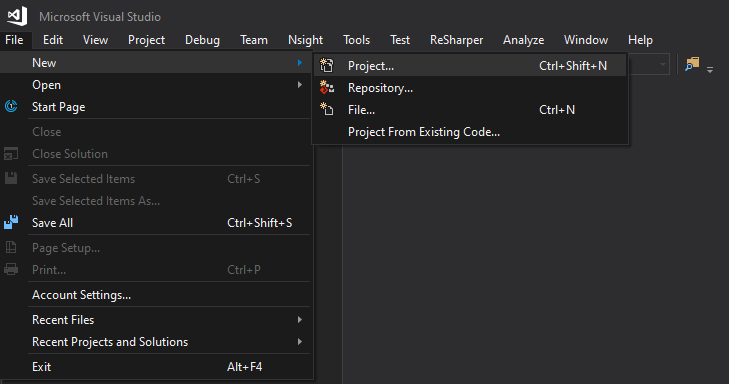
# Week 1: Easy Graphics

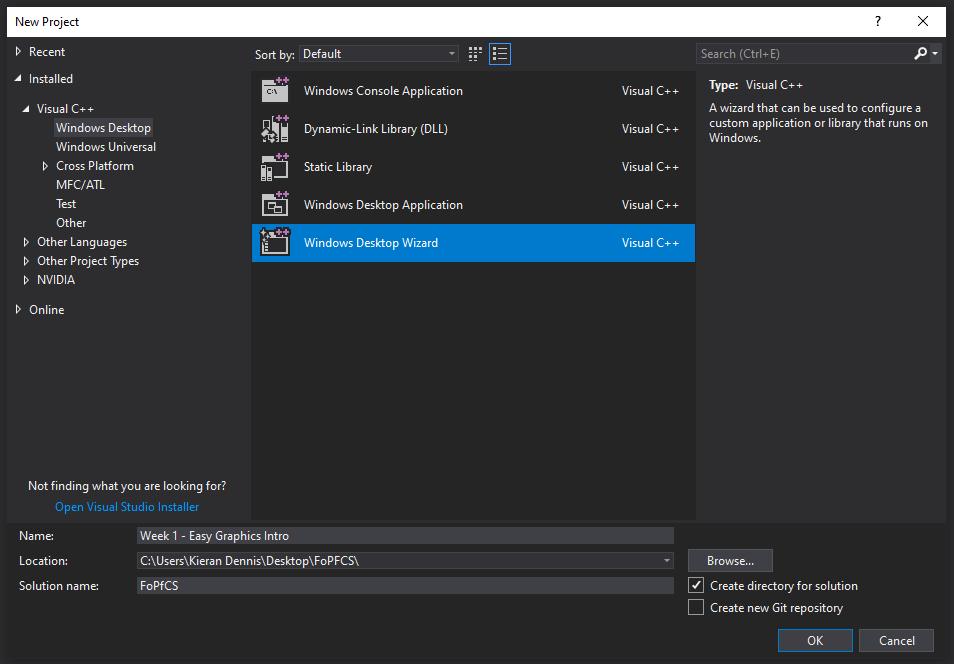
This week we will look at creating a different type of application to the console ones you have been used to in semester 1, which will allow us to work on graphical applications.

## Creating a Win32 Visual studio Project

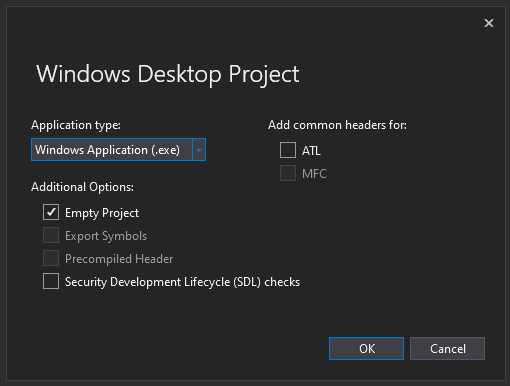
Open the new project window



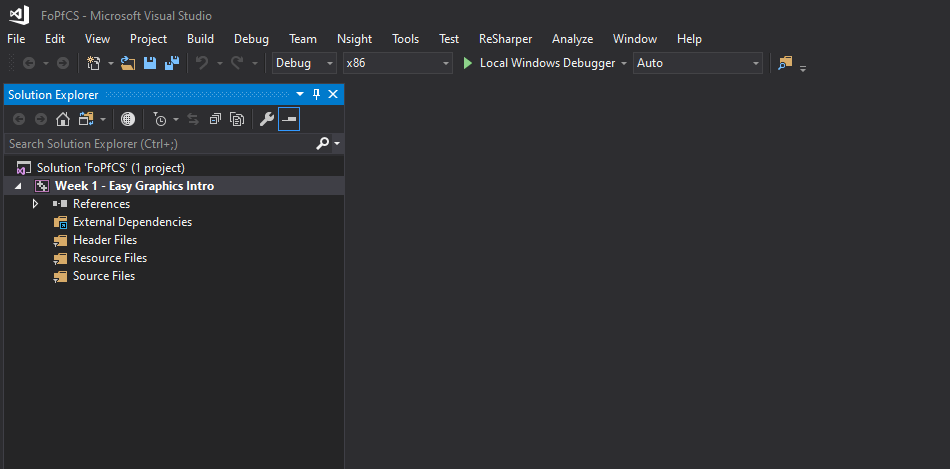
Under the ‘Visual C++’ section, select ‘windows desktop, then ‘Windows desktop Wizard’. Give your project and solution a useful name and then click OK.



On the next window, Set the Application type as ‘Windows Application (.exe)’, and tick the ‘Empty Project’ box. Untick any other boxes as with the example below, then click OK.

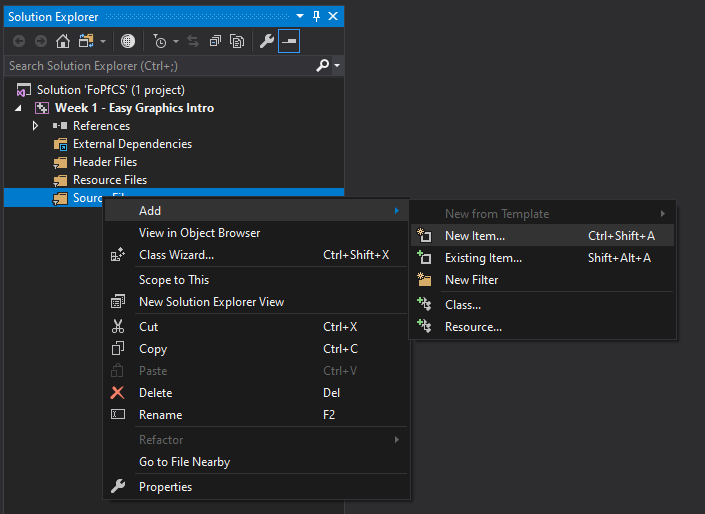


You should now have a project that looks something like this:

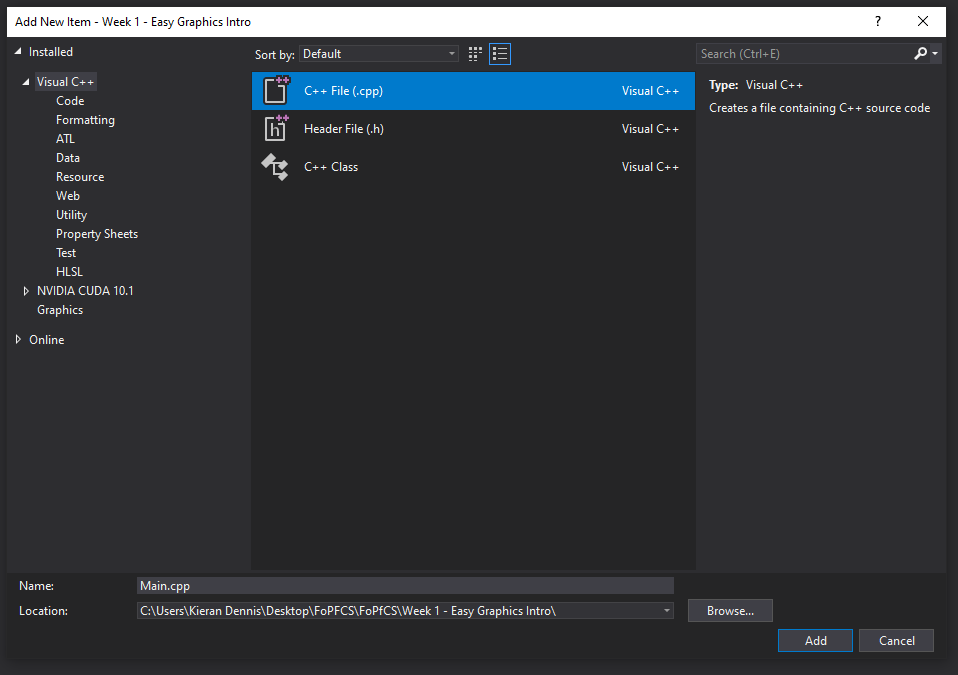


## Setting up Easy Graphics

First let’s create a main file, The entry point for our application.



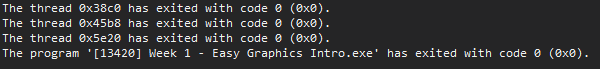
Next, name it main.cpp and click add.



You will now have an empty cpp file in your project, lets fill out some basic code so we can run it. First, Include the windows header file, and then create your main method.

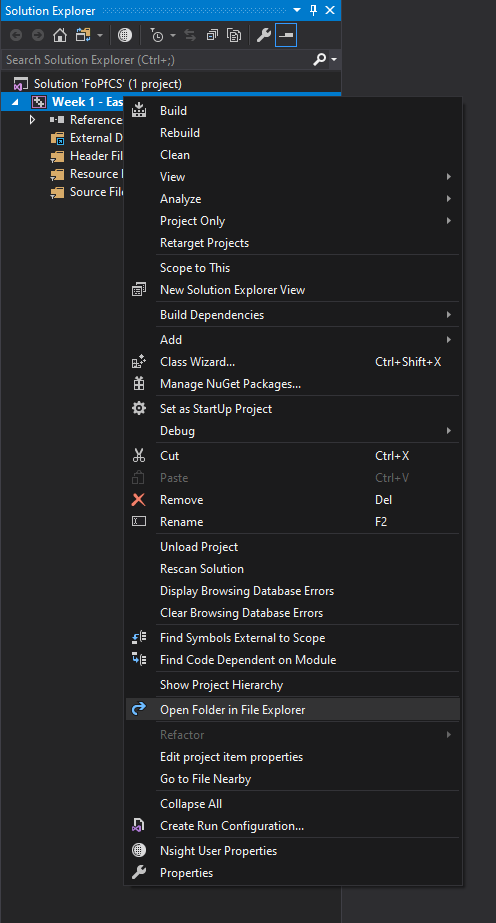


This should run and then immediately close, with the output looking something like this:

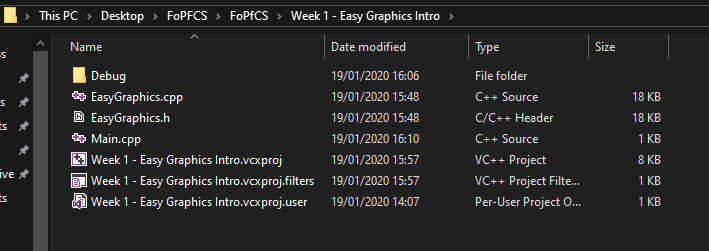


After that is working, download the EasyGraphics .cpp and .h file from the blackboard site. Save these into the same folder as your project.

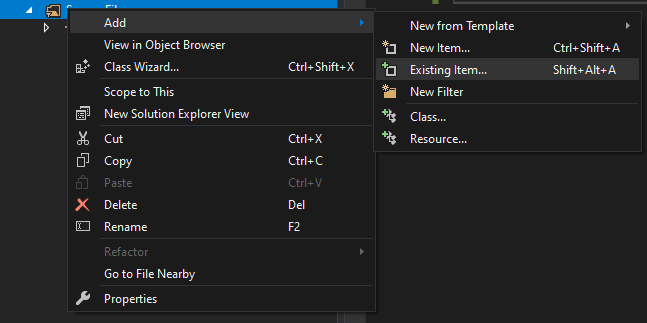
You can easily find the location of your project by right-clicking and selecting ‘open folder in file explorer’ as below:



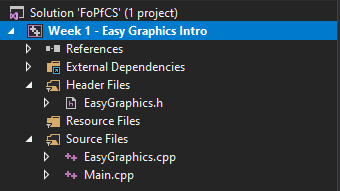
Once you have downloaded and saved these files, the folder should look something like this:



Right click and select Add > Existing Item

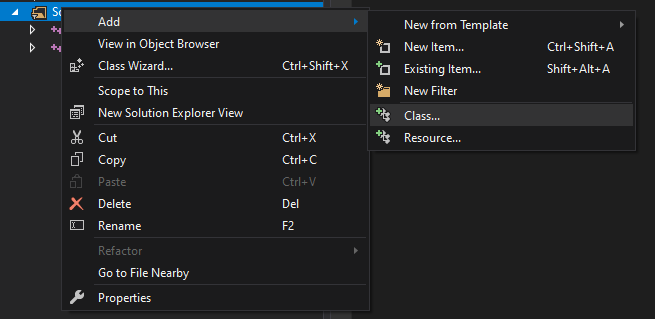


Add both the EasyGraphics.cpp and the EasyGraphics.h file to your project. Make sure to drag the .h file into the header files folder. Your project should now look like this:

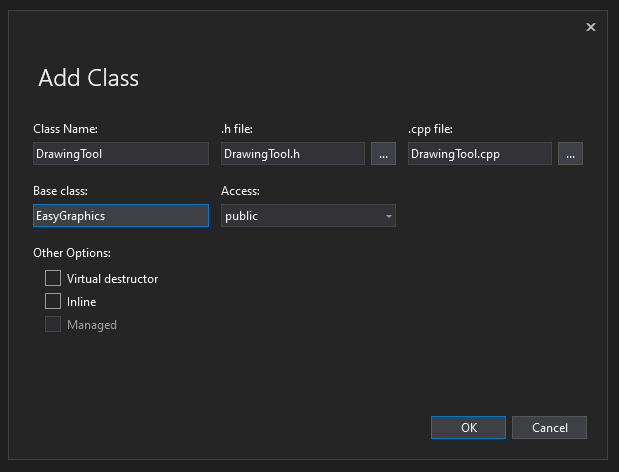


Let’s start using the EasyGraphics library, the method shown might seem convoluted, but follow along for now and all will be explained.

Firstly, we need to create a place for us to write our code, so let’s create a new class. Go to Add > Class.



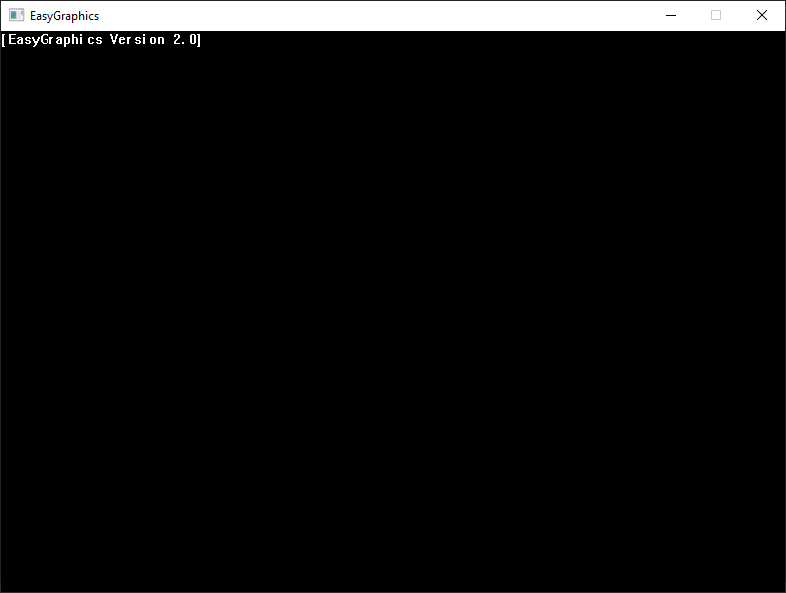
Fill in the next Screen with as follows, giving your new ‘Class’ a friendly name, I’m going to call mine DrawingTool. Fill out the Base class field with ‘EasyGraphics’, This will allow our new class to use the code within Easy Graphics.



Now that we have our DrawingTool Class, we need to use it. Let’s write a bit of code within our main function to tie this all together.



Running the program now should give a basic output in a window like this:



## Drawing to the canvas

So far, we’ve been getting things set up. Now its time to draw something to the screen!

We can do this by overriding functions from the EasyGraphics library.

First, we need to set up our onDraw method, here we tell our program what to do each time it is told to draw to the screen. The header file contains a ‘Prototype’ of our onDraw function, and the cpp file the implementation. All we will do for now is clear the Screen.

**DrawingTool.h**



**DrawingTool.cpp**

****

Now you should have a plain white background on which to start drawing things!

We can now use the functions given to us by EasyGraphics. Have a go at using the following functions within the onDraw method of DrawingTool.cpp. Explanations of these can be found within EasyGraphics.h, along with colour constants.



**TASK: draw a house using these functions.**

## Mouse Interaction

We need our mouse to be able to interact with what’s on the canvas. So, to practice that let’s draw a circle around the cursor.

First, we’ll need to extend another function from EasyGraphics: **onMouseMove.** Add this to the DrawingTool files as we did before for the onDraw function.

We’ll also need somewhere to store the mouse’s current position, the onMouseMove function gives us this in the form of an x and y, so lets add two variables in the Drawing tool header file and call them mx and my. We can then assign these inside the OnMouseMove function.



Try using mx and my to draw a circle within the onDraw method.

What do you notice? Does the circle follow the mouse?

*Hint: Does your program draw each time the mouse moves?*

## Stopping the flickering: Double Buffering

Once you have the circle following your mouse around, you’ll notice that anything drawn to the canvas flickers - but don’t worry, this can be fixed!

So far, our program has been drawing to the canvas immediately, each shape drawn to the screen is done so one after the other as soon as the function is called. This has the side effect of not lining up with the refresh rate of the monitor. To fix this, we tell EasyGraphics to first draw to an off-screen canvas and then show everything all at once. This is called double buffering, and we do this with the following function:

setImmediateDrawMode(false);

Add this to the constructor of the Drawing tool.

You’ll notice that the screen no shows nothing, this is because we have to manually call EasyGraphics’ OnDraw method to tell it to show everything on the off-screen canvas. Add the following line to the end of the drawing tool’s onDraw method.

EasyGraphics::onDraw();

## OnKeyDown

Here’s a hint at how to make your application respond to keypresses:



Follow the same steps as before, make sure to prototype this function in the header file. You might need to add another variable to store a key press state.

**Task: Make the Door of the house drawn previously open and close on a key press.**