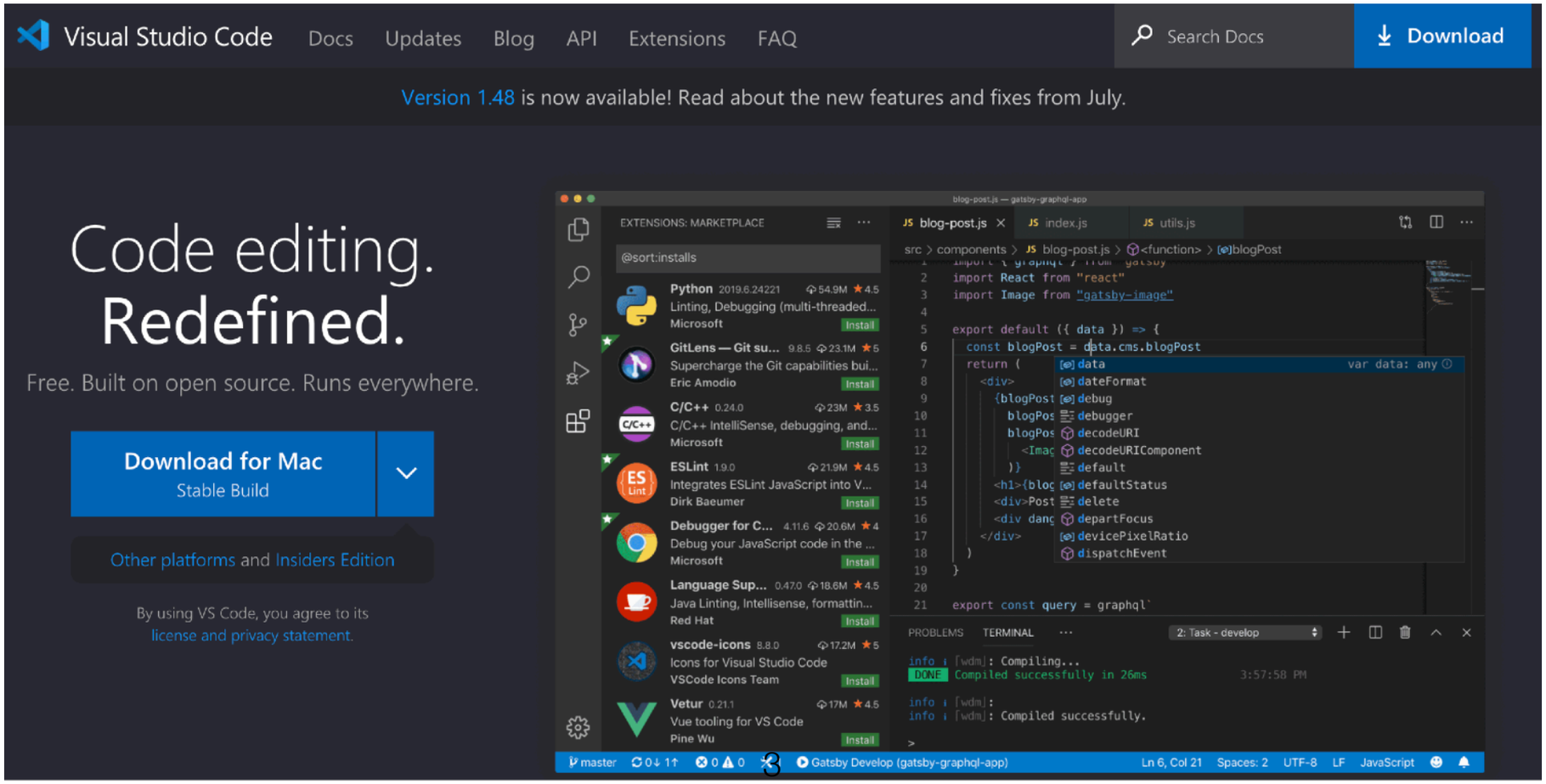
# PL Practice 1

# Prepare for Practice Sessions

- We've already gone through some tools we will use for our programming language practices.
- Your main tasks to prepare practices are installing various compilers (or interpreters) for languages, and also setup your development environment.

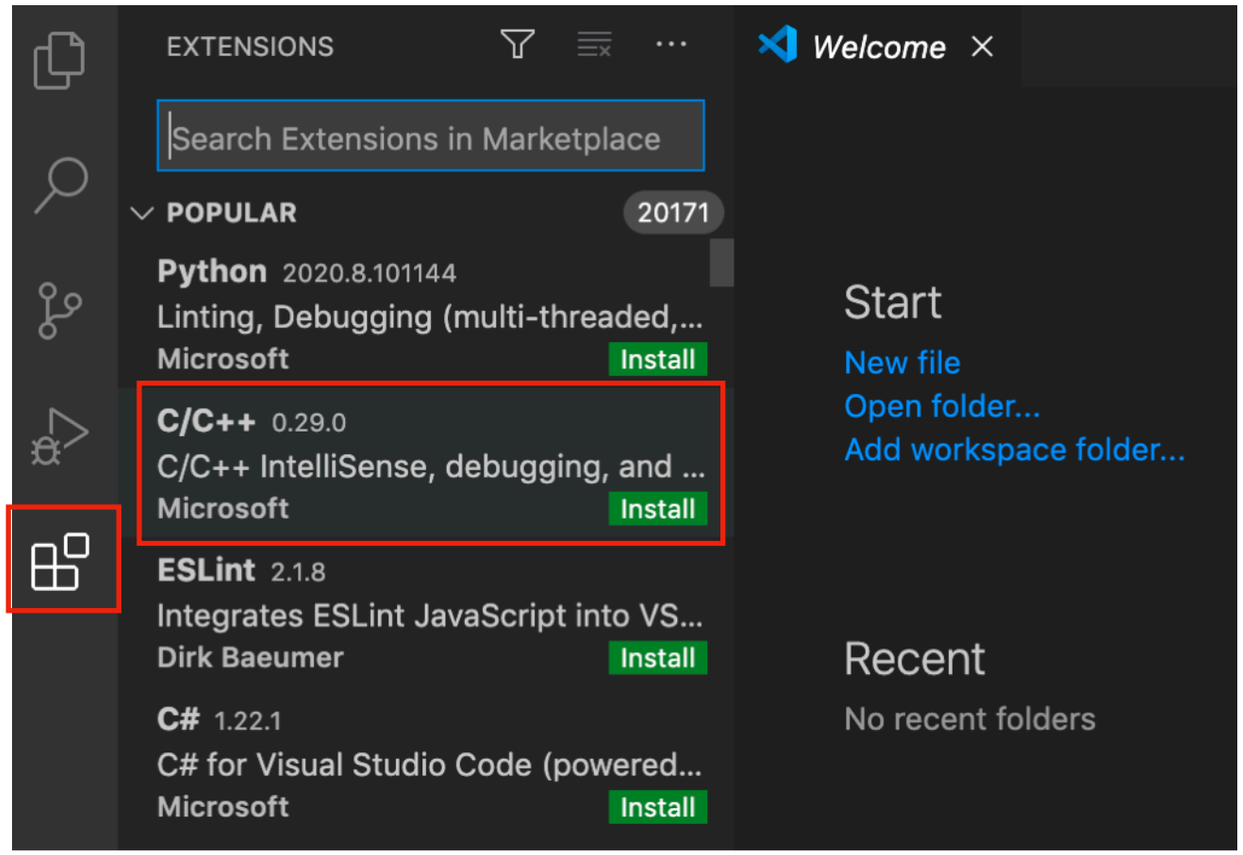
# VSCode

- Visual Studio Code: Free IDE developed by Microsoft.
- Support various OS - Windows, Mac, Linux
- Using Extensions to support various programming languages.



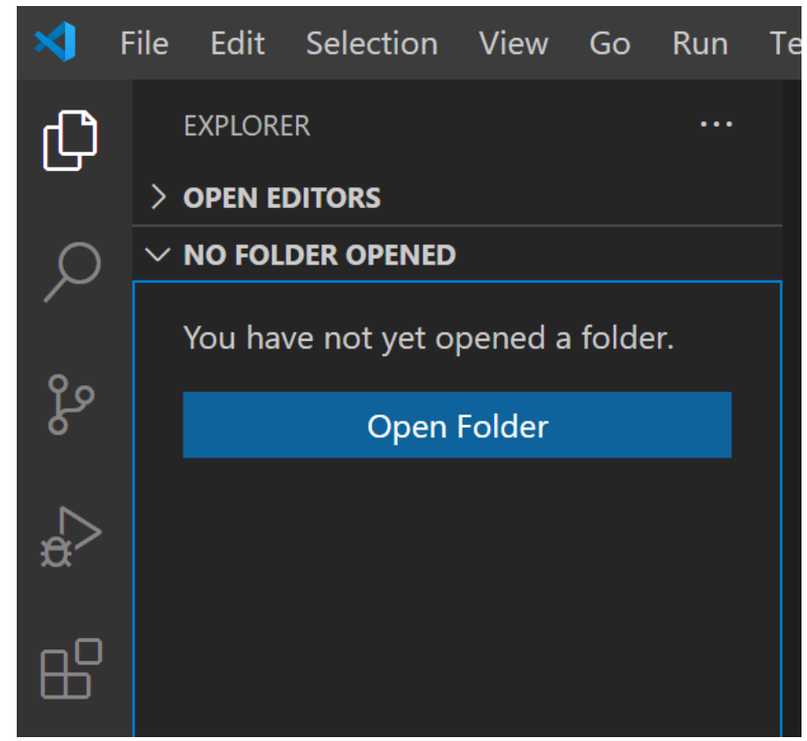
# Installing Extensions

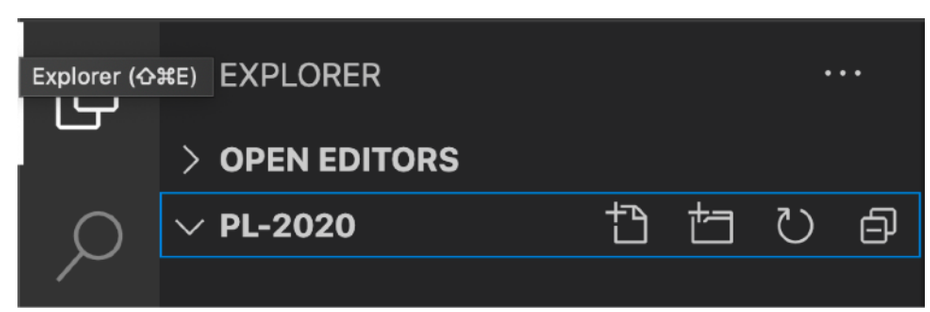
- Let's try to install C/C++ extension.
- Assuming that you've already installed C/C++ compilers in previous courses.
- If not, it would be better to install them first.



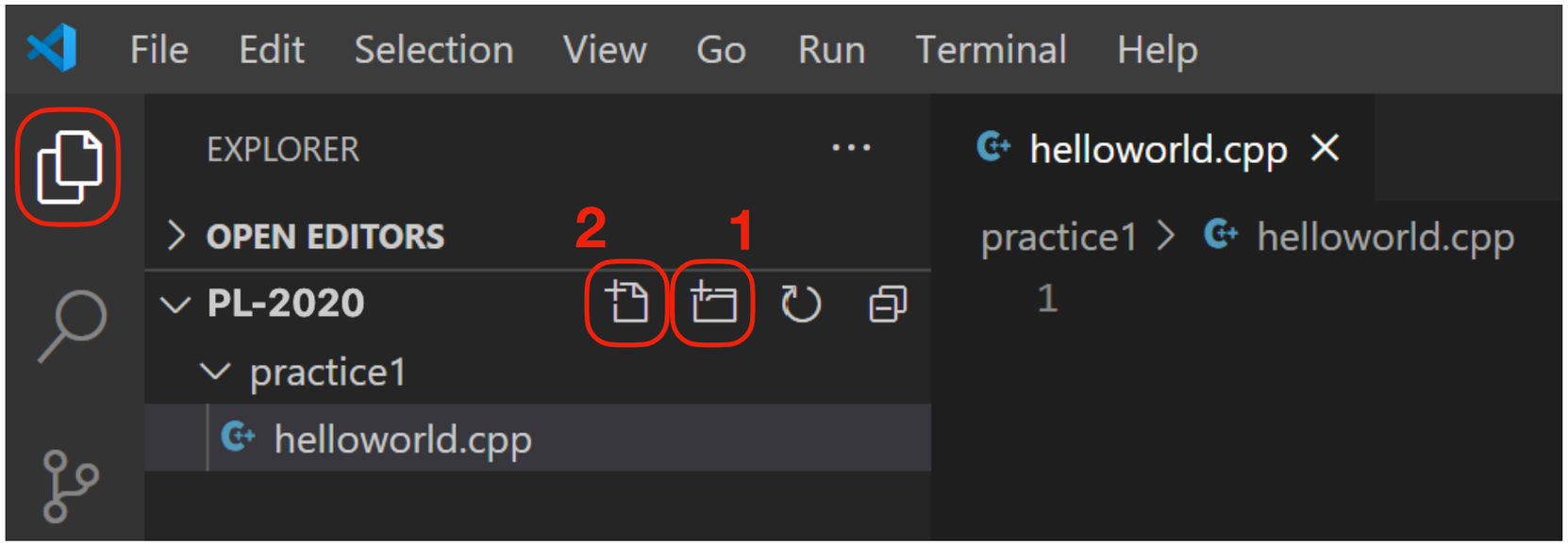
# Workspace

- To create a new workspace, click '**Open Folder**' button.
- Then there will be a window to create a new folder.
- Type the folder's name (e.g., PL-2020) and create it.
- You can see the workspace is created in Explorer.



# Let’s Create a File




- In Explorer, Click the second icon(1) to create a new folder.
- Click the first icon(2) and create a new file named helloworld.cpp.
- You can see that helloworld.cpp file is opened on the right automatically.

# Build/Run Configuration

- To build and run a file, you need to configure Build and Run tasks.
- VSCode simply runs the command in a configuration file ‘tasks.json’.
- Here is an example of default tasks.json for Windows.
- You can also find various examples of tasks.json on Google.

helloworld.cpp  tasks.json

.vscode > tasks.json > [ ] tasks > {} 0

```
1  {
2      "version": "2.0.0",
3      "tasks": [
4          {
5              "type": "shell",
6              "label": "C/C++: g++.exe build active file",
7              "command": "C:\\MinGW\\bin\\g++.exe",
8              "args": [
9                  "-g",
10                 "${file}",
11                 "-o",
12                 "${fileDirname}\\${fileBasenameNoExtension}.exe"
13             ],
14             "options": {
15                 "cwd": "${workspaceFolder}"
16             },
17             "problemMatcher": [
18                 "$gcc"
19             ],
20             "group": {
21                 "kind": "build",
22                 "isDefault": true
23             }
24         }
25     ]
26 }
```

# JSON File

- Each item is a name/value pair.
- “name”: value
- a value can be an <object>, <array>, string, number, true, false, null.
- <object> is enclosed by { } - unordered list of name/value pairs.
- <array> is enclosed by [ ] - ordered list of values.
- More details: www.json.org

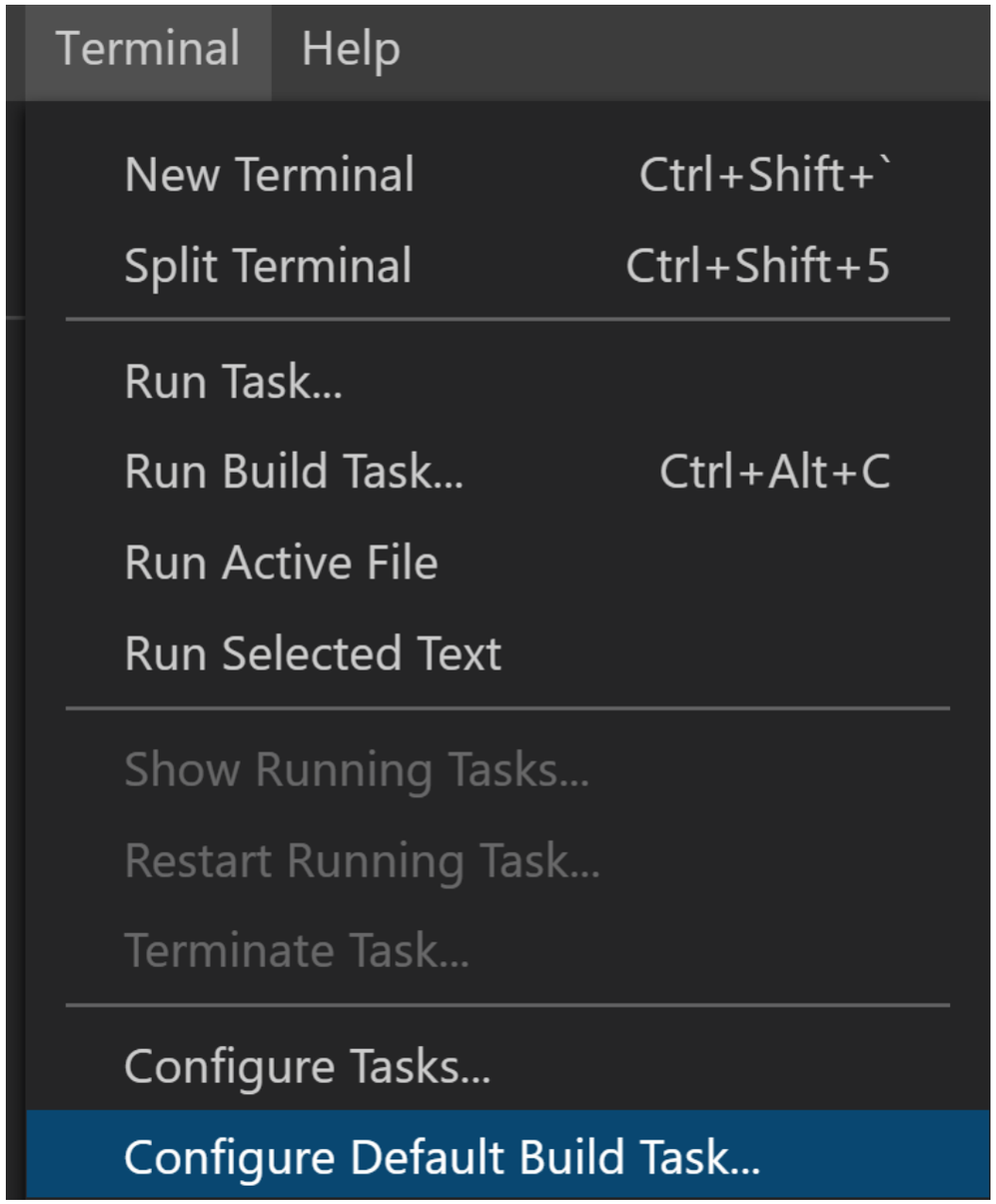
loworld.cpp  {} tasks.json ×

de > {} tasks.json > [ ] tasks > {} 0

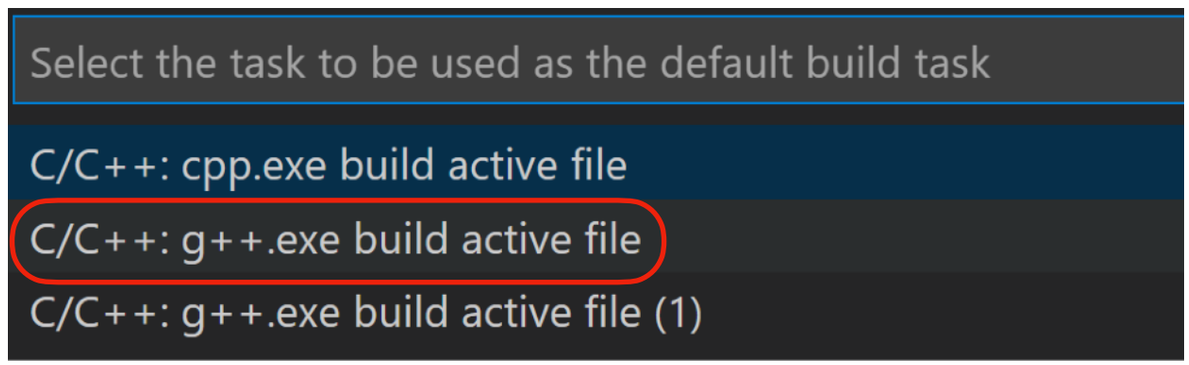
```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "shell",
            "label": "C/C++: g++.exe build active file",
            "command": "C:\\MinGW\\bin\\g++.exe",
            "args": [
                "-g",
                "${file}",
                "-o",
                "${fileDirname}\\${fileBasenameNoExtension}
            ],
            "options": {
                "cwd": "${workspaceFolder}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            }
        }
    ]
}
```

# Build/Run Configuration (1)

- To configure, select helloworld.cpp first.
- Select Terminal → Configure Default Build Task.
- You will see a list of configurations related to C/C++.



# Build/Run Configuration (2)




- List shown could be slightly different.
- Select the one with g++.exe.
- The second item is the configuration for building active file (i.e., selected file) with g++.exe.

# Build/Run Configuration (3)

- tasks.json is automatically created under .vscode folder and opened.
- If you already installed compilers properly, you can see the values are already filled.
- label is the name shown on the list, command is the path to your compiler.
- If it's empty or incorrect, please modify it correctly.
- args is short for arguments for execution.
  - -g: including debugging information.
  - ${file}: selected file name.
  - -o ${fileDirname}~~~: designate the name of execution file.

helloworld.cpp  {} tasks.json ×

.vscode > {} tasks.json > [ ] tasks > {} 0

```
1  {
2      "version": "2.0.0",
3      "tasks": [
4          {
5              "type": "shell",
6              "label": "C/C++: g++.exe build active file",
7              "command": "C:\\MinGW\\bin\\g++.exe",
8              "args": [
9                  "-g",
10                 "${file}",
11                 "-o",
12                 "${fileDirname}\\${fileBasenameNoExtension}.exe"
13             ],
14             "options": {
15                 "cwd": "${workspaceFolder}"
16             },
17             "problemMatcher": [
18                 "$gcc"
19             ],
20             "group": {
21                 "kind": "build",
22                 "isDefault": true
23             }
24         }
25     ]
26 }
```

# Build/Run Configuration (4)

- Current configuration is only for Build task.
- For convenience, let's add Run task.
- Add another object for Run task after Build task - check the yellow '*' for position.
- For Mac/Linux, you can use the following command at the bottom.

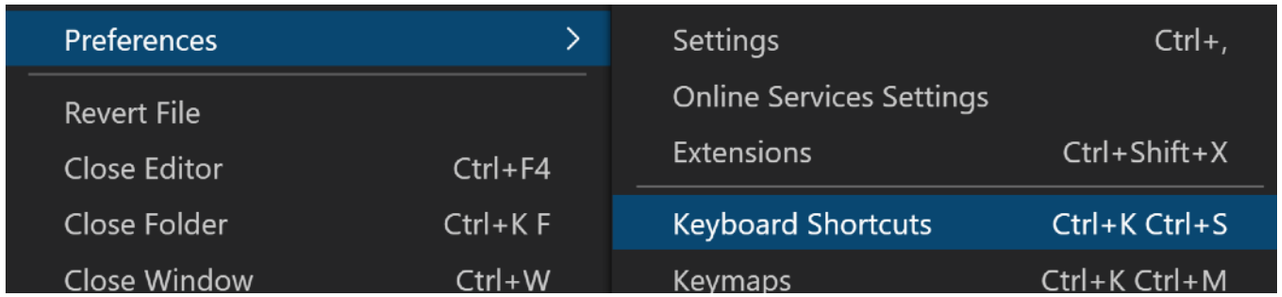
```
            "isDefault": true
        }
*},
 {
        "label": "Run C++",
        "command": "cmd",
        "args": [
            "/C",
            "${fileDirname}\\${fileBasenameNoExtension}.exe"
        ],
        "group": {
            "kind": "test",
            "isDefault": true
        }
    }
]
```

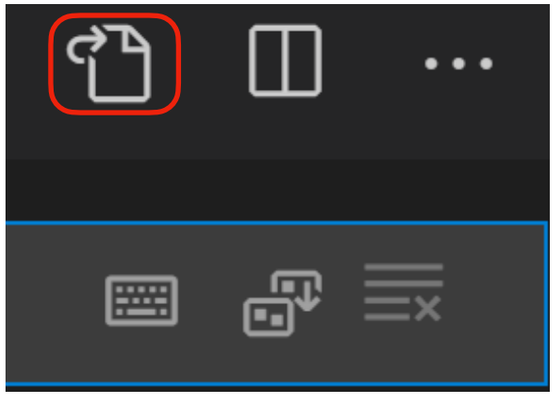
**For Mac/Linux**

```
"command": "${fileDirname}/${fileBasenameNoExtension}.exe",
```

# Build/Run Configuration (5)

- Next step is for shortcuts.
- Press Ctrl+K, Ctrl+S, then you can see **K**eyboard **S**hortcut configuration.
- Or you can simply choose a menu item.
- Find the icon marked with red rectangle and click it.



# Build/Run Configuration (6)

```
// Place your key bindings in this file to override the defaults
[
    //Build
    { "key": "ctrl+alt+b", "command": "workbench.action.tasks.build" },
    //Run
    { "key": "ctrl+alt+r", "command": "workbench.action.tasks.test" }
]
```

- Another JSON file (keybindings.json) for shortcut setting will be opened.
- Put above contents between [ ] and save.
- The first is a shortcut for Build, and the second is for Run.

# Hello World

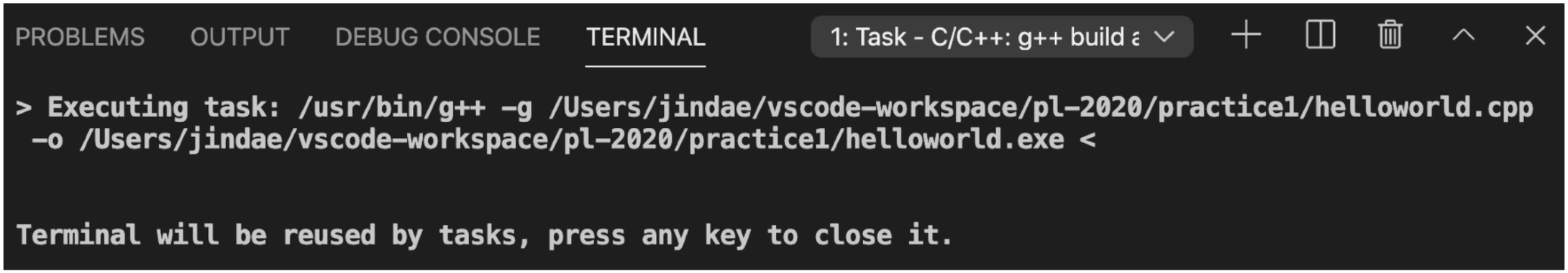
- Let's write the first program with VSCode.
- On the right, it is a typical 'Hello World' program.
- You can put any message you want to print.
- But you should write a program to print a message, to check the attendance for this practice.

helloworld.cpp ×

practice1 > helloworld.cpp > main()

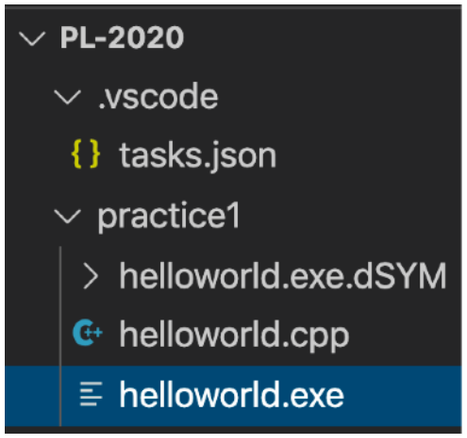
```cpp
#include<iostream>
using namespace std;

int main() {
    cout << "Hello World!!\n";
    return 0;
}
```

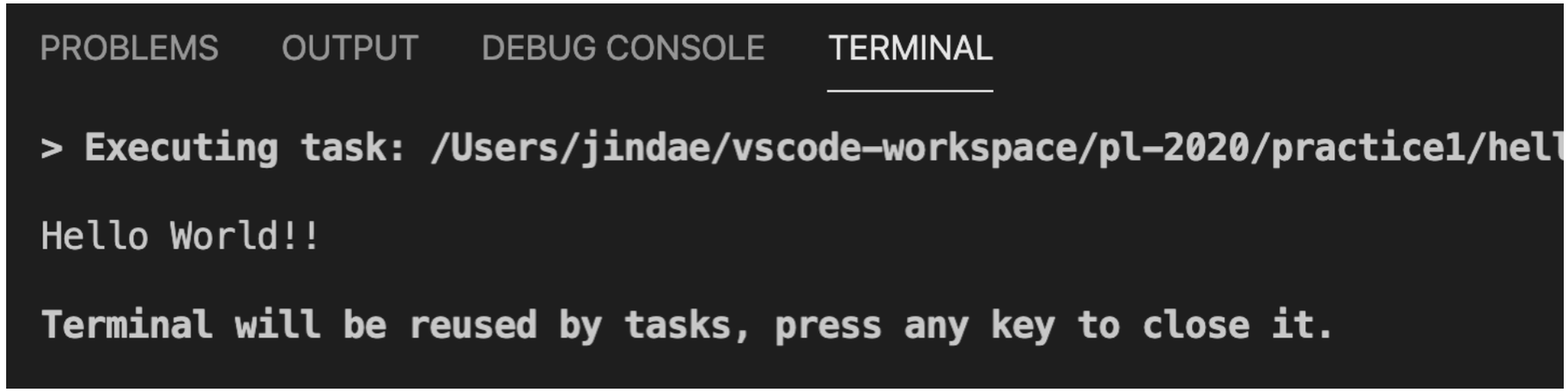
# Build Hello World




- Now let's build and run the file.
- Select helloworld.cpp.
- Press ctrl+alt+B (or whatever you configured) to build.
- You can see the compiler's running in Terminal.

# Run Hello World




- If the build was successful, you can see an executable file is created (name could be different based on your configuration).
- Select helloworld.exe (the created executable file), and press ctrl+alt+r (or whatever you configured for Run).
- If you see the message you type in Terminal, you finished your first practice.
- *.dSYM stores debug symbols in macOS. Just ignore it.

# For Attendance

- Capture the screen that “Hello World!” is printed.
- If you’re not using VSCode, write any program to print “Hello World!” message, and run it in your IDE.
- Capture the screen of your IDE’s terminal.
- Submit your captured image in e-Class for attendance verification.

# Summary

- Today’s practice is simply about installing VSCode.
- We will use VSCode for practices.
- But, we will also use some other IDE or REPL tools too.