Math 155A (Spring 2022) Project #0:
Basic Introduction to using Microsoft Visual Studio C++ 2022 and OpenGL

Task for project #0: Do *all* the steps below. Fill out the google form at https://forms.gle/oYscdjwxoMN5QMDTA as instructed in the main assignment sheet for Project #0 (available at the course web page).

This document gives basic information on how to run **Microsoft Visual Studio 2022** for C++ programs on Windows 10. Similar instructions work for earlier versions of Visual Studio.

If you have problems with any aspect: Please post about it on the piazza web page (or the discord page).

PART I. CREATING A NEW PROJECT. CREATING AND ADDING FILES. SIMPLE DEBUGGING.

Creating a new C++ project (also called a "Solution") from scratch. Visual C++ groups source files into a "project", and groups projects into "solutions". We do not need much of this infrastructure for our course applications. Instead, we will always create a Solution that contains a single project.

Step 0: Start Visual C++.

- Run the program: From the Windows **Start Menu**, run **Visual Studio 2022 (or 2029)**. (There is no need to "sign in" when prompted.) When prompted, select the Visual C++ settings. (and wait for Visual Studio to start up: it may take a couple minutes). When it asks for **Development Settings**, pick **Visual C++** from the dropdown menu (but **General** will work too, but then the shortcut keys will act a little differently). Select "**Start Visual Studio")**

Warning on UCSD's cloudlabs AWS server: This was set up on cloudlabs only this morning. If you get an error trying to open Visual Studio on cloudlabs, please post about it on piazza.

Step 1: Create the project/solution:

- From the opening screen, choose **Create a new project.** Choose **"Empty Project" for a Visual C++ console program.** It is important to start with an "empty" C++ project. (Earlier versions of Visual Studio have different methods for choosing an "Empty Project" --- make sure is it for an empty Visual C++ general project.)
- Alternately, if you do not have an opening screen: From menu bar, select **File…**, then **New,** then **Project…**, then proceed as in the previous step.
- Set the project name, for instance **HelloWorld.**
- Set a location for the project/solution folder. For instance "My155AProjects", will cause Visual Studio create or use the folder "My155AProjects/HelloWorld". It is usually a good idea to check the box to place the project and the solution in the same folder. (Or, in older versions of VS, to not make a separate folder for the project.)
- ***Very Important:*** Save you your files somewhere in a directory inside "Documents". In cloudlabs, anything placed outside the Documents folder will be erased when you logout. This includes

anything placed in the Visual Studio default location of "source/repos". The Documents folder will have a name like "H:" or "L:**"** (for drive "H" or "L") or a long name like:
sbuss (\\amznfsx7umcv4bw.AD.UCSD.EDU\Share\users),
but with your own user name of course.

- For future projects, you may wish to also save source files elsewhere on the cloud, such as in your google drive. Students have sometimes lost their source files in past years due to placing files in a non-stable location.
- Click "Create" to create the empty project.

Step 2. Add new source files(s). Usually these are .c or .cpp or .h files. This is only for brand new files. If you instead have an existing file that you want to include in the project, see **Step 6** below.

- From the menu bar: **File,** then **New**, then **File…**.
- Choose **Visual C++** templates on the left-hand side of the window that opens. Then choose **C++ File (.cpp)** or other file type as desired. Select **Open**, and you will get a window showing a blank file, temporarily named Source1.cpp. (A similar method works to open a **.h** header file.)
- Type your code into the source window. For instance

```
#include <stdio.h>
int main ()
{
        printf( "Hello World.\n" );
        getc(stdin);
}
```

- Save the file (**File** then **Save Source1.cpp As...**) preferably into the folder (named "HelloWorld" for instance) where your project files are placed. While doing this, rename the file to be something more useful than "Source1.cpp", for instance, "HelloWorld.cpp".

Alternate method for action of Step 2: Use Menu bar **Project…**, then **Add New Item…**.

Step 3. Make the file part of the project/solution. (A slightly silly step, but needed because of the infrastructure of Visual C++.)

- Have the cursor focus in the file to be added to the project, for instance **HelloWorld.cpp.**
- From menu bar: **File**, then **Move HelloWorld.cpp into ...**, then **HelloWorld** (or whatever your project is named).

Alternate method: After file has been saved (say, as in Step 2), use Menu bar **Project**, then **Add Existing Item….** This is also discussed in Step 6 below.

Making a file part of the project/solution means that Visual Studio will know to automatically recompile and rebuild the project every time you change the source code. Files that are part of the project will be shown in the **Solution Explorer** window, usually on the left side of the screen (sometimes on the right).

Step 4. Compile the program.

- From Menu bar **Build**, then **Build Solution.** Or press **Cntl-Shift-B**. (Or maybe the **F7** key depending on your setup.)
- Check the **Output** window that opens at the bottom of the screen.

- Fix any compile errors and repeat. There are two windows listing error messages, an "Error List" window and an "Output" window. Visual Studio will show you the "Error List" but it is usually much better to use the "Output" window. Do this by clicking on the "Output" tab at the bottom of the window.
- Once the output window ends with ***Build: 1 succeeded, …*** it is compiled without errors. It is recommended, but usually not required, to fix all warnings. (Warnings about the "PDB", Project Database, for glew/glfw should be ignored.)
- Extra tip: Sometime you need to use **Rebuild Solution** instead of **Build Solution** if the compiler gets confused about file creation times.

Step 5. Run the program.

- From the menu bar: **Debug**, then **Start Debugging**. Or press the **F5** key.
- See what happens. You should see a console window with "Hello World." Press ENTER/RETURN to end the HelloWorld program shown above.
- The console window has its own icon down in the windows taskbar, that looks like a console: it may be purple (or black). If you lose track of the console window, check the task bar!

A not uncommon issue is that the console is still running your program, but you have gone ahead and updated the source code. Visual Studio will refuse to compile without giving an error message! When this happens, find the console window and terminate the running C++ program.

Question A: What happens differently if you omit the **getc(stdin);** command? Why? [This is a slightly tricky question. Hint: This commands reads a character from the console window.] **Your answer may vary considerably depending on which version of Visual Studio you are running.**

Step 6. SKIP THIS STEP FOR NOW, but here is how to **Add existing files to the project.** If you have an existing file you want to start with, instead of typing your code from scratch.

- Consider whether you want to make a new copy and place it in your project directory. **This is almost always a good idea!!!**
- From the menu bar: **Project**, then **Add Existing Item…**, then select the files you wish to include in the project (usually only .c, .cpp, and .h files). Please note this does **not** make a copy of the file. It just includes it from wherever it is. (See the previous bullet item.)
- You can add multiple files at once, by selecting them together.
- You should now be able to see the files in the ***Solution Explorer*** window on the left (or right) side of your Visual Studio window (under ***Header Files*** or ***Source Files***). Click the file names to open a source file.
- An alternate way to add files: Open the file for editing, then under the menu bar **File** menu, use the **Move … into** option.

Step 7. Debugging example.

- Update the **HelloWorld** program to include a simple loop:

```
#include <stdio.h>
int main ()
{
        printf( "Hello World.\n" );
        int i = 0;
        for ( int j=0; j<7; j++ ) {
```

```
            i += j*j;
        }
        printf( "i = %d.\n", i );
        getc(stdin);
    }
```

- Compile the program; fix any problems. If you get compile errors, click on the error message to go to the line with the error. Helpful hint: Visual Studio will dynamically put little red squiggles where there are (or will be) compile errors. Mouse over the squiggle I += Jto see the potential compile error.)
- Now in the window for the source code, find the grey column to the left side of the window. Click there in that column on the line "`int i = 0;`", to put a red circle inside that column. This is a **breakpoint.** Click again if you want to remove the breakpoint. A breakpoint will cause the program to stop there for you debug.
- Insert breakpoints on the line "`int i = 0;`" and on the line "`getc(stdin);`".
- Run the program again. It will start running the program, and stop at the first breakpoint, showing with a yellow arrow what line the code is at.
- Note that below the source code, there is a panel entitled "**Autos"** showing the value of the variable i.
- Use the **F10** command (or menu bar, **Debug**, then **Step Over**) to single step the program execution to the next line. Note the value of the variable i changes in the **Autos** panel, and the value of j is added.
- Continue single stepping: Note how the values of i and j change, and how the arrow moves to the next line each time.
- It is also possible the change the values of i and j in the **Autos** panel. Just type in the desired value and press enter!
- Press **F5** again: the program continues running to the next breakpoint.
- You may use **Shift-F5** to terminate the program.

Reopening a Visual Studio project: The main project file is called a "Visual Studio Solution" file, a**.sln** file. Use this to reopen the project. Alternately, use the **File** menu in Visual Studio to open the file.

Question B: Does debugging, and single-stepping, and watching variables' values change make sense to you? Is this something new for you? Do you have any questions about debugging in this way?

Other useful things to know about debugging. When you have function calls, use **F11** and **Shift+F11** to step into a function call (instead of the next line in the current routine), and out of a function call.

It is strongly recommended that you get good at the basic features of debugging: it can GREATLY simplify programming! This kind of interactive debugging can be an enormous help in finding and fixing errors in your code. It is recommended that you make a habit of single-stepping like this through all new pieces of code. This can be invaluable in finding errors in code. In fact, this is one of the main reasons to use a good IDE like Visual Studio instead of a command line environment.

There is an enormous range of options and features in debugging. You may wish to explore these on your own as needed; or ask one of the TAs or Professor Buss for pointers. The commands above take care of most basic needs however, and are a great help in debugging your code.

PART II – TRYING OUT AN OPENGL PROGRAM --- SimpleDrawModern

Now try making a new Visual Studio solution (project) with a sample OpenGL program:

Step 1. Download *SimpleDrawModern.zip* from http://www.math.ucsd.edu/~sbuss/MathCG2/OpenGLsoft/SimpleDrawModern/. Extract the files from the zip file, and place them in a new folder (say under your "MyProjects" folder, where your HelloWorld project is). **It is important to extract the files, and not try to use them while they are still inside the zip file.**

Step 2. Create a new Visual Studio C++ project, following the same instructions as before. Add all three sources files from that you just downloaded. Do not add the executable file to the project however.

Step 3. Run the program. (See the "workaround" below if the program will not compile due to missing headers.)

- Build and then run the program. (Use **Cntl-Shift-B** or maybe **F5,** or use the dropdown menus for **Build** and **Debug**). This open two windows: a ***graphics*** window and a ***console*** window. The graphics window holds all the output of OpenGL commands. The console window holds text input and output (stdin, stdout, and stderr). In this course, we never use stdin.
- For the SimpleDrawModern program, you should initially see three overlapping shaded (colored) triangles.
- Hit the space bar to cycle through five different images: with line segments, then a "line strip", then a "line loop", then dots, and finally the three triangles again. Also note what happens when you resize the window, say by making it tall and skinny, or short and wide.
- If everything worked OK, go onto to Step 4.

POTENTIAL PROBLEMS (that can happen if you are running on your own computer, but should not happen on CloudLabs or the computer labs):
*** If there are error messages about missing GLFW and GLEW files, your systems is misconfigured. If so, here is a work-around that can work even if you do have administrator privileges on your system:**

- Download the GLFW and GLEW files from the dropbox link https://www.dropbox.com/sh/tn5hzmn6hw2bab4/AADNwXoNzO-rZ7yohp71LZOca?dl=0
- At that location, there are three static libraries (.lib) files, and two folders (GL and GLFW). Please note whether you are getting the 32-bit (x86) versions or the 64 bit (x64) versions.
- Place **all** of these in the ***same*** folder as your Solution file (.sln file).
- In each of the two .cpp files, SimpleDrawModern.cpp and ShaderMgrSDM.cpp, there are two #include lines: #include <GL/glew.h> and #include <GLFW/glfw3.h>. Change these to be #include "GL/glew.h" and #include "GLFW/glfw3.h" instead. (Changing angle brackets to double quotes.)
- The program should be able to compile and run now. You will need to select the appropriate x86 or x64 software version (at the top of the Visual Studio window) for this to work. If you are using your own computer, you should install GLFW and GLEW correctly to simplify your work during the quarter. If you get this problem on the computer lab systems, please let us the ID for the machine with the problem, so that ACM can re-install GLFW and GLEW on that system.

*** If you get an error message about cannot access the .exe file, it could be that your antivirus software is blocking access to it as an unknown executable. (Again, should not happen in CloudLabs.)**

Step 4. Play with the SimpleDrawModern program a bit.

- LOOK AT SOURCE FILES: Examine the SimpleDrawModern.cpp and ShaderMgrSDM.cpp source files. To open the source code files, use the Solution Explorer window, which will be on the right hand or left hand of the Visual C++ windows. Double click on file names to open the file.
- LOOK AT THE CODE TO SEE HOW KEYBOARD INPUT IS HANDLED (IN THE GRAPHICS WINDOW):
    - Note from the comments at the beginning that it takes space key as input. This space key is to be typed into the ***graphics window***, not the console window.
    - You may also use the Escape key or "X" or "x" to exit. All these are typed into the graphics window.
    - We will discuss this in a class zoom lecture.
- PLAY WITH COLORS:
    - Try changing the line loop's color from yellow to cyan. For this, find the appropriate glVertexAttrib3f command on line 208 in SimpleDrawModern.cpp with color arguments (1.0, 1.0, 0.2) and replace them with the RGB values (0.0, 1.0, 1.0) for Cyan. Recompile the program, and see if it worked!
    - Try the same again, but now changing the color to **BROWN.** (Brown can be a difficult color to match.) You may wish to google for "RGB color" or "RGB code for brown". You will most find suggested R/G/B values ranging between 0 and 255. Scale them into the range 0.0 to 1.0 by dividing by 255 to use them in the SimpleDrawModern program. Are you able to obtain a good brown color? (The best brown color might not be named "brown".)
- UNDERSTAND HOW INTERRUPT-DRIVEN PROGRAMMING WORKS. We'll discuss this in lecture too.
    - Take a look at how the code handles key strokes with the routine **key_callback** in SimpleDrawModern.cpp. This is a kind of interrupt-driven, or callback-based interface; it may be rather different from the way you are used to programs receiving input. This is setup by the function **setup_callbacks** using the OpenGL/GLEW routine **`glfwSetKeyCallback`.**

Question C: Were you able to successfully change the green lines to cyan? And to brown? Did you find a good brown color? What was it?

Question D: Does the code for receiving keystrokes as input make sense to you? Do you have good idea of how the interrupt-driven programming works?

By the way, it is OK to answer "No" to these questions. If so, please talk with a TA or Professor Buss, and let us know in the google form what needs more explanation.

PART III: REVIEW THE ABOVE MATERIAL.

Final Question: Do you have any questions about how or why things work in the above steps? Did you any particular difficulties? Were there any steps you could not make work successfully?