C++ for VSCode

Updated: March 2020

Contents

[Installing C++ for VSCode]
[Running and debugging C++ scripts without CodeRunner]
[Appendix A: launch.json example]
[Appendix B: Compiling and running code without the debugger play button]

If stuck, see:

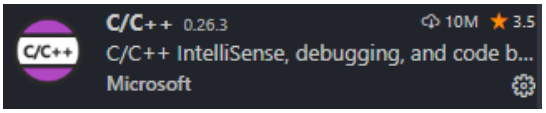
https://code.visualstudio.com/docs/cpp/config-mingw
https://www.youtube.com/watch?v=DIw02CaEusY

Installing C++ for VSCode:

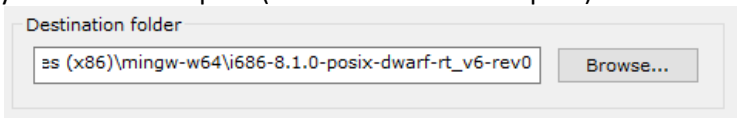
More info: https://code.visualstudio.com/docs/cpp/config-mingw

Steps:

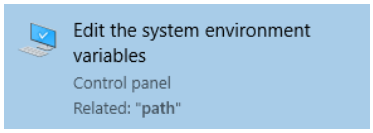
1. Download the C++ VSCode extension



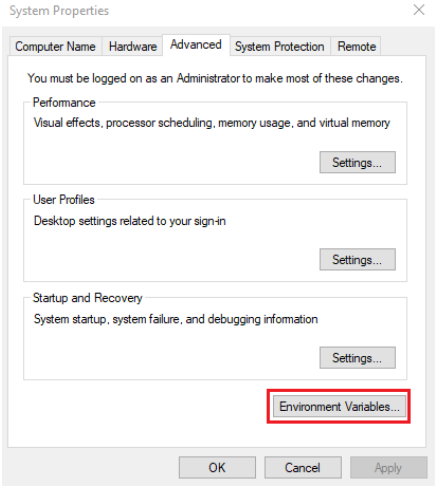
2. Install MinGW:
   a. https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win32/Personal%20Builds/mingw-builds/installer/mingw-w64-install.exe/download
   b. Take note of your installation path (needs to be added to path)



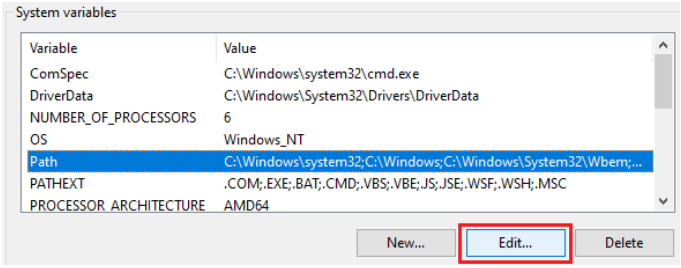
3. Add the "bin" directory of your MinGW installation to path
   a. Search path in windows



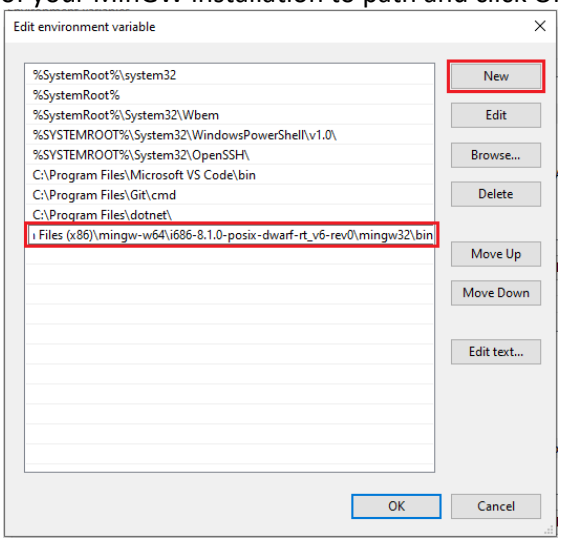
   b. In the "Advanced" tab, click the "Environment Variables" button



   c. Under "System variables" edit "Path"



d. Add the “bin” folder of your MinGW installation to path and click OK



4. Check your MigGW installation
   a. Run `g++ --version` in command prompt

```
g++ (i686-posix-dwarf-rev0, Built by MinGW-W64 project) 8.1.0
Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

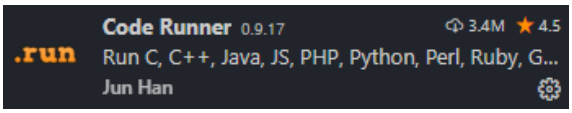
   b. Run `gdb --version` in command prompt

```
GNU gdb (GDB) 8.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-w64-mingw32".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
```

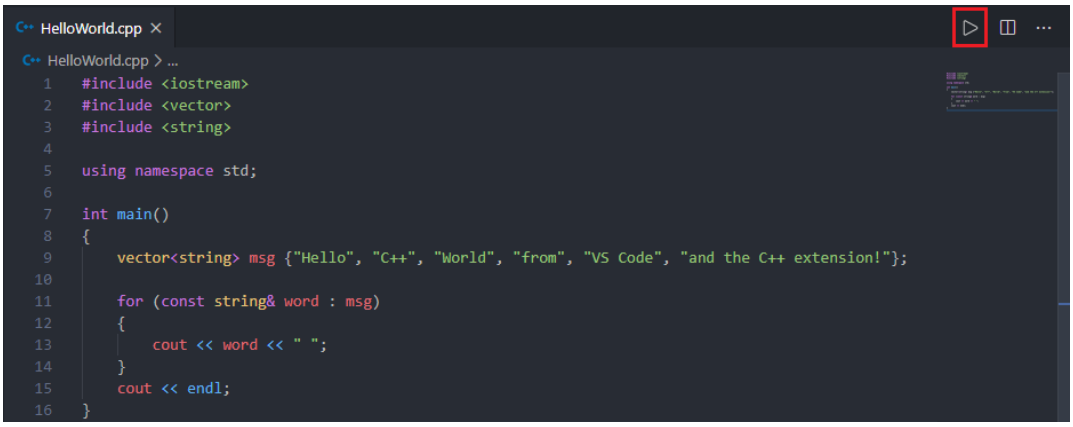
5. If already open, restart VSCode. Open a folder in VSCode and make a HelloWorld.cpp script using the following: (A copy of this script can be found in the “more info” link at the top)

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main()
{
    vector<string> msg {"Hello", "C++", "World", "from", "VS Code", "and the C++
extension!"};

    for (const string& word : msg)
    {
        cout << word << " ";
    }
    cout << endl;
}
```

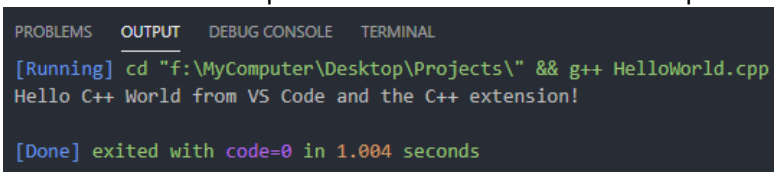
6. Download the Code Runner extension



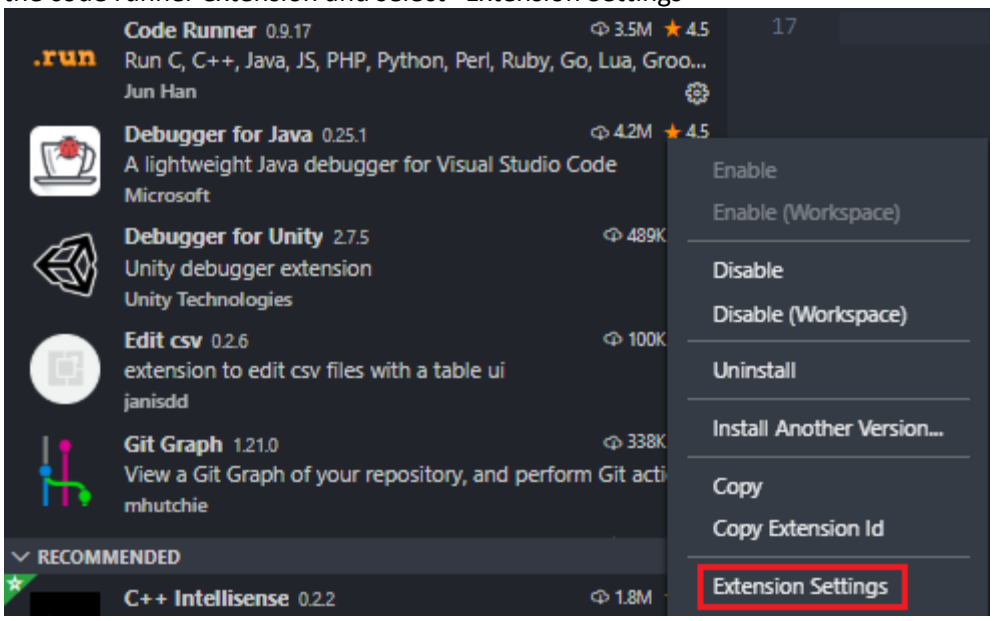
7. Click the Code Runner run button



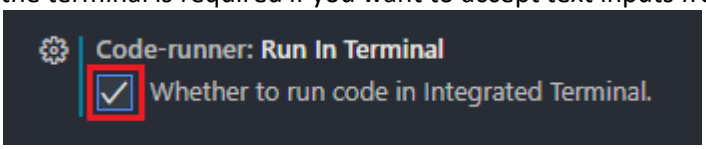
8. Celebrate the fact that Code Runner has optimized the tedious C++ code compilation and running process



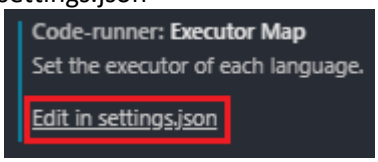
9. However, a few more steps are required to run more complex OOP code with multiple .cpp files. Click the gear symbol on the code runner extension and select “Extension Settings”



10. Enable “Run In Terminal”. This runs code through the “Terminal” tab rather than through the “Output” tab. Running code through the terminal is required if you want to accept text inputs from a user during runtime.



11. Under Executor Map, select “Edit in settings.json”



12. Copy and paste the following into the JSON file as shown and save:
    a. These lines tell CodeRunner to compile all .cpp files in the directory rather than just the active file.

```
"code-runner.executorMap": {
        "cpp": "cd $dir && g++ -o $fileNameWithoutExt *.cpp && $dir$fileNameWithoutExt"
    }
```

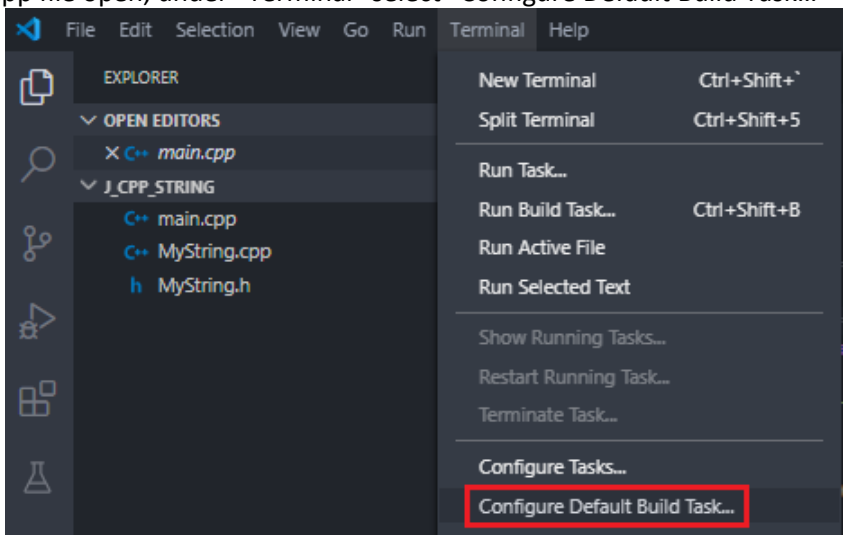
```
{
    "editor.suggestSelection": "first",
    "java.help.firstView": "gettingStarted",
    "java.home": "C:\\Users\\sirpa\\AppData\\Local\\Programs\\AdoptOpenJDK",
    "python.jediEnabled": false,
    "python.pythonPath": "C:\\Users\\sirpa\\Anaconda3",
    "vsintellicode.modify.editor.suggestSelection": "automaticallyOverrodeDefaultValue",
    "window.zoomLevel": -1,
    "workbench.colorTheme": "Atom One Dark",
    "workbench.iconTheme": "material-icon-theme",
    "git.autofetch": true,
    "java.configuration.checkProjectSettingsExclusions": false,
    "python.dataScience.sendSelectionToInteractiveWindow": true,
    "C_Cpp.updateChannel": "Insiders",
    "code-runner.executorMap": {
        "cpp": "cd $dir && g++ -o $fileNameWithoutExt *.cpp && $dir$fileNameWithoutExt"
    }
}
```

13. The above setup does not need to be repeated. It will now work for all future projects on your device. CodeRunner is a useful, low effort way to run C++ in VSCode. However, using this method, you will not be able to debug. To gain full C++ utility in VSCode, I suggest you see the following section.

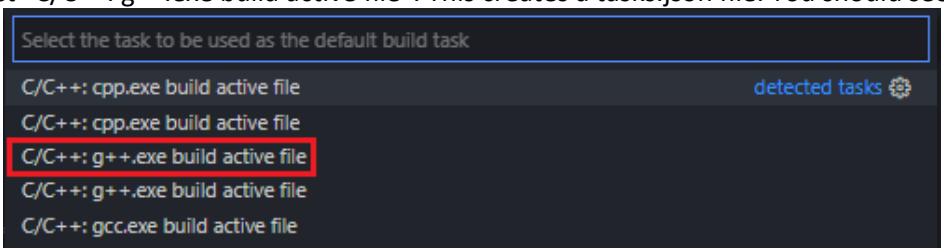
Running and debugging C++ scripts without CodeRunner

CodeRunner is an easy, yet shallow way to run C++ in VSCode. When you're doing larger projects, you're going to want the ability to debug. Running and debugging code without CodeRunner requires some initial setup (i.e. tasks.json and launch.json files must be created). However, after setup, the process for compiling, running, and debugging code is quick. This process is only required once per project (i.e. once per ".vscode" folder in your directory)

1. Important: Ensure your VSCode directory is set to the location of your .cpp files
2. Create a tasks.json file for building your code
    a. With a .cpp file open, under "Terminal" select "Configure Default Build Task…"



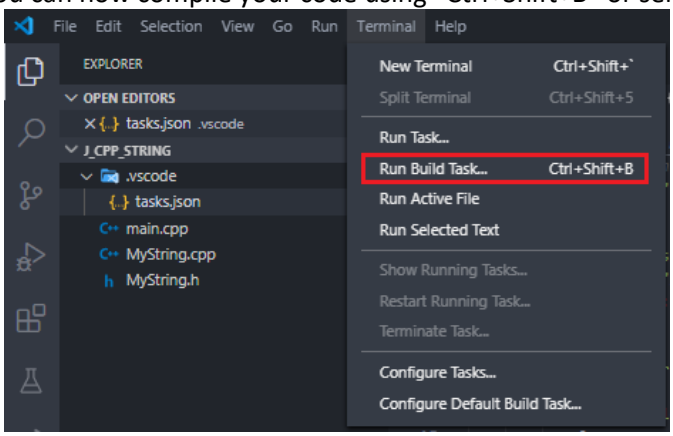
    b. Select "C/C++: g++.exe build active file". This creates a tasks.json file. You should see it in your directory.



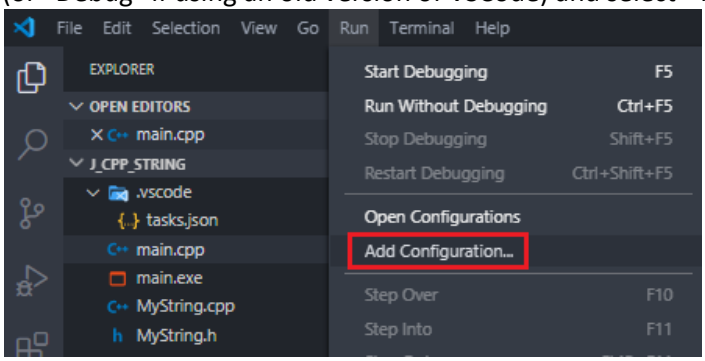
    c. Change the `"${file}"` entry under "args" to `"${workspaceFolder}\\*.cpp"` and save. This tells VSCode to compile all C++ files in the directory rather than just the active file.

```
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
            "type": "shell",
            "label": "g++.exe build active file",
            "command": "C:\\Program Files (x86)\\mingw-w64\\i686-8.1.0-posix-dwarf-rt_v6-rev0\\mingw32\\bin\\g++.exe",
            "args": [
                "-g",
                "${workspaceFolder}\\*.cpp",
                "-o",
                "${fileDirname}\\${fileBasenameNoExtension}.exe"
            ],
            "options": {
                "cwd": "C:\\Program Files (x86)\\mingw-w64\\i686-8.1.0-posix-dwarf-rt_v6-rev0\\mingw32\\bin"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            }
        }
    ]
}
```

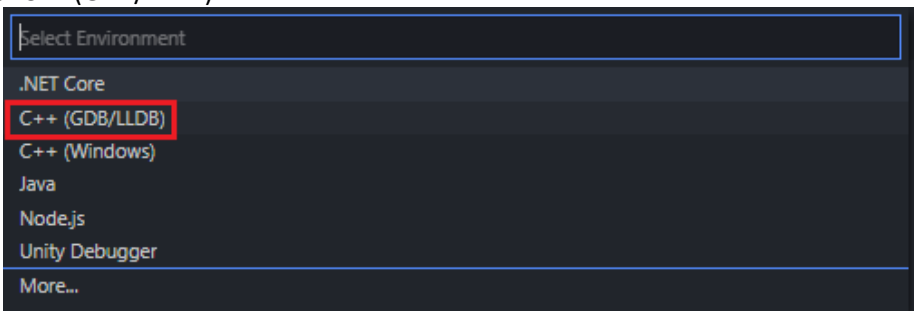
i. Note: You can now compile your code using “Ctrl+Shift+B” or selecting “Run Build Task”



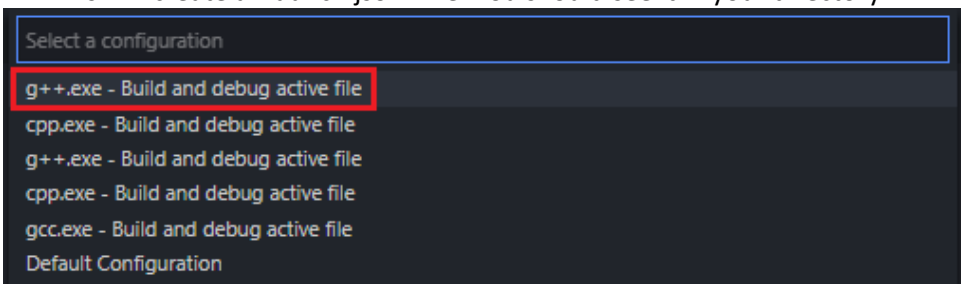
3. Create a launch.json file to configure the debugger
   a. Click on “Run” (or “Debug” if using an old version of VSCode) and select “Add Configuration”



   b. Select “C++ (GDB/LLDB).

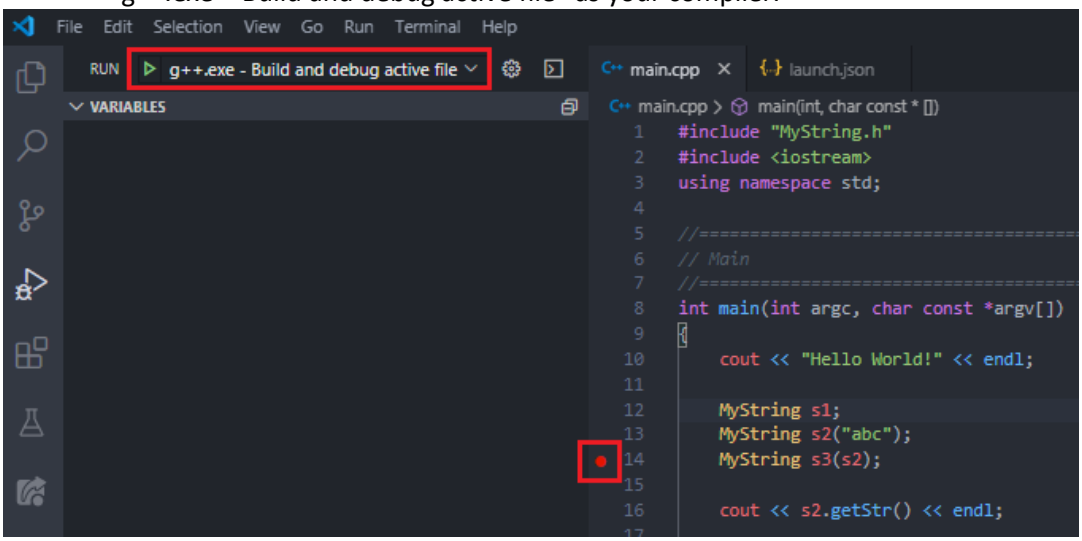


   c. Select “g++.exe – Build and debug active file”.
      i. This will create a “launch.json” file. You should see it in your directory.

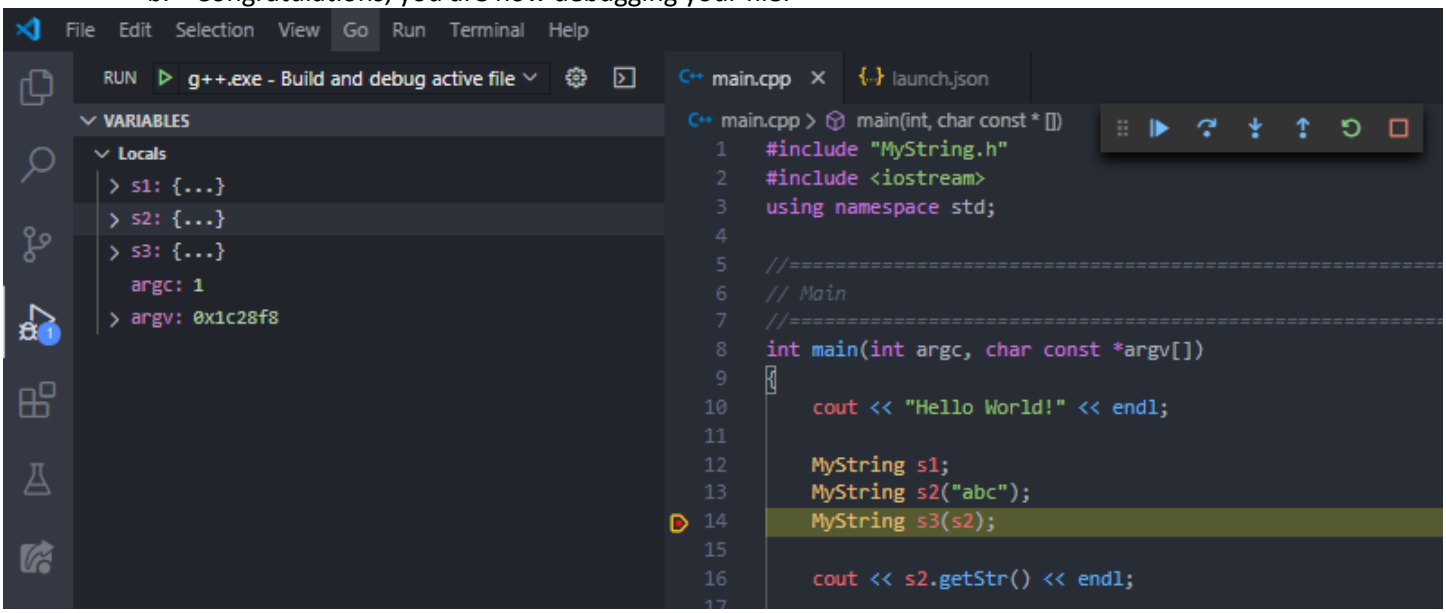


      ii. Note that this option automatically sets the paths to your executable file and your mingw32 directory. An example of a launch.json file can be found in Appendix A.

2. Debug your file
    a. Add breakpoints to your code. Then, on the debug tab, ensure your build is set to “g++.exe – Build and debug active file” and press the play button (or F5).
        i. Alternatively, you can right click the script and select “Build and Debug Active File” then select “g++.exe – Build and debug active file” as your compiler.



    b. Congratulations, you are now debugging your file.

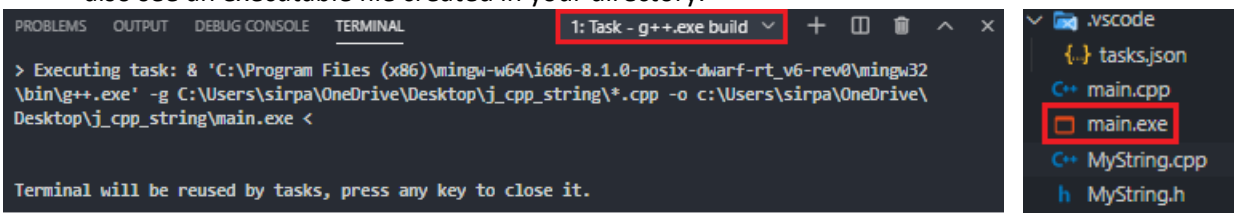


Appendix A: launch.json example

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "g++.exe - Build and debug active file",
            "type": "cppdbg",
            "request": "launch",
            "program": "${fileDirname}\\${fileBasenameNoExtension}.exe",
            "args": [],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "miDebuggerPath": "C:\\Program Files (x86)\\mingw-w64\\i686-8.1.0-posix-dwarf-rt_v6-rev0\\mingw32\\bin\\gdb.exe",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                }
            ],
            "preLaunchTask": "g++.exe build active file"
        }
    ]
}
```

Appendix B: Compiling and running code without the debugger play button

1. Create a tasks.json file for building your code (if you haven't already)
2. Compile and run your code
    a. Press "Ctrl+Shift+B" to compile your code. A "Task – g++.exe build" terminal should launch. You should also see an executable file created in your directory.



    b. Press "+" to open a new terminal and run your executable file using `.\[filename].exe`