Visual Studio Code Setup Guide

EEEN30222 Computer Systems Architecture

1 Prerequisites

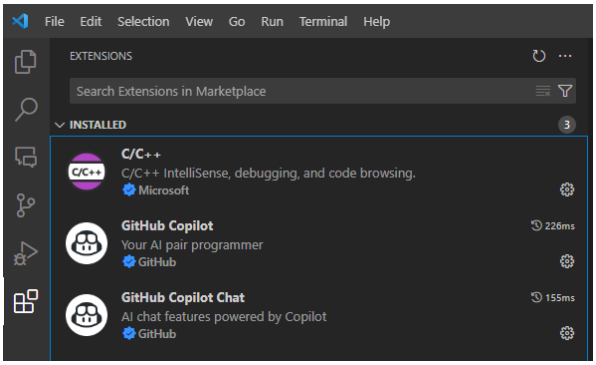
If you are working on a University Computer, skip to Section 3.

This guide is a simplified version of Microsoft's own setup guide for C++, with a few elements changed to reflect the needs of this course. Before setting up Visual Studio Code (VSC), you will need a C/C++ compiler. The recommended compiler is gcc, which is already installed on the university systems via the developer environment mingw. To install it on your own Windows system, follow the full installation process documented in this video. One method to install mingw on an Apple device is to install MacPorts and then run `sudo port install mingw-w64`. Although they will try to help, it is not possible for the Lab Demonstrators to learn the details of every possible OS and computer setup. We ask you to try and use the University computers if you are having trouble with your own system.

2 Installing VSC and Extensions

First, download VSC. Take care not to download Visual Studio, as that is a separate program that we do not need. The installer can be run with the default settings. Once the installation is complete, you will likely be greeted with some release notes. You can close these, and the welcome screen.

VSC has a wide array of extensions available to enable operation in many languages and platforms. Since this course uses the C programming language, the associated extensions may prove useful. Go to the extensions tab on the left and install the C/C++ extension by Microsoft. You may also wish to install other extensions. A variety of tools are available to assist with git integration along with several AI code assistants.

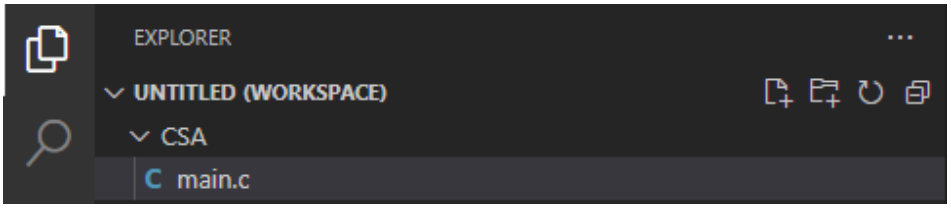


3 Creating and configuring the project

VSC operates using a workspace which is simply an open folder on the PC. Go to the explorer tab on the left, and select `Open Folder`. This will open file explorer, where you can choose the location that your files will be saved.

NOTE: If you are on a university PC, navigate to `This PC > home$(\nask.man.ac.uk)(P:)` and put your folder there[1].

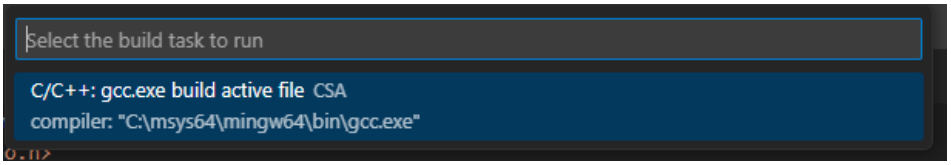
Create a blank folder with any name, and open it. You may have to indicate that you trust this folder. Since you created it, and it is empty, it can be trusted. Now, in VSC, right click in the explorer panel to create a new file, and call it `main.c`. VSC will automatically recognise it as a C file, and highlight keywords to make the code easier to read. Your explorer should now look like this:



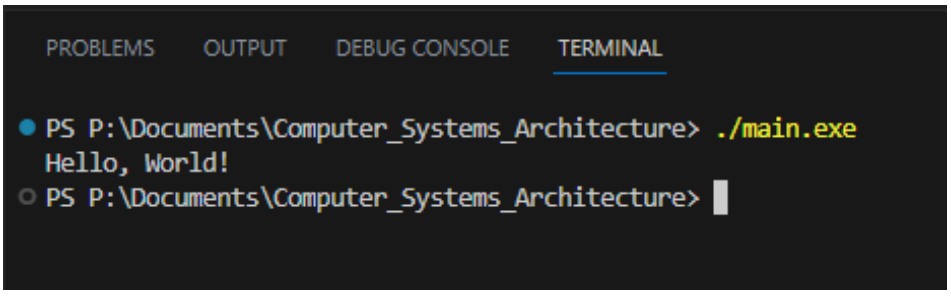
Copy the following code into `main.c` and save the file:

```
#include <stdio.h>
int main() {
   printf("Hello, World!");
   return 0;
}
```

Now, the file needs compiling. When using compile or run commands, they are applied to the active file, so select `main.c` in the file explorer. Then, go to `Terminal > Run Build Task` or press Ctrl + Shift + B. If mingw was installed correctly, the following dialogue should appear as VSC finds the correct compiler.



Don't worry if there are multiple options, it just means that VSC has found multiple C++ compilers. Select `C/C++: gcc.exe build active file`. In the Terminal, you will see the build command running and the build outputs. In your file explorer, if the build was successful, you should see a file called `main.exe`. If you type `./main.exe` into the terminal (this can be found below the code editor window), you should see the output `Hello, World!`. If you do not have a terminal visible, click `Terminal > New Terminal`.



[1] If you navigate this way, VSC sets the directory to `P:\...`, whereas other navigation approaches set it to `\\nask.man.ac.uk\home$\...` which causes errors.

4 Debugging

If you wish, this is enough to be getting on with. However, manually compiling and typing `./main.exe` can get a bit tiresome. VSC enables compiling, running, and debugging a file with one button. Go to `Terminal > Configure Default Build Task`. In the same dialogue as before, select `C/C++: gcc.exe build active file`. This creates a new folder and file, `/.vscode/tasks.json`. In there are the configuration parameters to build the file. Now, pressing Ctrl + Shift + B will attempt to compile the active file. Try it with `main.c` open. The compilation will fail if you have `tasks.json` selected, as this is not a C file.

For running and debugging, go to the Run and Debug tab on the left. With `main.c` open, select `create a launch.json file`, then in the dialogue that appears, select `C++ (GDB/LLDB)`. This will again create a new config file, but this time we need to edit it. Paste the code on the next page into your `launch.json`.

The key changes from the default `launch.json` are: the program name has been set, the debugger path removed such that VSC will find it automatically, and a pre-launch task has been added, linked to the build task we set up earlier. This means that, when you run your code, VSC will compile it automatically first. Now, go to the `Run and Debug` tab on the left, and click the green run icon at the top (the drop-down should say `(gdb)Launch`). Alternatively, press F5. At the bottom of the screen, the debug console contains information about the process, and the program output is in the terminal as before. You can now add breakpoints to your code (by clicking next to the line numbers), and the debugger will stop at the break point. The controls at the top can then be used to step through your code one line at a time, and you can watch the values of your variables in the left panel.

Pay attention to the `args` line in the `launch.json` file. The coursework requires you to make use of command line arguments, and if you are running your file using this setup, your command line arguments must go on that line in `launch.json`.

4.1 launch.json configuration parameters.

```
{
"configurations": [
    {
        "preLaunchTask": "C/C++: gcc.exe build active file",
        "name": "(gdb) Launch",
        "type": "cppdbg",
        "request": "launch",
        "program": "${fileDirname}\\${fileBasenameNoExtension}.exe",
        "args": [],
        "stopAtEntry": false,
        "cwd": "${fileDirname}",
        "environment": [],
        "externalConsole": false,
        "MIMode": "gdb",
        "setupCommands": [
            {
                "description": "Enable pretty-printing for gdb",
                "text": "-enable-pretty-printing",
                "ignoreFailures": true
            },
            {
                "description": "Set Disassembly Flavor to Intel",
                "text": "-gdb-set disassembly-flavor intel",
                "ignoreFailures": true
            }
        ]
    }
]
}
```

V1.0 26/02/24

V1.1 27/01/25

Christopher Blum

This document was created in Latex using Overleaf, the source code is available on Github.