Do the same for Include directories but add the following instead:

Include

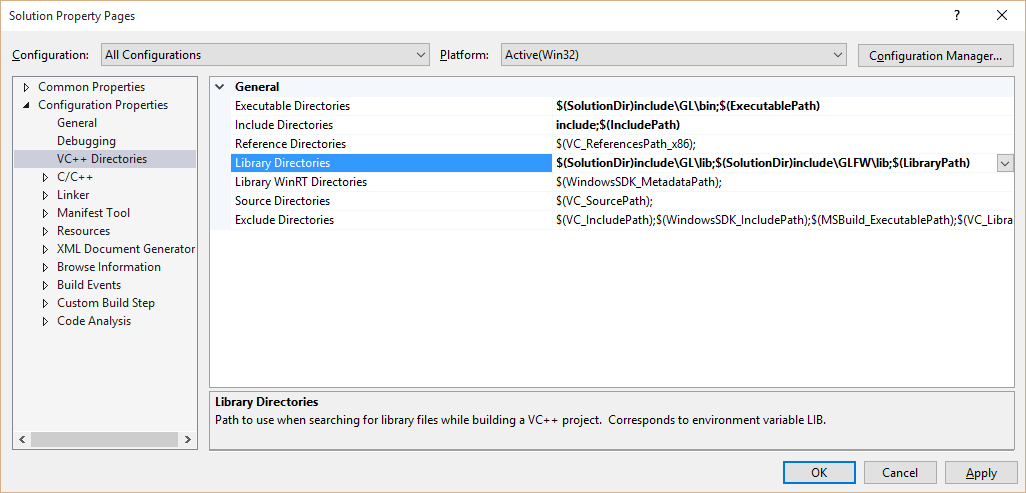
29 c) For Library directories add:

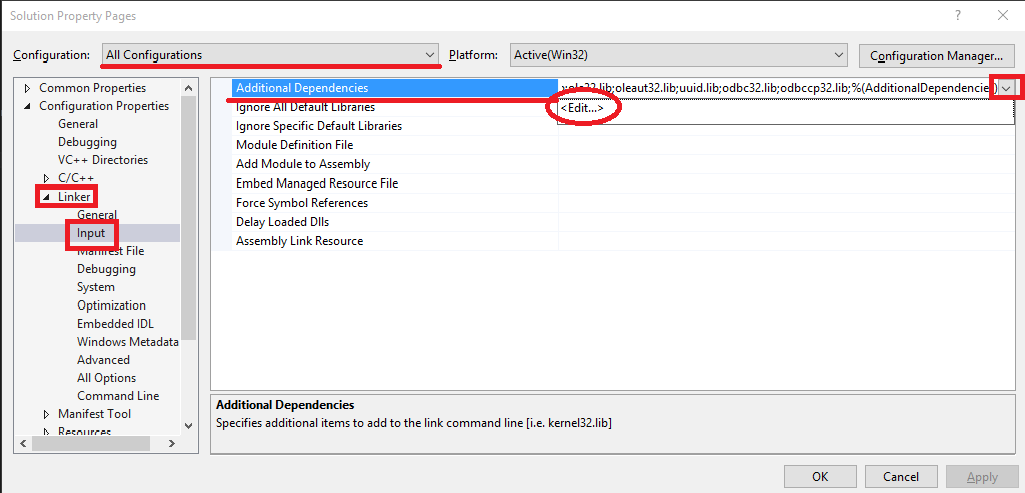
$(SolutionDir)include\GL\lib

$(SolutionDir)include\GLFW\lib

-make sure it looks like this in the end: $(SolutionDir)include\GL\lib;$(SolutionDir)include\GLFW\lib;$(LibraryPath)

29 d) At the end your Property window should look like this:



30) Under “Additional Dependencies” in Linker Input click on the arrow so you have access to the edit options.

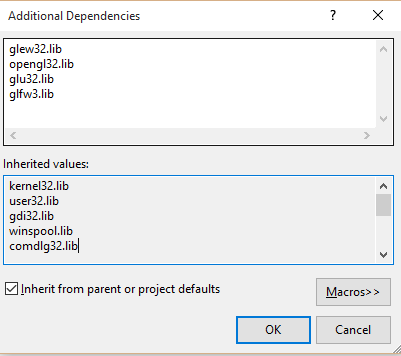
31) Insert the following dependencies:

glew32.lib

opengl32.lib

glu32.lib

glfw3.lib



32) In Post-Build Event under Build Events expand the arrow under Command Line and add the following ***(Don’t forget to change all the quotes to the right type)***:

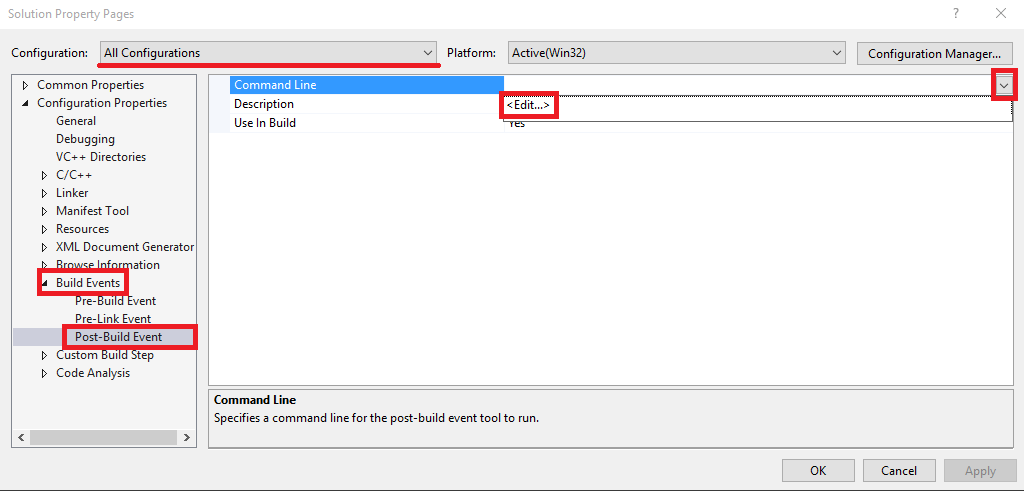
mkdir "$(SolutionDir)\_Binary\"

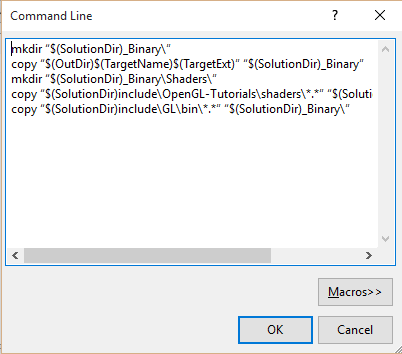
copy "$(OutDir)$(TargetName)$(TargetExt)" "$(SolutionDir)\_Binary"

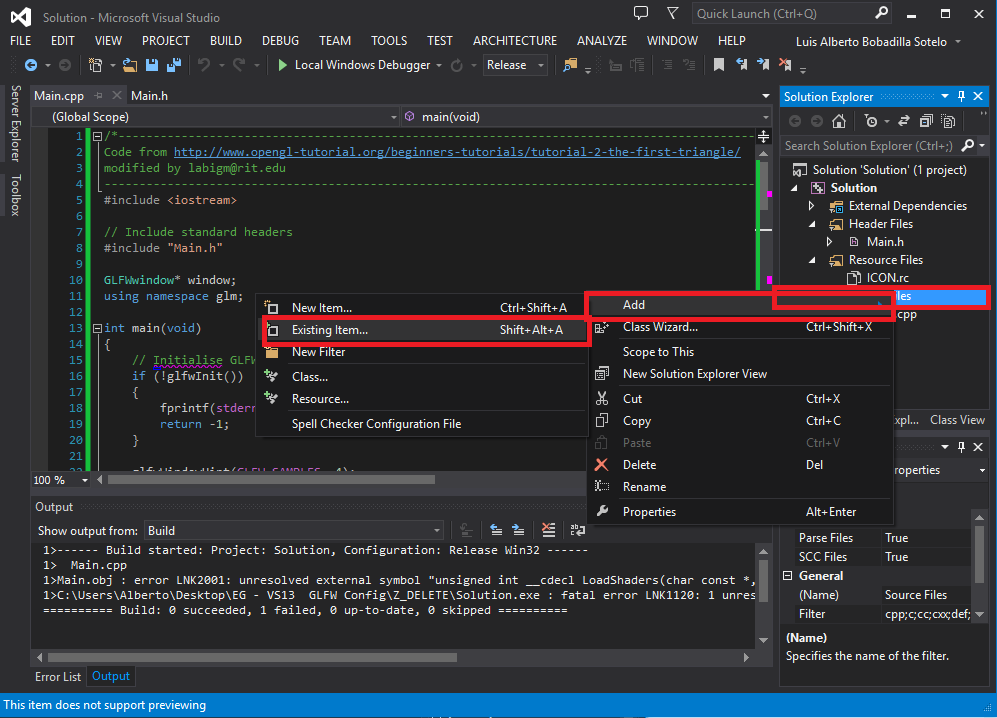
mkdir "$(SolutionDir)\_Binary\Shaders\"

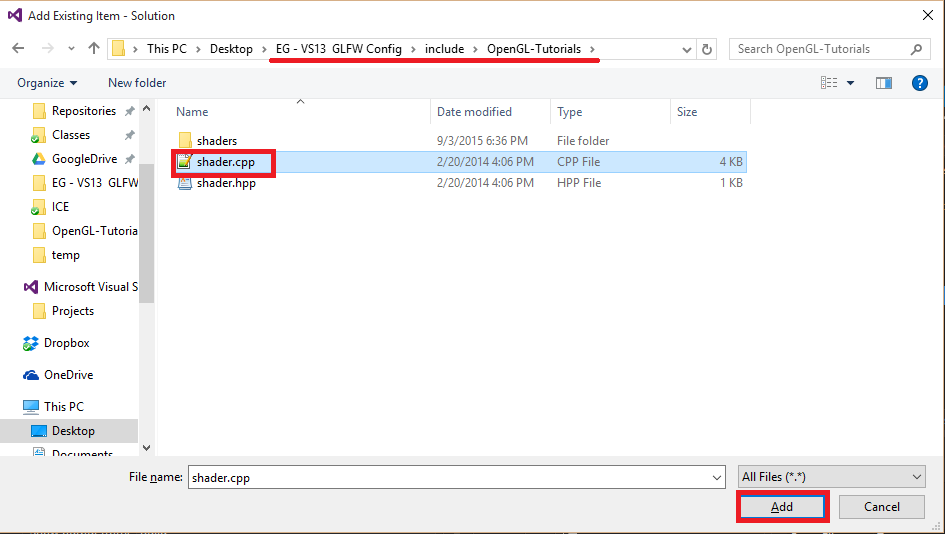
copy "$(SolutionDir)include\OpenGL-Tutorials\shaders\\*.\*" "$(SolutionDir)\_Binary\Shaders\"

copy "$(SolutionDir)include\GL\bin\\*.\*" "$(SolutionDir)\_Binary\"

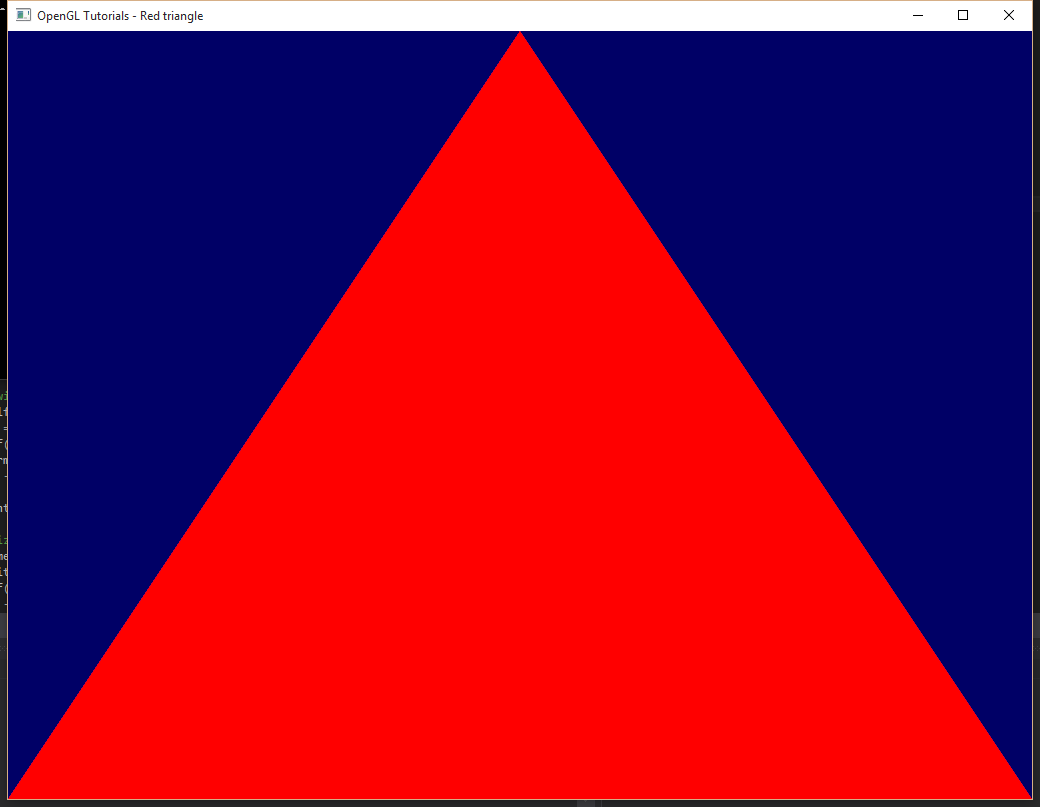




33) Replace your main.cpp file’s with the one provided and include the files shader.cpp to your “source files” folder and the shader.hpp to your “header files” (the files live under include\OpenGL-Tutorials\):



34) Should you have no errors, the window will display:



35) Remove the sdf file and the intermediary Z\_Delete folder and zip your folder.

36) Zip your project and show it to the TA or Teacher, you should also show them the \_Binary folder and execute the release version of the file (the one without the \_d appended)

NOTE: This configuration will only work for an x86 solution, you can apply the same changes to a 64 solution but the corresponding libs and dlls are needed (I only provided 86 versions)