2_Linux 下 C++开发

第一步：安装依赖环境

sudo apt install build-essential

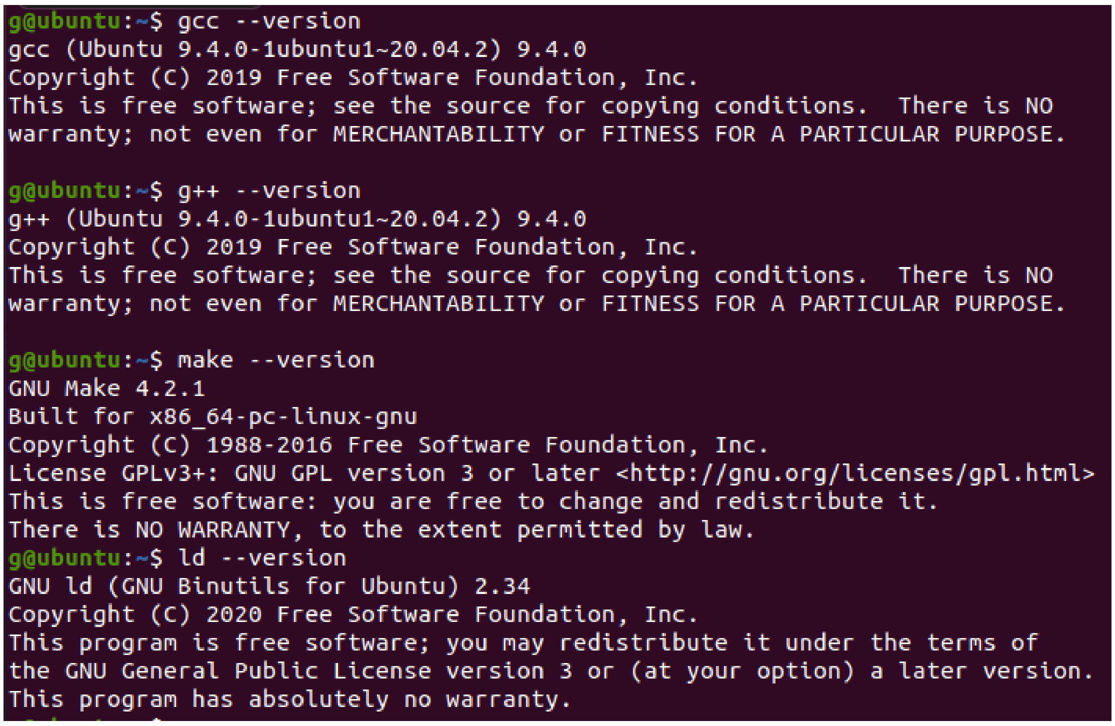
安装完成后，可以通过以下命令验证 GCC 和 G++ 是否安装成功：

gcc --version

g++ --version

make --version

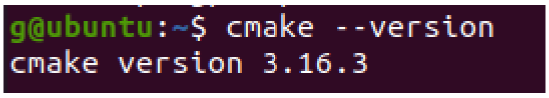
ld --version



```
g@ubuntu:~$ gcc --version
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
Copyright (C) 2019 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.

g@ubuntu:~$ g++ --version
g++ (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
Copyright (C) 2019 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.

g@ubuntu:~$ make --version
GNU Make 4.2.1
Built for x86_64-pc-linux-gnu
Copyright (C) 1988-2016 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.
g@ubuntu:~$ ld --version
GNU ld (GNU Binutils for Ubuntu) 2.34
Copyright (C) 2020 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) a later version.
This program has absolutely no warranty.
```

sudo apt install cmake

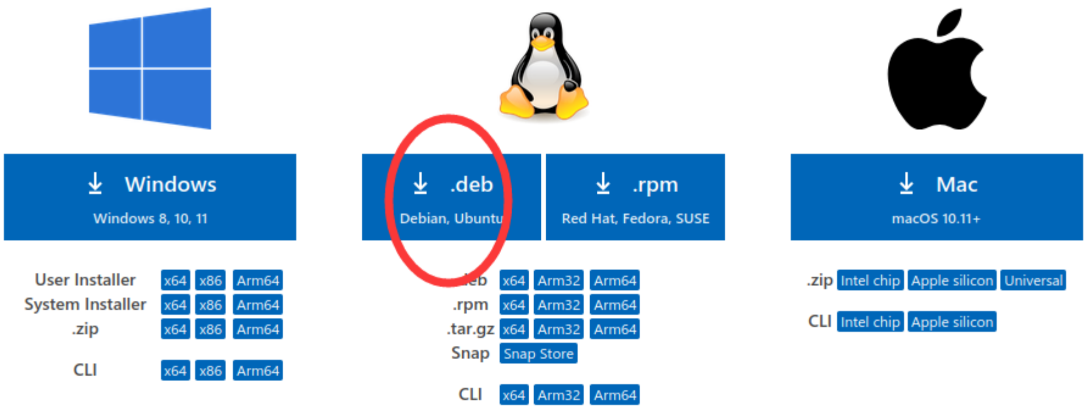
cmake --version

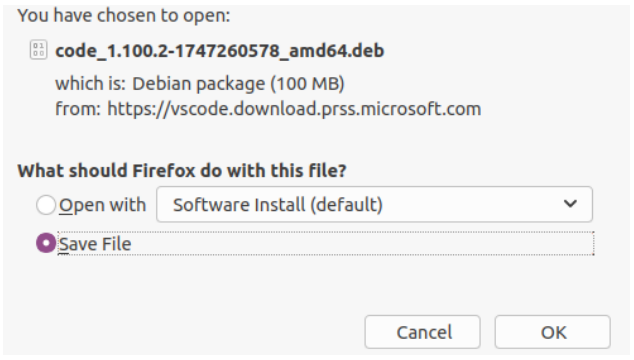


```
g@ubuntu:~$ cmake --version
cmake version 3.16.3
```

第二步：安装 VScode

https://code.visualstudio.com/Download





You have chosen to open:

code_1.100.2-1747260578_amd64.deb

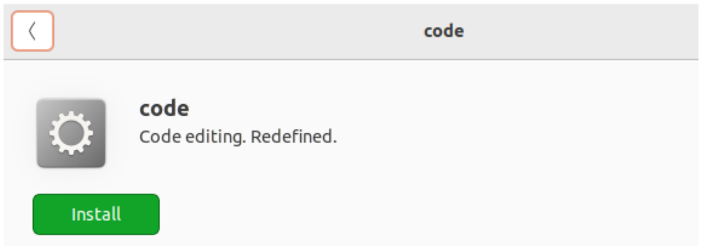
which is: Debian package (100 MB)
from: https://vscode.download.prss.microsoft.com

What should Firefox do with this file?

Open with Software Install (default) [✓ Save File]

Cancel OK

双击下载的 code_1.100.2-1747260578_amd64.deb

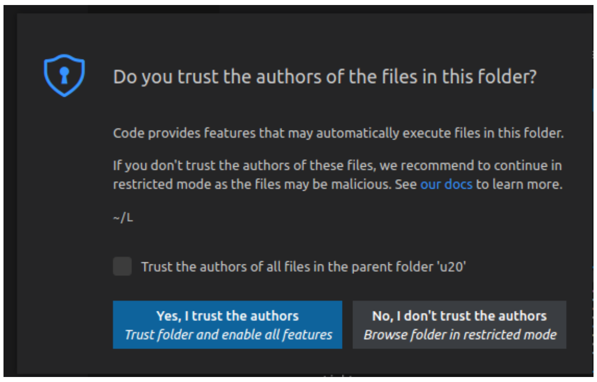


code

code
Code editing. Redefined.

Install

命令行：code .进入



Do you trust the authors of the files in this folder?

Code provides features that may automatically execute files in this folder.

If you don't trust the authors of these files, we recommend to continue in restricted mode as the files may be malicious. See our docs to learn more.

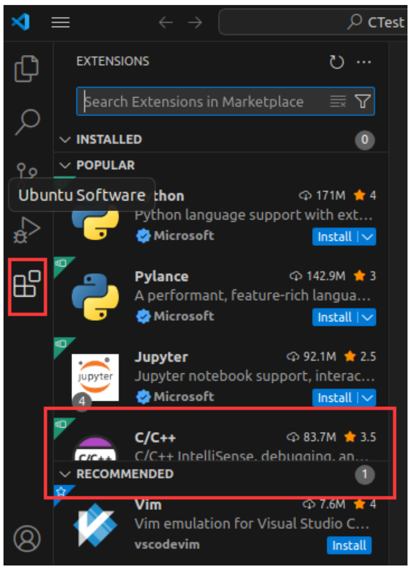
~/L

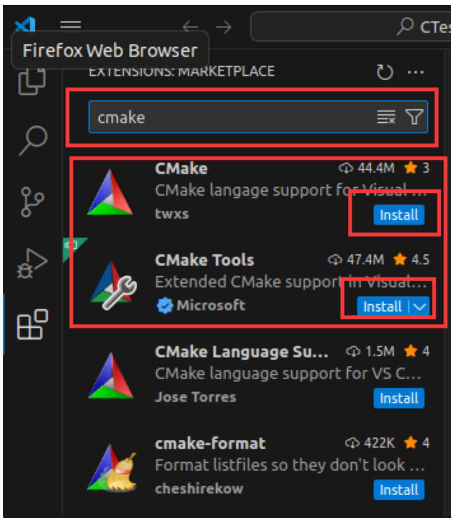
Trust the authors of all files in the parent folder 'u20'

Yes, I trust the authors
Trust folder and enable all features

No, I don't trust the authors
Browse folder in restricted mode

安装 C/C++，CMake，CMake Tools:


CTest
EXTENSIONS
Search Extensions in Marketplace
INSTALLED
POPULAR
Ubuntu Software
Python language support with ext...
Microsoft
Install
Pylance
A performant, feature-rich langua...
Microsoft
Install
Jupyter
Jupyter notebook support, interac...
Microsoft
Install
C/C++
C/C++ IntelliSense, debugging, an...
RECOMMENDED
Vim
Vim emulation for Visual Studio C...
vscodevim
Install



第三步：linux 下使用 VS Code 进行 C/C++开发

3.1 编译基础

在 Linux 下使用 Visual Studio Code (VSCode) 编译和调试 C++ 程序时，你可以选择使用 Makefile 或者 tasks.json 来管理编译过程，以及 launch.json 来配置调试会话。

- 对于简单的项目或快速原型开发，推荐使用 tasks.json 和 launch.json。
- 对于复杂的项目，推荐使用 Makefile 来管理构建过程，同时使用 launch.json 配置调试会话。
- 无论哪种方法，launch.json 都是配置调试会话的推荐方式。

1. CMakeLists.txt：这是一个标准的 CMake 配置文件，用于生成编译所需的 makefile 或其他构建系统文件。如果你的项目使用 CMake，这是必须的。CMake 是跨平台的编译工具，可以生成适用于多种编译器的构建系统。

CMakeLists.txt 示例

```
cmake_minimum_required(VERSION 3.10)
project(MyProject)

set(CMAKE_CXX_STANDARD 14)
set(CMAKE_CXX_STANDARD_REQUIRED True)

add_executable(MyExecutable main.cpp)
```

2. tasks.json：这个 JSON 文件定义了 VSCode 的任务（例如编译、构建项目等）。你可以通过 VSCode 的命令面板运行这些任务。这对于自动化编译过程非常有用。

tasks.json 示例

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "label": "build",
            "type": "shell",
            "command": "cmake --build build",
            "group": {
                "kind": "build",
                "isDefault": true
            }
        }
    ]
}
```

3. launch.json：这个 JSON 文件配置了调试器如何启动和附加到你的程序。它允许你设置断点、查看变量等，非常适合在开发过程中进行调试。

launch.json 示例

```
{
    "version": "0.2.0",
    "configurations": [
        {
```

```
                    "name": "C++ Launch",
                    "type": "cppdbg",
                    "request": "launch",
                    "program": "${workspaceFolder}/build/MyExecutable",
                    "args": [],
                    "stopAtEntry": false,
                    "cwd": "${workspaceFolder}",
                    "environment": [],
                    "externalConsole": false,
                    "MIMode": "gdb",
                    "setupCommands": [
                        {
                            "description": "Enable pretty-printing for gdb",
                            "text": "-enable-pretty-printing",
                            "ignoreFailures": true
                        }
                    ]
                }
            ]
        }
```

3.2 编译

1. 建立 hello world 工程

步骤 1：编辑源文件 hello.cpp

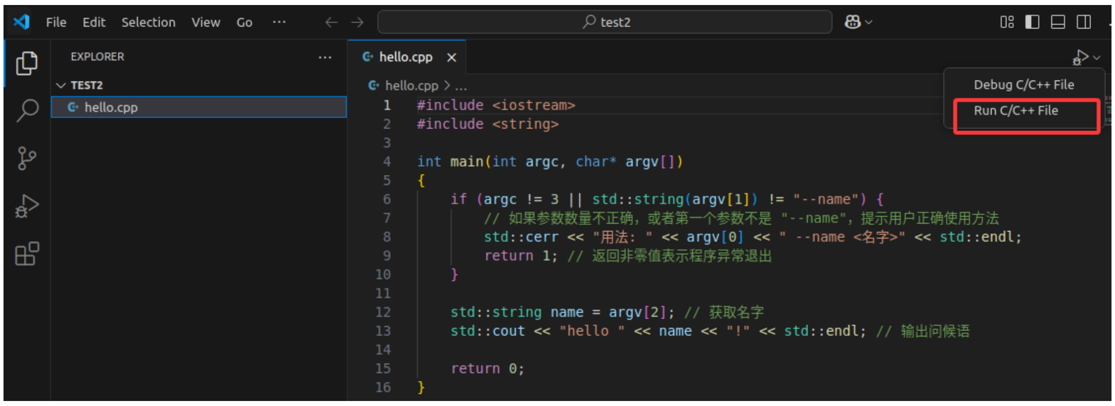
步骤 2：编译运行

● 方式一： 命令行编译运行：

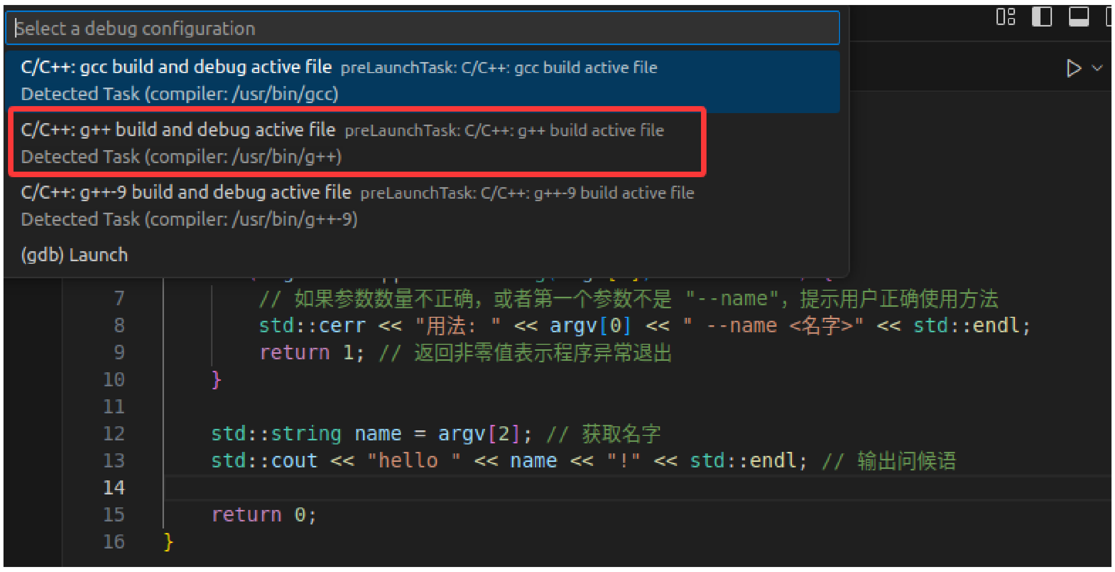
```
g++ hello.cpp -o 1
g++ hello.cpp -o hello
./hello --name LiMing
```

● 方式二： 单文件工程的编译运行调试：

1）编译运行



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

int main(int argc, char* argv[])
{
    if (argc != 3 || std::string(argv[1]) != "--name") {
        // 如果参数数量不正确，或者第一个参数不是 "--name"，提示用户正确使用方法
        std::cerr << "用法: " << argv[0] << " --name <名字>" << std::endl;
        return 1; // 返回非零值表示程序异常退出
    }

    std::string name = argv[2]; // 获取名字
    std::cout << "hello " << name << "!" << std::endl; // 输出问候语

    return 0;
}
```



Select a debug configuration

C/C++: gcc build and debug active file preLaunchTask: C/C++: gcc build active file
Detected Task (compiler: /usr/bin/gcc)

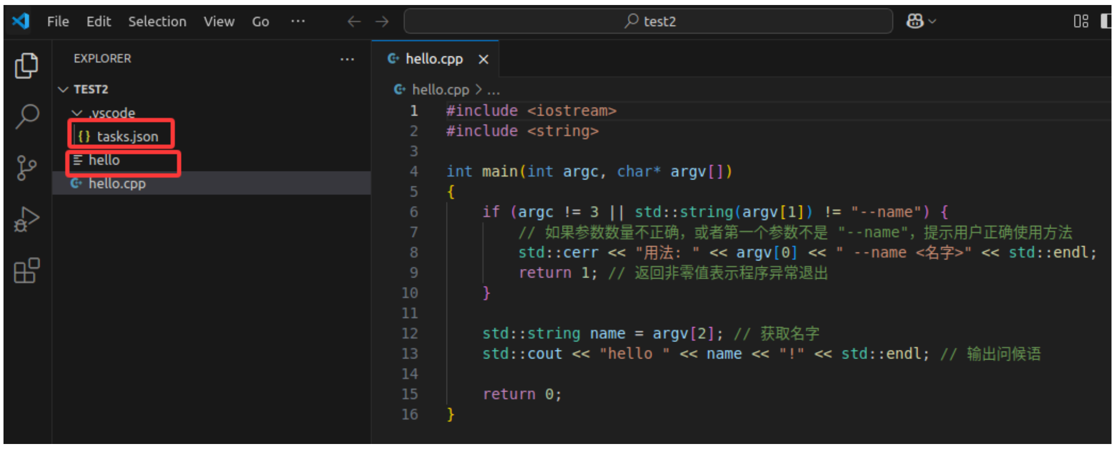
C/C++: g++ build and debug active file preLaunchTask: C/C++: g++ build active file
Detected Task (compiler: /usr/bin/g++)

C/C++: g++-9 build and debug active file preLaunchTask: C/C++: g++-9 build active file
Detected Task (compiler: /usr/bin/g++-9)

(gdb) Launch

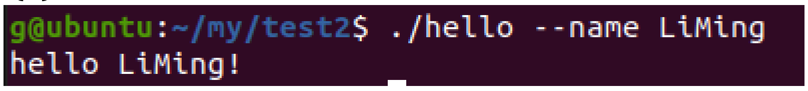
```
        // 如果参数数量不正确，或者第一个参数不是 "--name"，提示用户正确使用方法
        std::cerr << "用法: " << argv[0] << " --name <名字>" << std::endl;
        return 1; // 返回非零值表示程序异常退出
    }

    std::string name = argv[2]; // 获取名字
    std::cout << "hello " << name << "!" << std::endl; // 输出问候语

    return 0;
}
```

会在当前目录下生成可执行文件 hello，在.vscode 目录下生成 tasks.json



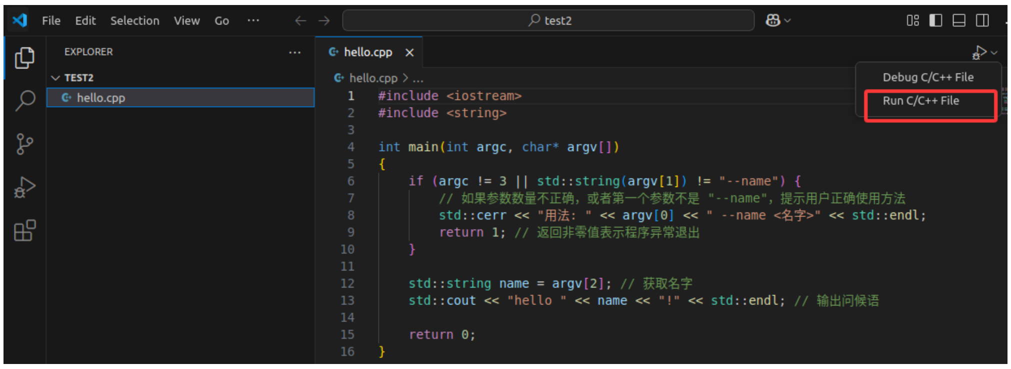
```
#include <iostream>
#include <string>

int main(int argc, char* argv[])
{
    if (argc != 3 || std::string(argv[1]) != "--name") {
        // 如果参数数量不正确，或者第一个参数不是 "--name"，提示用户正确使用方法
        std::cerr << "用法: " << argv[0] << " --name <名字>" << std::endl;
        return 1; // 返回非零值表示程序异常退出
    }

    std::string name = argv[2]; // 获取名字
    std::cout << "hello " << name << "!" << std::endl; // 输出问候语

    return 0;
}
```

(1)可在命令行运行 hello



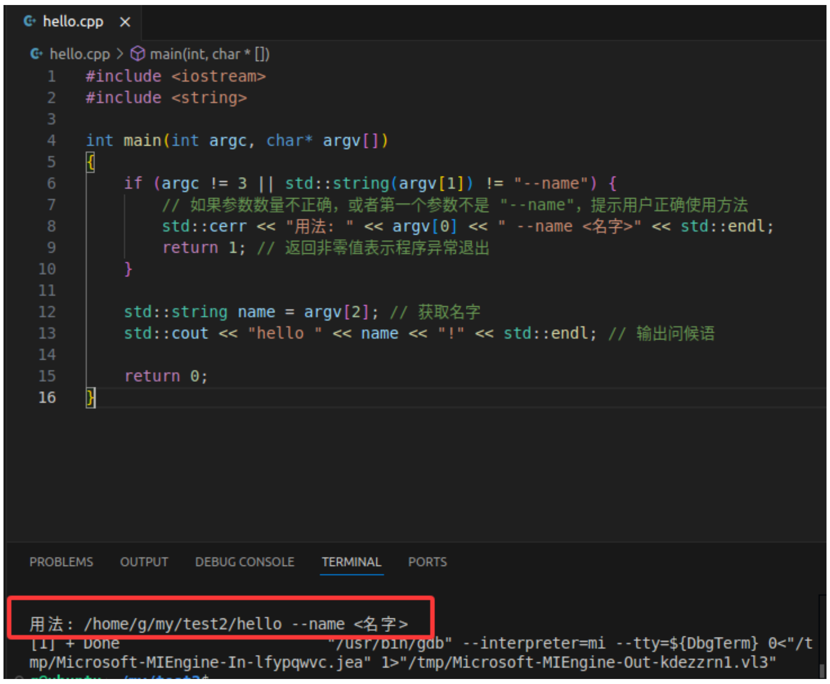
```
g@ubuntu:~/my/test2$ ./hello --name LiMing
hello LiMing!
```

(2) 可在 VS 中运行



```
 1  #include <iostream>
 2  #include <string>
 3
 4  int main(int argc, char* argv[])
 5  {
 6      if (argc != 3 || std::string(argv[1]) != "--name") {
 7          // 如果参数数量不正确，或者第一个参数不是 "--name"，提示用户正确使用方法
 8          std::cerr << "用法: " << argv[0] << " --name <名字>" << std::endl;
 9          return 1; // 返回非零值表示程序异常退出
10      }
11
12      std::string name = argv[2]; // 获取名字
13      std::cout << "hello " << name << "!" << std::endl; // 输出问候语
14
15      return 0;
16  }
```

结果为



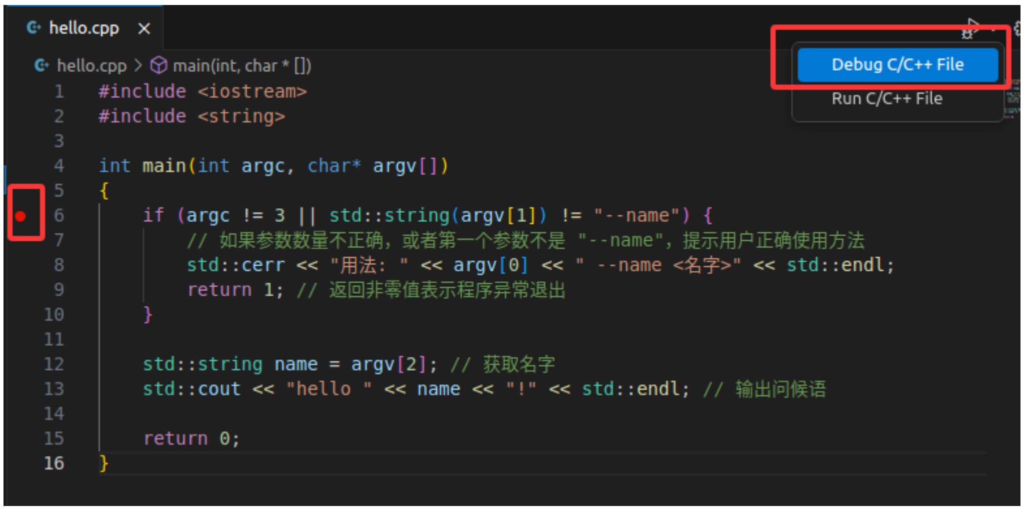
```
 1  #include <iostream>
 2  #include <string>
 3
 4  int main(int argc, char* argv[])
 5  {
 6      if (argc != 3 || std::string(argv[1]) != "--name") {
 7          // 如果参数数量不正确，或者第一个参数不是 "--name"，提示用户正确使用方法
 8          std::cerr << "用法: " << argv[0] << " --name <名字>" << std::endl;
 9          return 1; // 返回非零值表示程序异常退出
10      }
11
12      std::string name = argv[2]; // 获取名字
13      std::cout << "hello " << name << "!" << std::endl; // 输出问候语
14
15      return 0;
16  }
```

```
用法: /home/g/my/test2/hello --name <名字>
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/t
mp/Microsoft-MIEngine-In-lfypqwvc.jea" 1>"/tmp/Microsoft-MIEngine-Out-kdezzrn1.vl3"
```

若想在 VScode 中输入参数，参加下文 3）设置输入参数，进行运行或调试。

2）调试

在第六行设置断点，并 debug

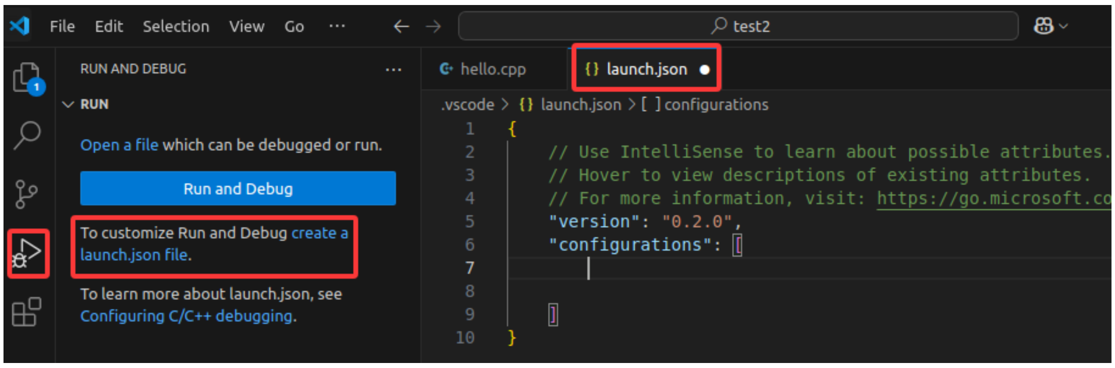


```
 1  #include <iostream>
 2  #include <string>
 3
 4  int main(int argc, char* argv[])
 5  {
 6      if (argc != 3 || std::string(argv[1]) != "--name") {
 7          // 如果参数数量不正确，或者第一个参数不是 "--name"，提示用户正确使用方法
 8          std::cerr << "用法: " << argv[0] << " --name <名字>" << std::endl;
 9          return 1; // 返回非零值表示程序异常退出
10      }
11
12      std::string name = argv[2]; // 获取名字
13      std::cout << "hello " << name << "!" << std::endl; // 输出问候语
14
15      return 0;
16  }
```

F10 或 F11 可单步执行。

3）设置输入参数，进行运行或调试

Step1：创建默认 launch.json



Open a file which can be debugged or run.

To customize Run and Debug create a launch.json file.

To learn more about launch.json, see Configuring C/C++ debugging.

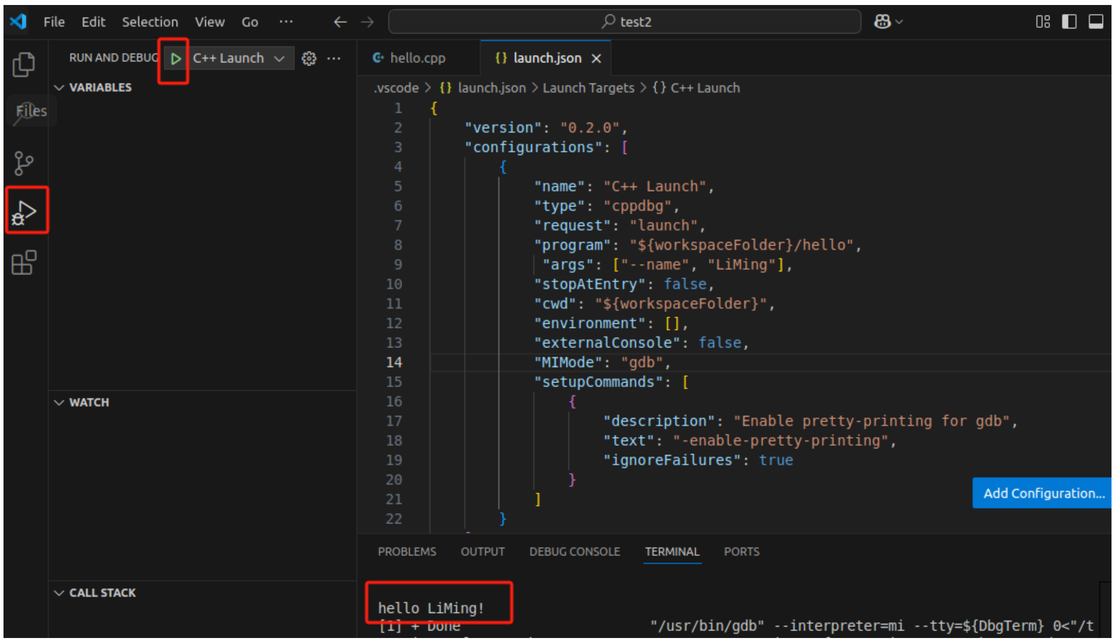
```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.co
    "version": "0.2.0",
    "configurations": [
        |

    ]
}
```

Step2：修改 launch.json 为

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "C++ Launch",
            "type": "cppdbg",
            "request": "launch",
            "program": "${workspaceFolder}/ hello",
             "args": ["--name", "LiMing"],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                }
            ]
        }
    ]
}
```

Step3：ctrl+S 保存文件

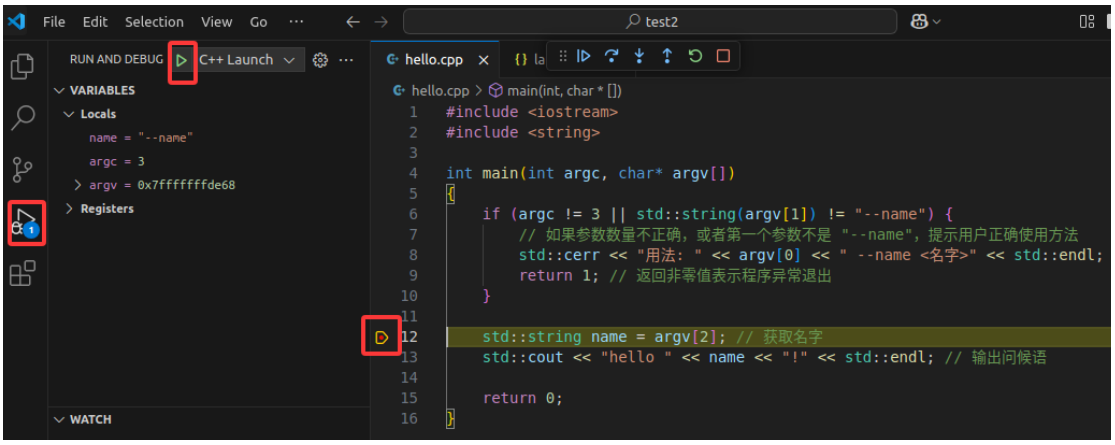
Step4：运行


File Edit Selection View Go ···
test2
RUN AND DEBUG
C++ Launch
VARIABLES
WATCH
CALL STACK
hello.cpp
launch.json
.vscode > {} launch.json > Launch Targets > {} C++ Launch
Add Configuration...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```json
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "C++ Launch",
            "type": "cppdbg",
            "request": "launch",
            "program": "${workspaceFolder}/hello",
             "args": ["--name", "LiMing"],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                }
            ]
        }
```

```
hello LiMing!
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/t
```

或调试



```
name = "--name"
argc = 3
argv = 0x7fffffffde68
```

```cpp
1   #include <iostream>
2   #include <string>
3
4   int main(int argc, char* argv[])
5   {
6       if (argc != 3 || std::string(argv[1]) != "--name") {
7           // 如果参数数量不正确，或者第一个参数不是 "--name"，提示用户正确使用方法
8           std::cerr << "用法: " << argv[0] << " --name <名字>" << std::endl;
9           return 1; // 返回非零值表示程序异常退出
10      }
11
12      std::string name = argv[2]; // 获取名字
13      std::cout << "hello " << name << "!" << std::endl; // 输出问候语
14
15      return 0;
16  }
```

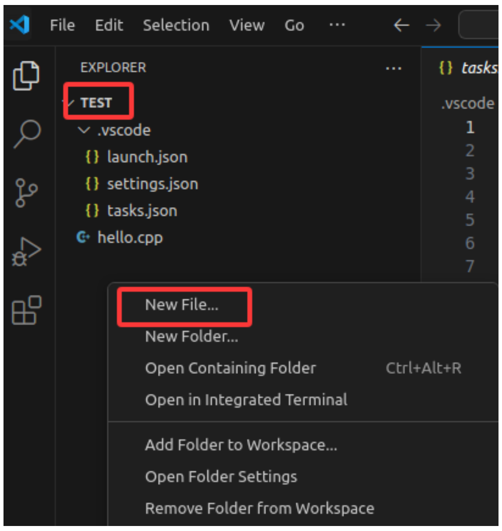
上图为预先在 12 行设置了断点，

运行或调试的快捷键为 F5

- 方式三：复杂工程 Makefile 编译运行调试：

1）code .

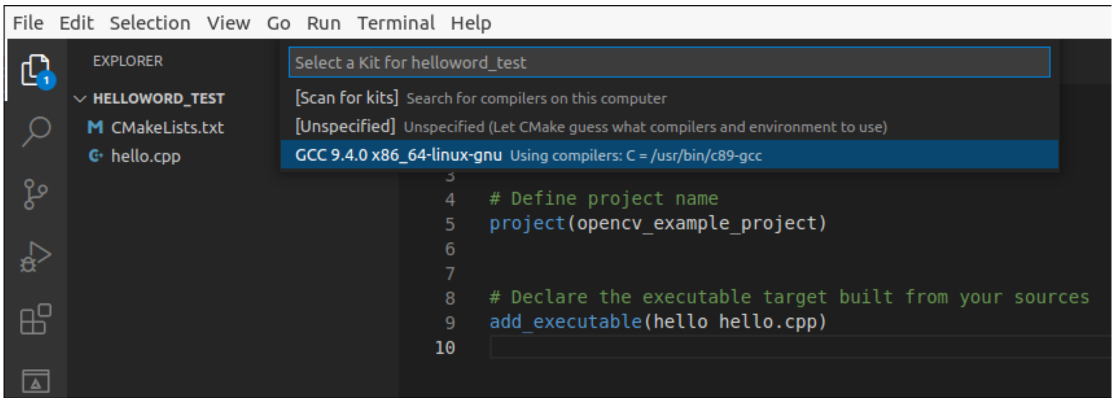
2）在当前工程根目录下新建 CMakeLists.txt，



输入 CMakeLists.txt，并将其内容修改为

```
# cmake needs this line
cmake_minimum_required(VERSION 3.5)

# Define project name
project(helloworld)

# Declare the executable target built from your sources
add_executable(hello hello.cpp)
```

3）快捷键 Ctrl+Shift+P，打开命令面板，搜索 CMake，选择第一个 CMake：Configure，之后选择 GCC9.4.0 x86——64-linux-gnu

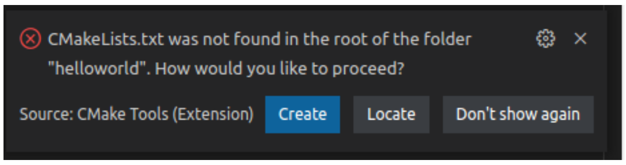


Select a Kit for helloword_test
[Scan for kits] Search for compilers on this computer
[Unspecified] Unspecified (Let CMake guess what compilers and environment to use)
GCC 9.4.0 x86_64-linux-gnu Using compilers: C = /usr/bin/c89-gcc

```
# Define project name
project(opencv_example_project)


# Declare the executable target built from your sources
add_executable(hello hello.cpp)
```

****

若之前跳过步骤 2，没有 CMakelist.txt，右下角，可让 VS code 自动建立 CMakelist.txt，

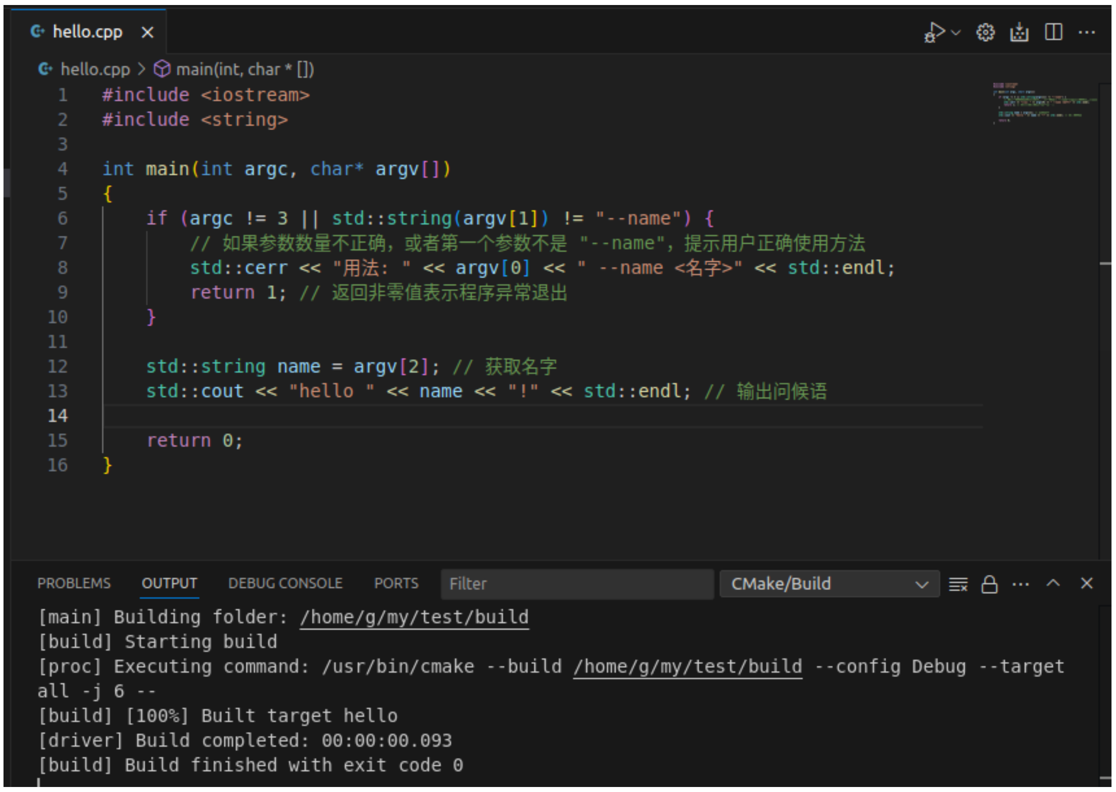


CMakeLists.txt was not found in the root of the folder "helloworld". How would you like to proceed?

Source: CMake Tools (Extension)

****

4）快捷键 Ctrl+Shift+P，打开命令面板，输入 CMake: Build 之后进行编译



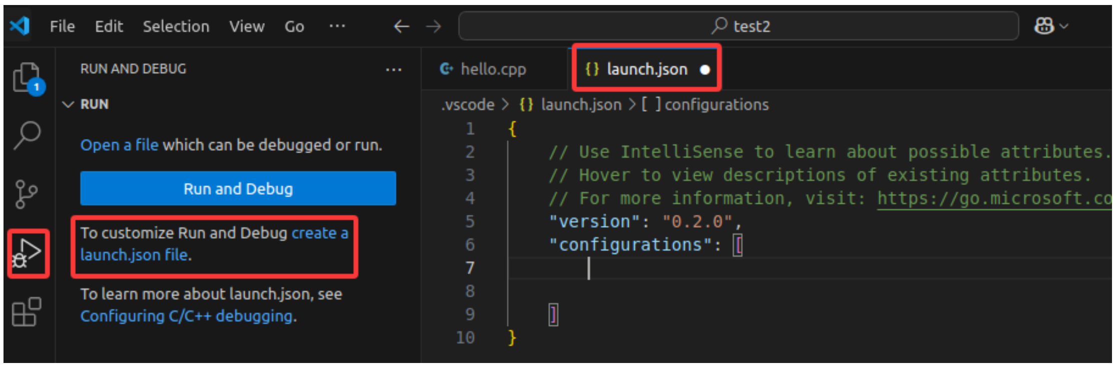
```cpp
#include <iostream>
#include <string>

int main(int argc, char* argv[])
{
    if (argc != 3 || std::string(argv[1]) != "--name") {
        // 如果参数数量不正确，或者第一个参数不是 "--name"，提示用户正确使用方法
        std::cerr << "用法: " << argv[0] << " --name <名字>" << std::endl;
        return 1; // 返回非零值表示程序异常退出
    }

    std::string name = argv[2]; // 获取名字
    std::cout << "hello " << name << "!" << std::endl; // 输出问候语

    return 0;
}
```

```
[main] Building folder: /home/g/my/test/build
[build] Starting build
[proc] Executing command: /usr/bin/cmake --build /home/g/my/test/build --config Debug --target
all -j 6 --
[build] [100%] Built target hello
[driver] Build completed: 00:00:00.093
[build] Build finished with exit code 0
```

显示成功，并在 build 目录下生成可执行文件 hello，这是由 CMakeLists.txt 里的 add_executable(hello hello.cpp)设定的。

5）运行与调试

Step1：创建默认 launch.json



Open a file which can be debugged or run.

To customize Run and Debug create a launch.json file.

To learn more about launch.json, see Configuring C/C++ debugging.

```json
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.co
    "version": "0.2.0",
    "configurations": [


    ]
}
```

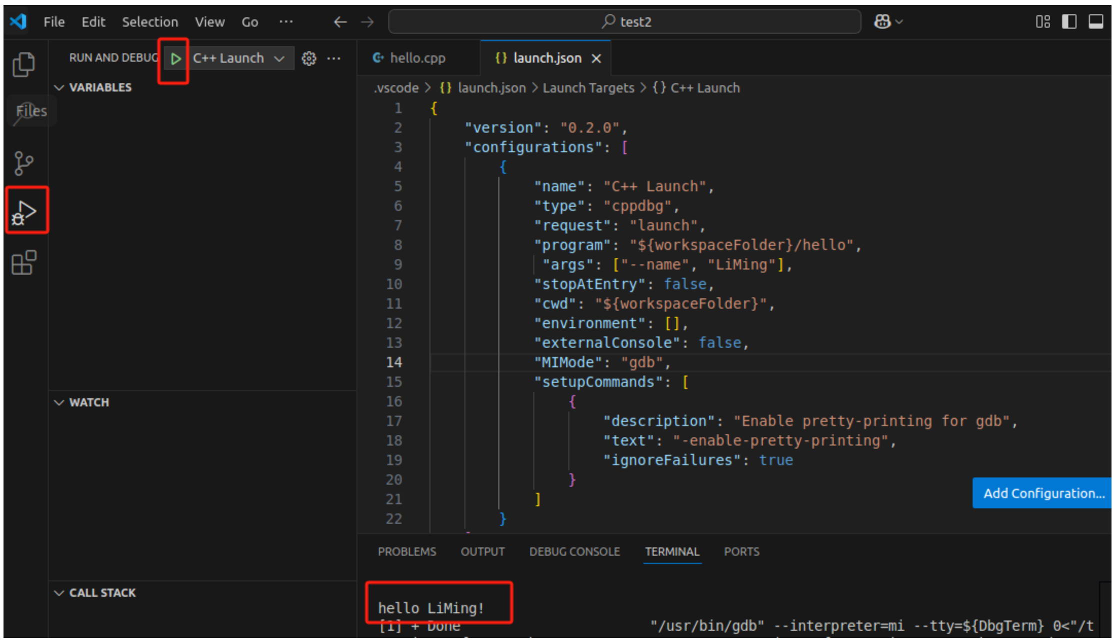
Step2：修改 launch.json 为

{

```
    "version": "0.2.0",
    "configurations": [
        {
            "name": "C++ Launch",
            "type": "cppdbg",
            "request": "launch",
            "program": "${workspaceFolder}/ replace",
             "args": ["--name", "LiMing"],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                }
            ]
        }
    ]
}
```

Step3：ctrl+S 保存文件

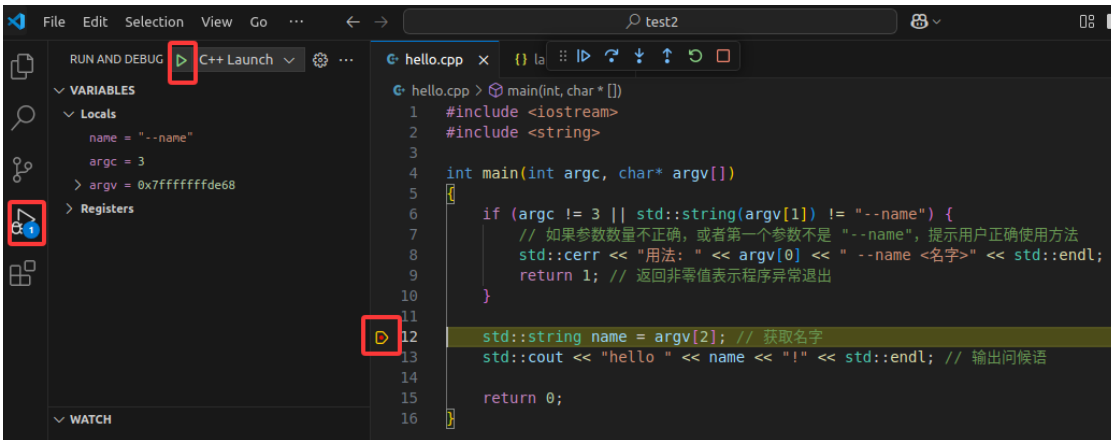
Step4：运行



```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "C++ Launch",
            "type": "cppdbg",
            "request": "launch",
            "program": "${workspaceFolder}/hello",
             "args": ["--name", "LiMing"],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                }
            ]
        }
```

```
hello LiMing!
[1] + Done                    "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/t
```

或调试



```
name = "--name"
argc = 3
argv = 0x7fffffffde68
```

```
1   #include <iostream>
2   #include <string>
3
4   int main(int argc, char* argv[])
5   {
6       if (argc != 3 || std::string(argv[1]) != "--name") {
7           // 如果参数数量不正确，或者第一个参数不是 "--name"，提示用户正确使用方法
8           std::cerr << "用法: " << argv[0] << " --name <名字>" << std::endl;
9           return 1; // 返回非零值表示程序异常退出
10      }
11
12      std::string name = argv[2]; // 获取名字
13      std::cout << "hello " << name << "!" << std::endl; // 输出问候语
14
15      return 0;
16  }
```

上图为预先在 12 行设置了断点，

运行或调试的快捷键为 F5

注意：launch.json 中的 program 应设置为 build 目录下可执行文件。

即　"program": "${workspaceFolder}/ build/hello",