

Spike: Task 5**Title:** IDE Familiarity**Author:** Sam Huffer, 101633177**Goals / deliverables:**

- A QuickStart guide to Visual Studio detailing, with screenshots:
 - The steps to creating a new command line project, adding new files, and compiling and running it.
 - The steps to create a breakpoint, run the program in Visual Studio's Integrated debugging system so the program stops at the specified point, and inspect values of variables during debugging.

Technologies, Tools, and Resources used:

- Visual Studio 2017 (Version 15.9.8, with JetBrains ReSharper installed)
- Microsoft Word
- Snipping Tool / the "print screen" key

Tasks undertaken:

- Found the version number of Visual Studio 2017 under "Help > About Microsoft Visual Studio".
- Set up the QuickStart guide in a new Word document with appropriate section headings.
- Worked through the steps to achieve each of the tasks that the QuickStart guide is intended to instruct beginners about, taking screenshots at necessary intervals, pasting screenshots in the document under the correct headings, and adding captions going into appropriate detail about the depicted step of the task at hand.

What we found out:

- Shortcut keys (where available) for the aforementioned tasks.
- Refreshed my understanding of how to work with breakpoints.
- How to view the values of variables at a particular line, within a method, and chosen variables across the program.

Visual Studio 2017 – QuickStart Guide

Version

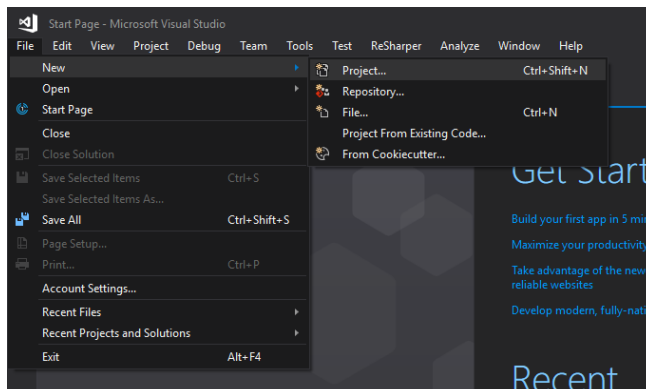
This QuickStart guide was made for users of Visual Studio 2017 (Version 15.9.8, with JetBrains ReSharper installed. Due to ReSharper, some steps may vary from what will work for your version of Visual Studio).

Overview

1. Project Setup
 - 1.1. Creating a new command line project
 - 1.2. Adding new files
 - 1.3. Compiling the program
 - 1.4. Running the program
2. Debugging
 - 2.1. Inserting breakpoints
 - 2.2. Run program so it stops at the breakpoint
 - 2.3. Inspecting values of variables during debugging

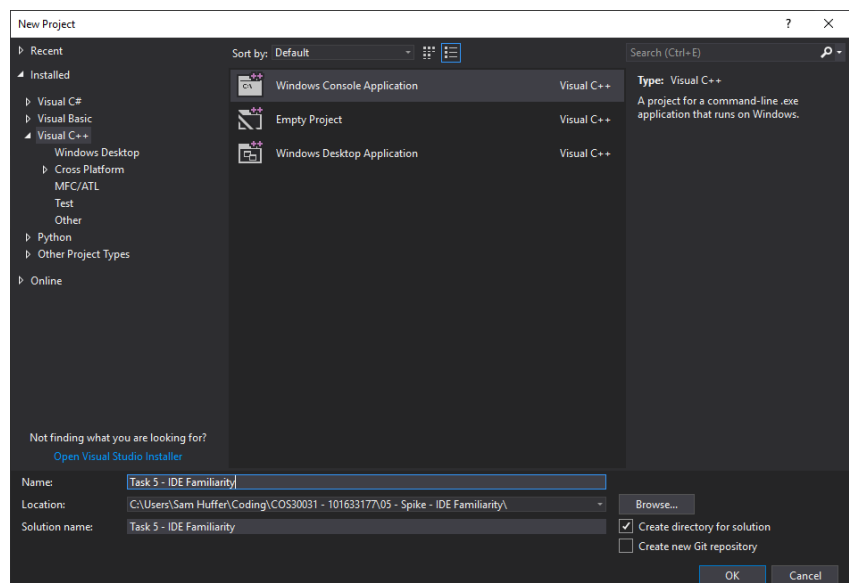
1. Project Setup

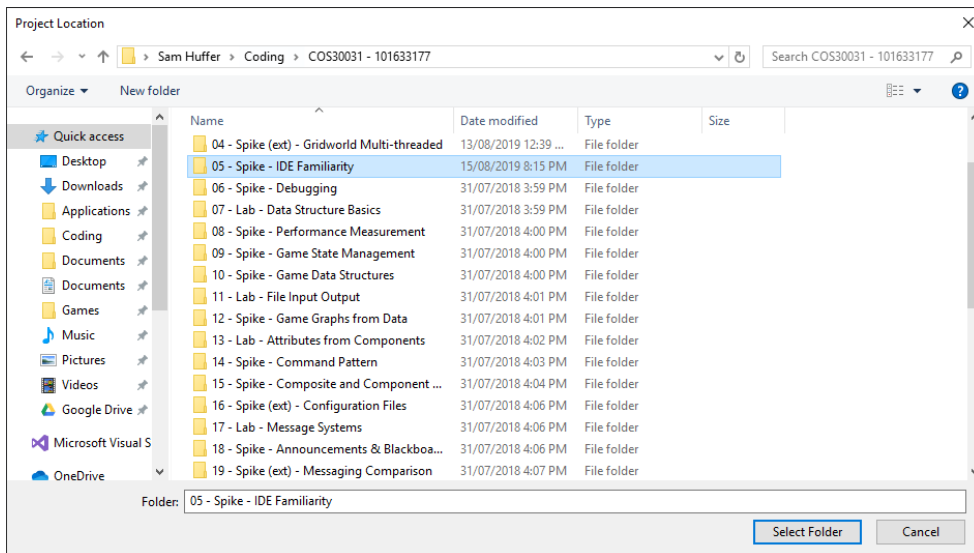
1.1. Creating a new command line project



Step 1: In the top-left corner, click "File > New > Project". This will open a dialog box for creating a new project. Shortcut keys: Ctrl + Shift + N.

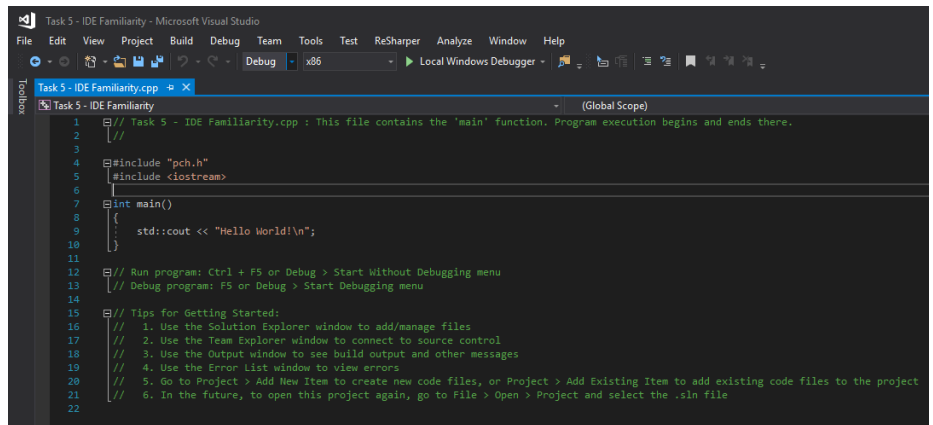
Step 2: When the dialog box opens, there will be a panel on the left with various project types grouped by language. Click on "Visual C++". This will show the appropriate project types in the centre of the dialog box. The one we want is the "Windows Console Application"; click on it to select it. Down the bottom you can enter your chosen name for your solution, and choose where to store it by clicking "Browse"; this will open a second dialog box.



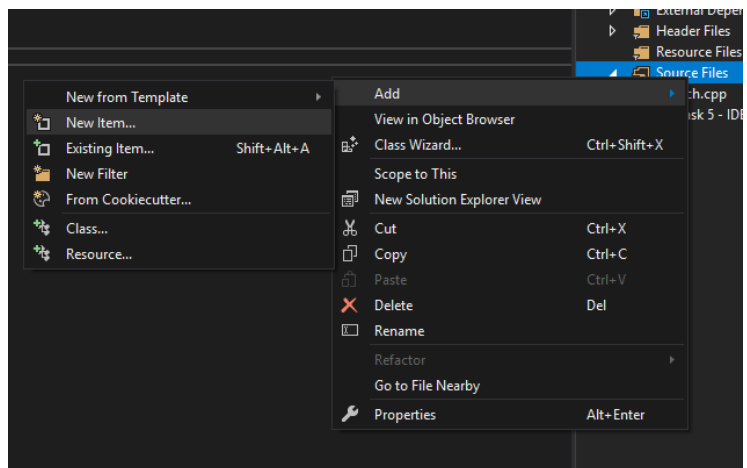
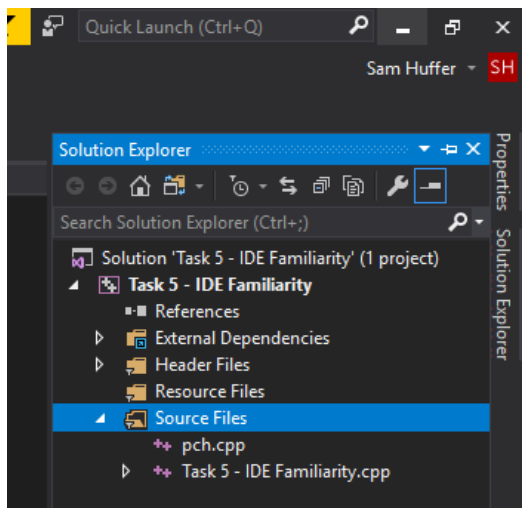


Step 3: In this new dialog box, navigate to the folder you wish to store your project in, select it, and click "Select Folder". If you navigate into the folder you wish to store your project in, click "Select Folder" without clicking on any sub-folders inside your chosen folder. Once this dialog box closes, leaving only the original, click "OK" in the bottom-right corner to create your project.

Once the project has been created, Visual Studio will open up a file editing tab for a C++ file conforming to its basic template. From here, you will be able to add, remove and change the code in that file at will.

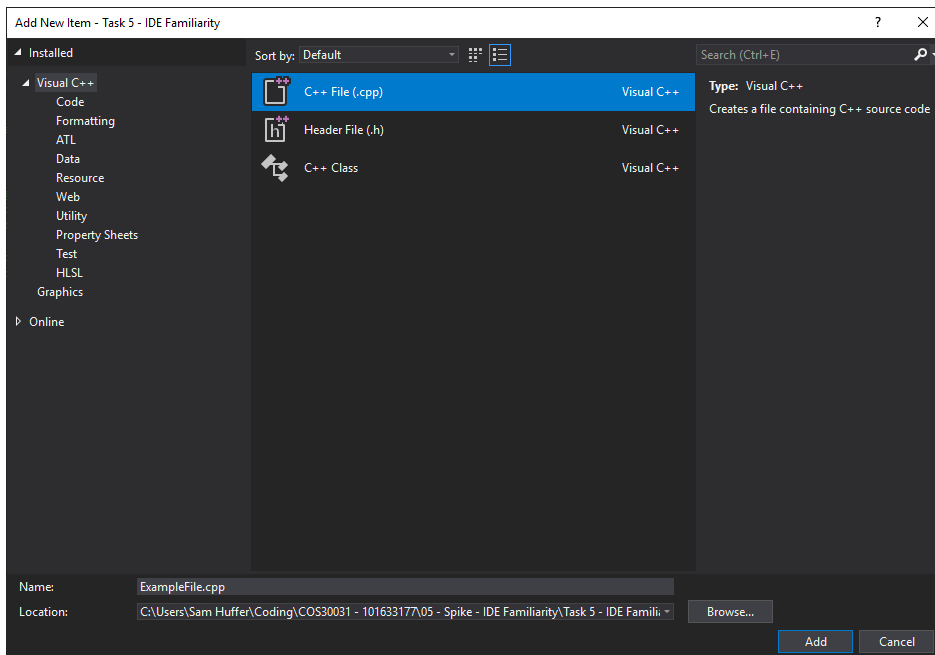


1.2. Adding new files

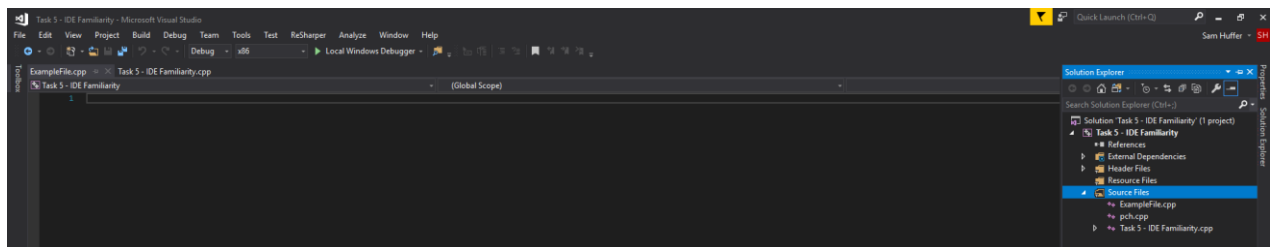


Step 1: Click on the docked "Solution Explorer" tab on the right-hand side of the screen (here seen under the "Properties" tab). This will open up the Solution Explorer tab. Clicking on "Source Files" to expand the folder will show you all the files you've coded that are currently in your project.

Step 2: To open a dialog box for adding a new file, right click anywhere on the Solution Explorer, and in the context menu that appears, click on "Add > New Item".

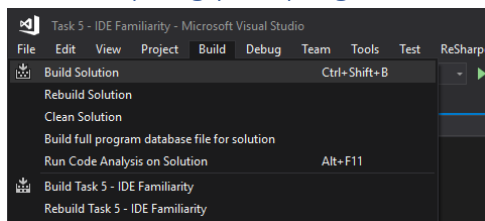


Step 3: In the dialog box, select “C++ File (.cpp)” in the centre, and type in the name you wish to give your new file (e.g. “ExampleFile.cpp”). If you wish to place it within a particular sub-folder of your project, click “Browse” in the bottom-right corner to open up the dialog box you saw when you were choosing where to store your project in Section 1.1, Step 3. Navigate to your desired folder and click “Select Folder”. Once complete, click “Add” in the bottom-right corner.

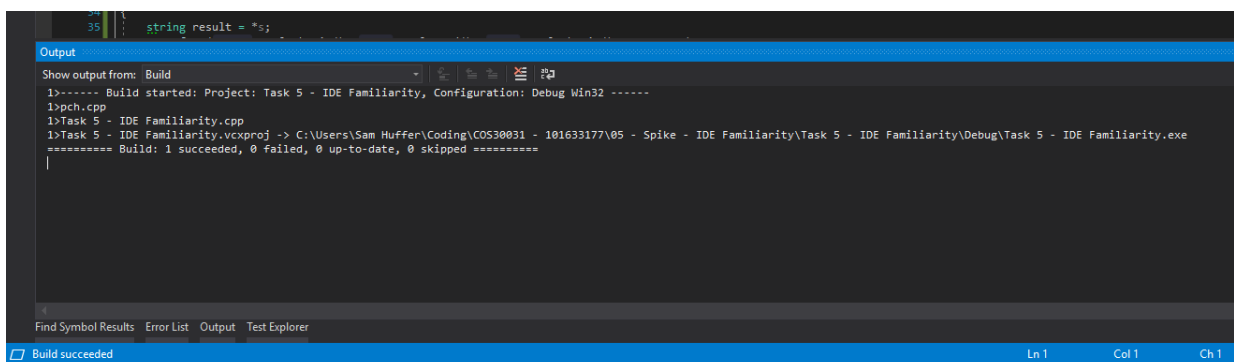


Once added, your new file will be opened in an editing tab next to any other files in the project you have open. This one will be completely blank. If you close it by clicking the “X” on the right of its tab, and wish to open it again later, it will now be available in the Solution Explorer in the “Source Files” folder.

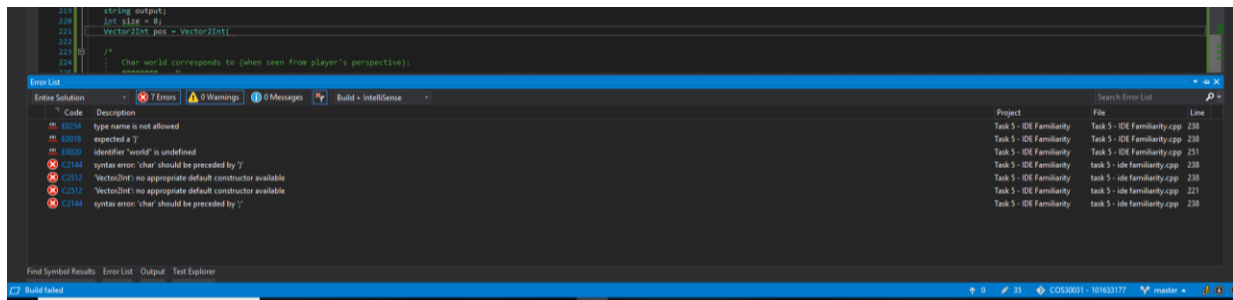
1.3. Compiling your program



Step 1: To build your program once you have some code you want to test, at the top of the screen, click “Build > Build Solution”. Shortcut keys: Ctrl + Shift + B.

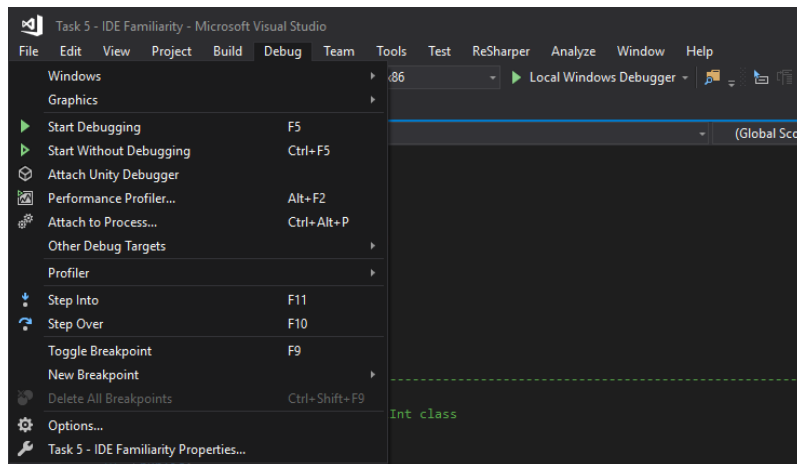


If the build is successful, you will see the “Output” tab pop up down the bottom with output similar to this, saying “Build: 1 successful”.



If there are bugs in your code, the build will fail and the “Error List” tab will pop up instead. It will outline what errors are in the code, and list which file they’re in and what line they’re on. Double click on an error to jump straight to its position in the code. Hint: click “Line” in the top-right corner of the “Error List” tab to order the errors by line number. Often an error at one point in the code will produce additional errors further down, even if the rest of the code is perfectly alright. For example, my incomplete assignment of a `Vector2Int` to the variable “pos” on line 221 (top-left of the screenshot) results in the compiler thinking that there are other errors below it. If I were to fix that error, any subsequent errors caused by it would be fixed. Note that unrelated errors would still remain and need to be fixed in their own right.

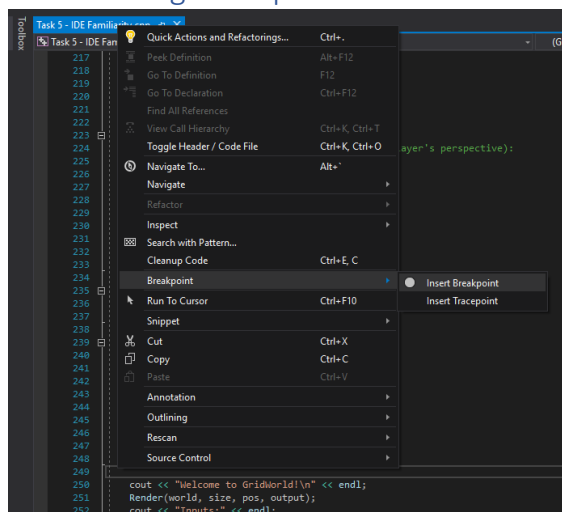
1.4. Running the program



Step 1: to run your program, you have several options. The easiest would be to click “Local Windows Debugger”, which is in the top-centre of the screen next to a green “play” triangle. You can also click on “Debug > Start Debugging”, which is equivalent to clicking “Local Windows Debugger”, or “Debug > Start Without Debugging”, which will also start your program but without running some of Visual Studio’s debugging functionality. The shortcut keys for these options are F5 and Ctrl + F5 respectively. Note: each will also build your program before running it if you have saved any changes since you last built it.

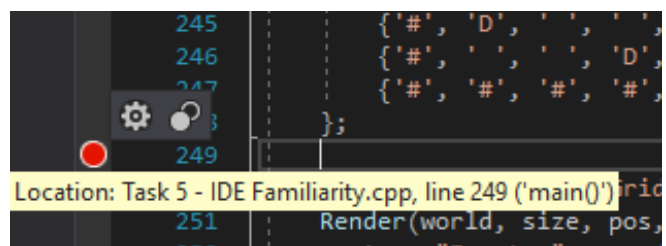
2. Debugging

2.1. Inserting breakpoints

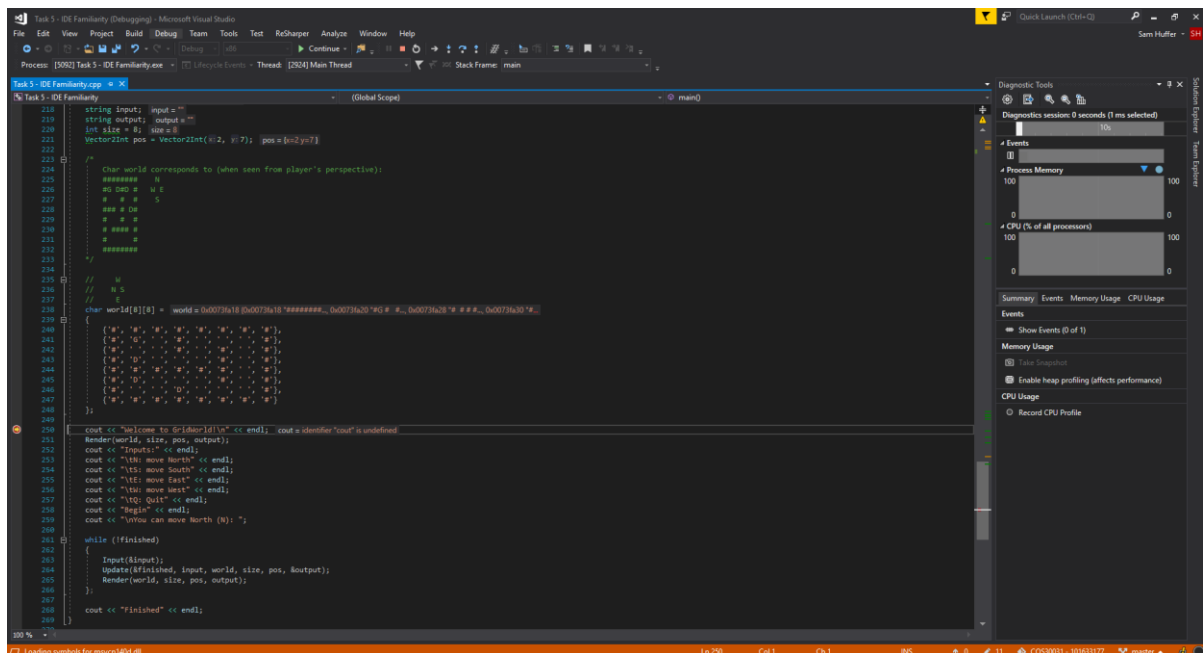


Step 1: to insert a breakpoint in your code such that it pauses execution at this point, right click on the line where you want to insert it, and in the context menu that appears, click on “Breakpoint > Insert Breakpoint”.

This will insert a red dot beside the number of the line where it was inserted. To remove it, simply click on it. To disable it, hover your mouse over it, and a box with a cog and two circles will appear. Click on the two circles to enable and disable the breakpoint without removing it.

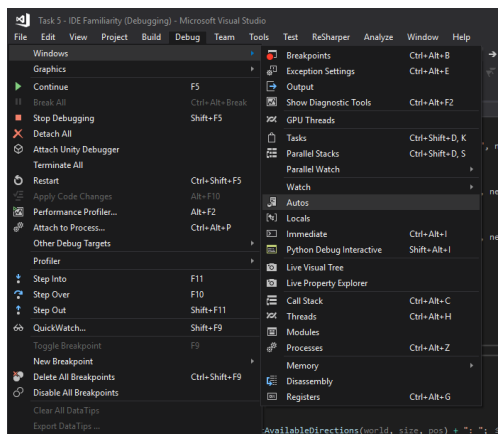


2.2. Run program so it stops at the breakpoint

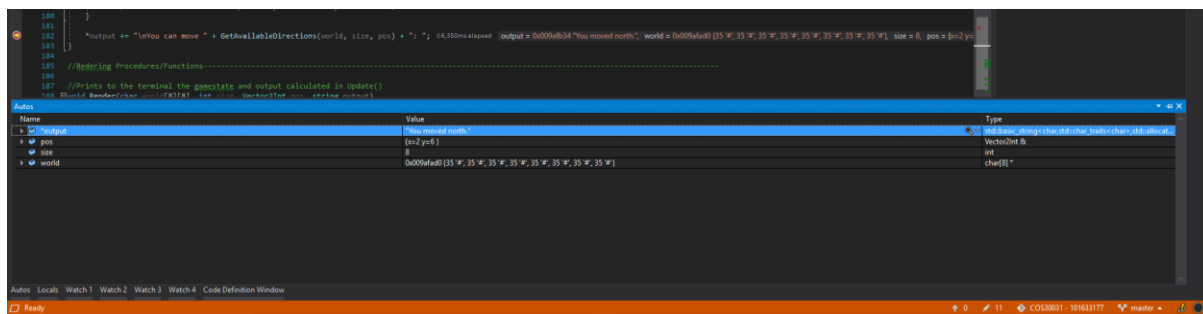


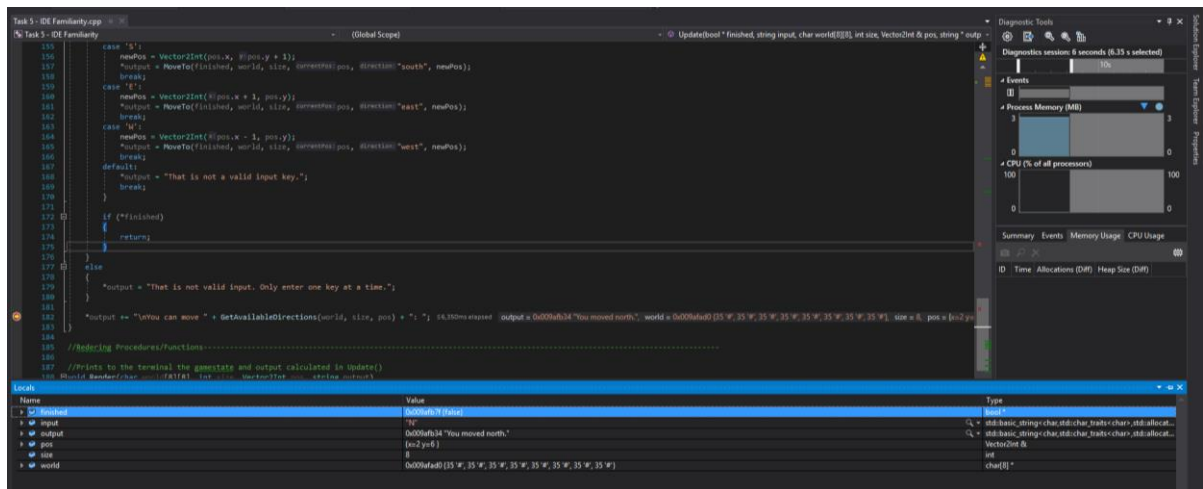
Step 1: to run a program such that it stops at a breakpoint when that breakpoint is reached, click on “Local Windows Debugger” or “Debug > Start Debugging” as discussed in Section 1.4. The program will start, and when you reach a line with an enabled breakpoint, it will switch from the window the program is running in to the editor window at the line featuring the breakpoint, with the red dot of the breakpoint encapsulating a yellow arrow to indicate the line, in case you have multiple breakpoints. Note: do not click on “Debug > Start Without Debugging”, as that will run the program without any debugging features, breakpoints included. If you run the program this way, Visual Studio will ignore your breakpoint.

2.3. Inspecting values of variables during debugging

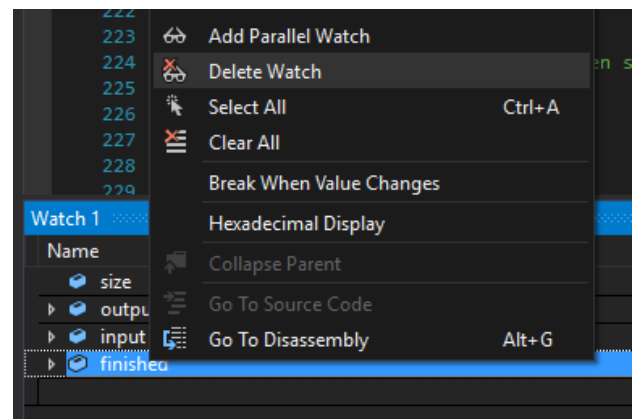
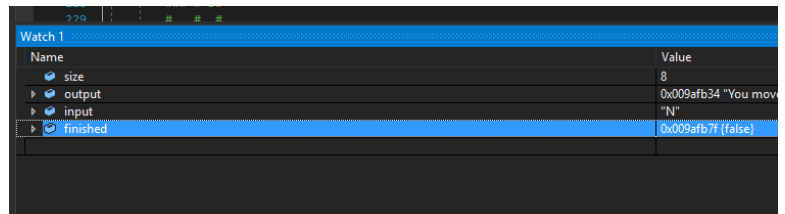
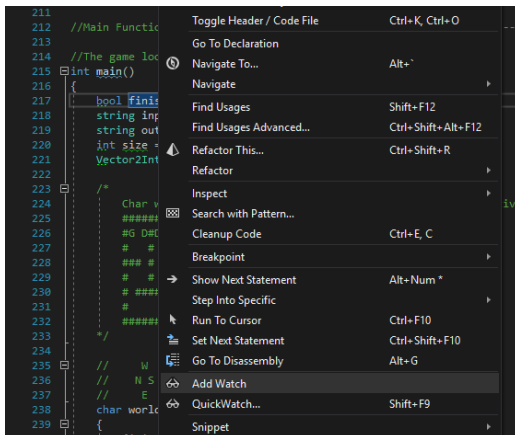


Once your program is running with debugging active, when it reaches a breakpoint, you may wish to inspect the values of variables in the code. There are several ways to do this. Firstly, to look at the values of variables present on the line of the breakpoint, click on “Debug > Windows > Autos” (left). This will open the Autos tab (below). (Once opened, this tab, if you don’t click its “X” button to exit it, will be available in a tray of tabs at the bottom of the screen (below, bottom left).)





To view the values of all variables in the method currently being executed, you will instead want to click on “Debug > Windows > Locals”. For example, my breakpoint above is in the Update method (you can see which method the line you’re typing on is in at top-centre-right), so the Locals window is showing the values of all variables passed to and defined in Update.



To view the values of specific variables throughout the execution of your program, regardless of where they are in the code compared to which line of code is being executed / paused on, right click on the variable in question and click “Add Watch” (above). This will open up the “Watch” tab and add the variable to the list of variables it is tracking (above-right). To stop tracking a variable, right click on it in the “Watch” tab and click “Delete Watch” (right).