实验环境配置

实验环境介绍

实验工具

实验至少需要用到以下工具，请同学们自行安装并学习，在下面也有简单的使用说明，这个程度的说明只能保证基础的使用，如果出现不符合预期的情况，请阅读文档

1. Git
2. CMake
3. GNU Makefiles / GCC / GDB

实验指导书中给出了最少的要安装的工具包，但是这些对于一个舒服的开发环境来说，还远远不够！

下面给出我在完成实验时使用到的一些工具

1.WSL2

提供Windows操作系统下的Linux环境

文档：WSL2安装

2.GCC

编译器

文档：GCC安装

3.GDB

调试器

文档：GDB安装

4.CMake

项目管理工具

文档：CMake安装 cmake与make

5.VS Code

编辑器

文档：VSCode与WSL2

6.VSCode CMake Tools Extension

图形化界面

文档：[VSCode CMake Tools Extension]

具体步骤

1.安装wsl

以管理员身份打开powershell，执行以下命令：

更新wsl

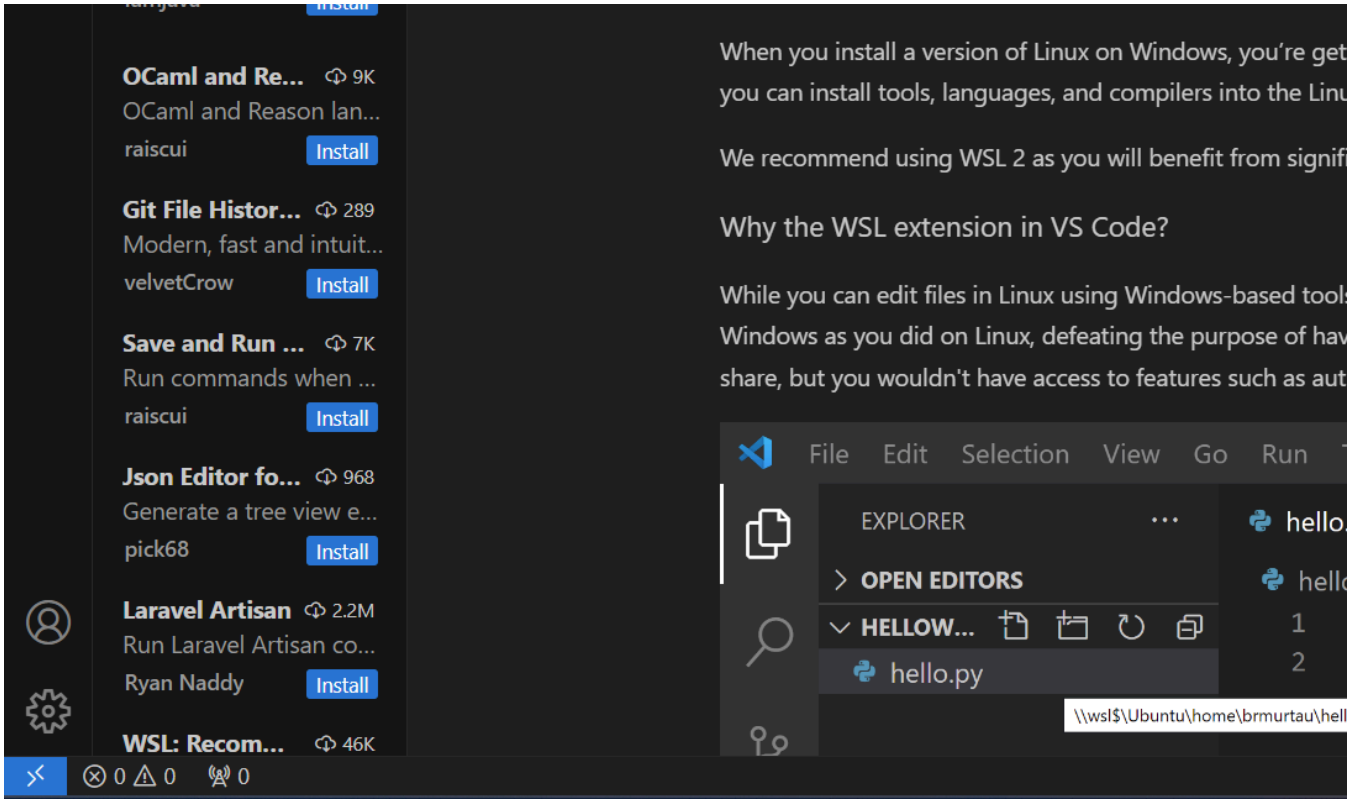
```
wsl --update --web-download
```

安装Ubuntu发行版

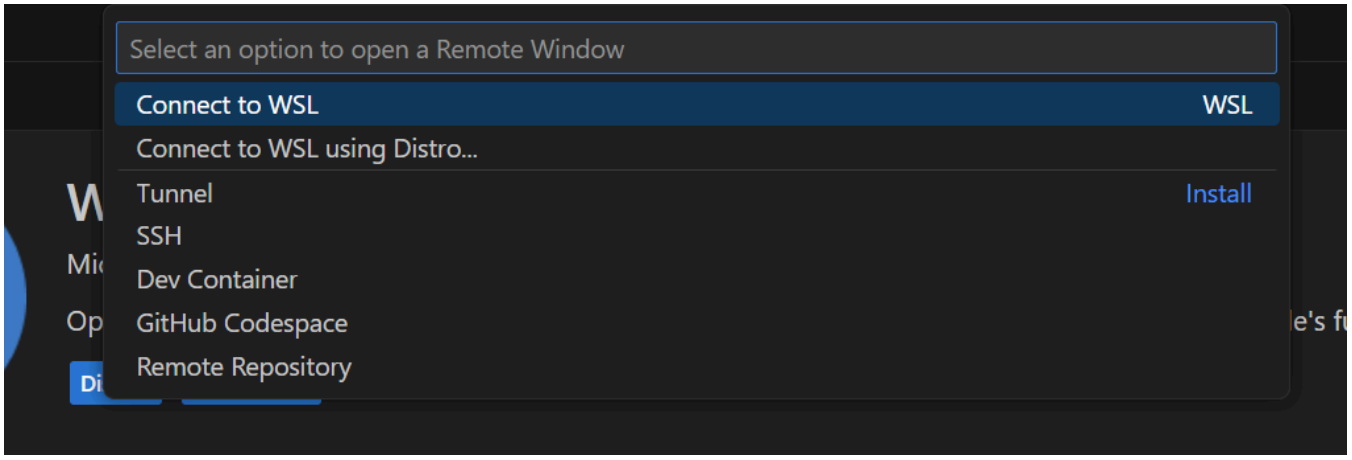
```
wsl --install -d Ubuntu --web-download
```

2.打开vscode搜索安装wsl插件

安装WSL插件



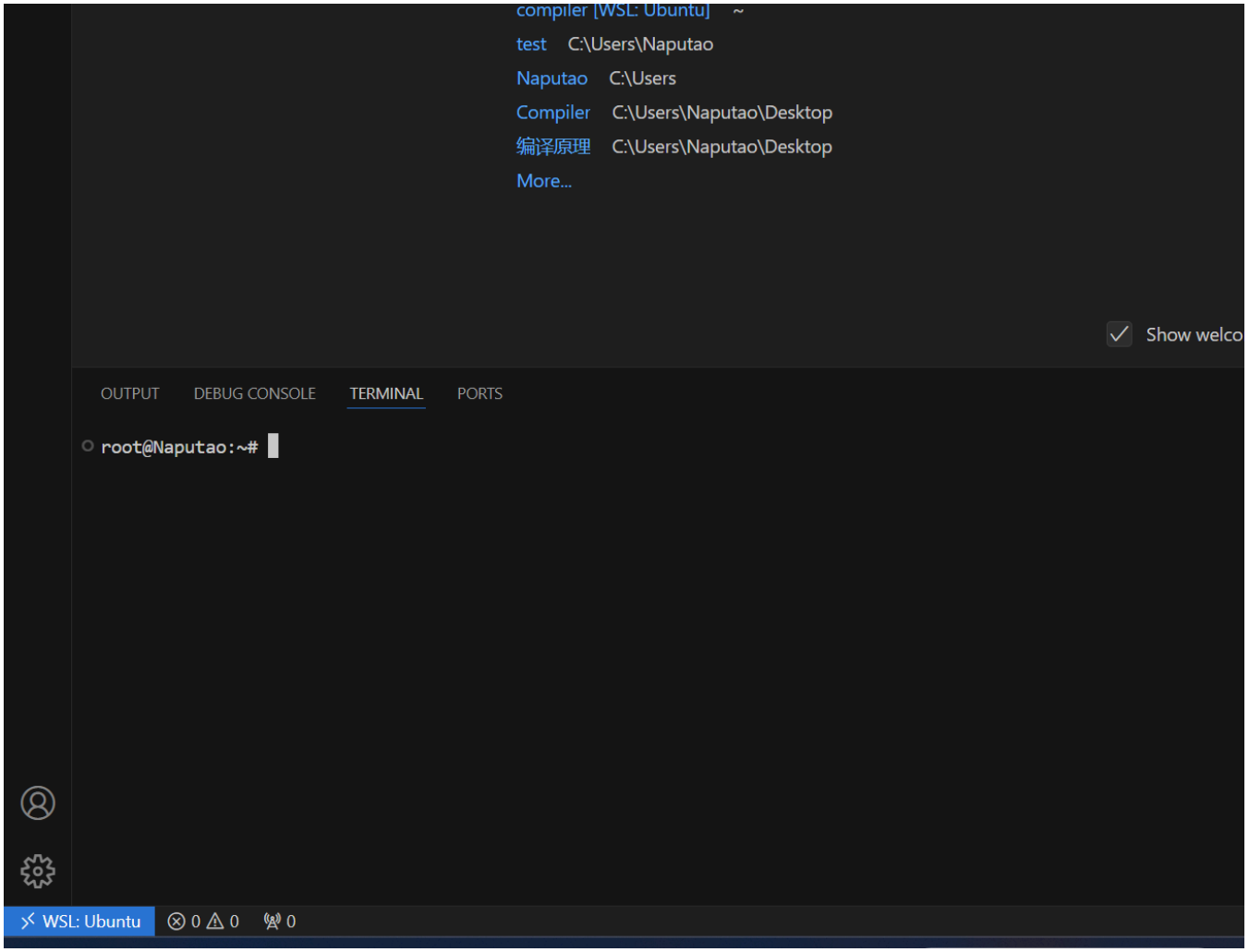
点击最左下角蓝色"><"按钮



点击“连接到WSL”

3.安装gcc g++ gdb make cmake

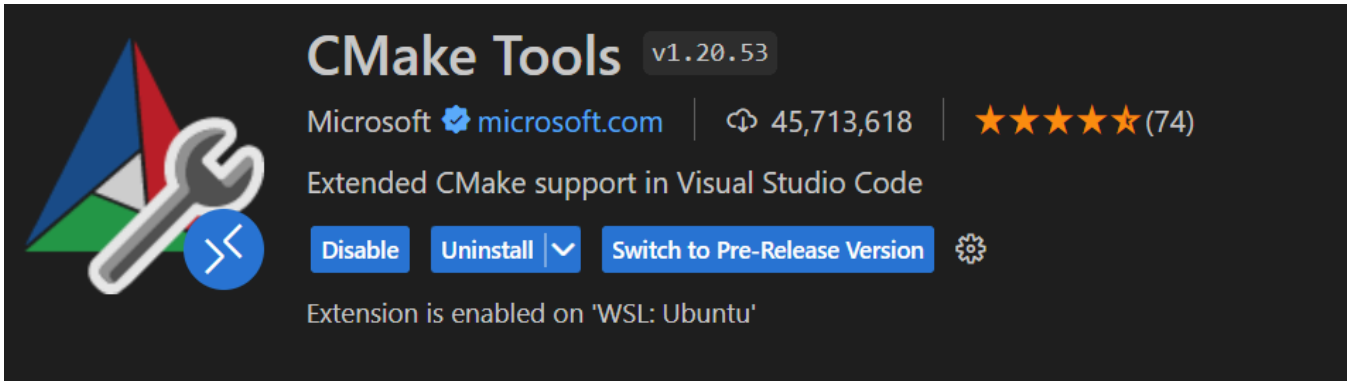
ctrl + '~'波浪键打开WSL下Ubuntu发行版的命令窗口



安装gcc g++ gdb make cmake

```
apt-get update
apt install gcc
apt install g++
apt install gdb
apt install make
apt install cmake
```

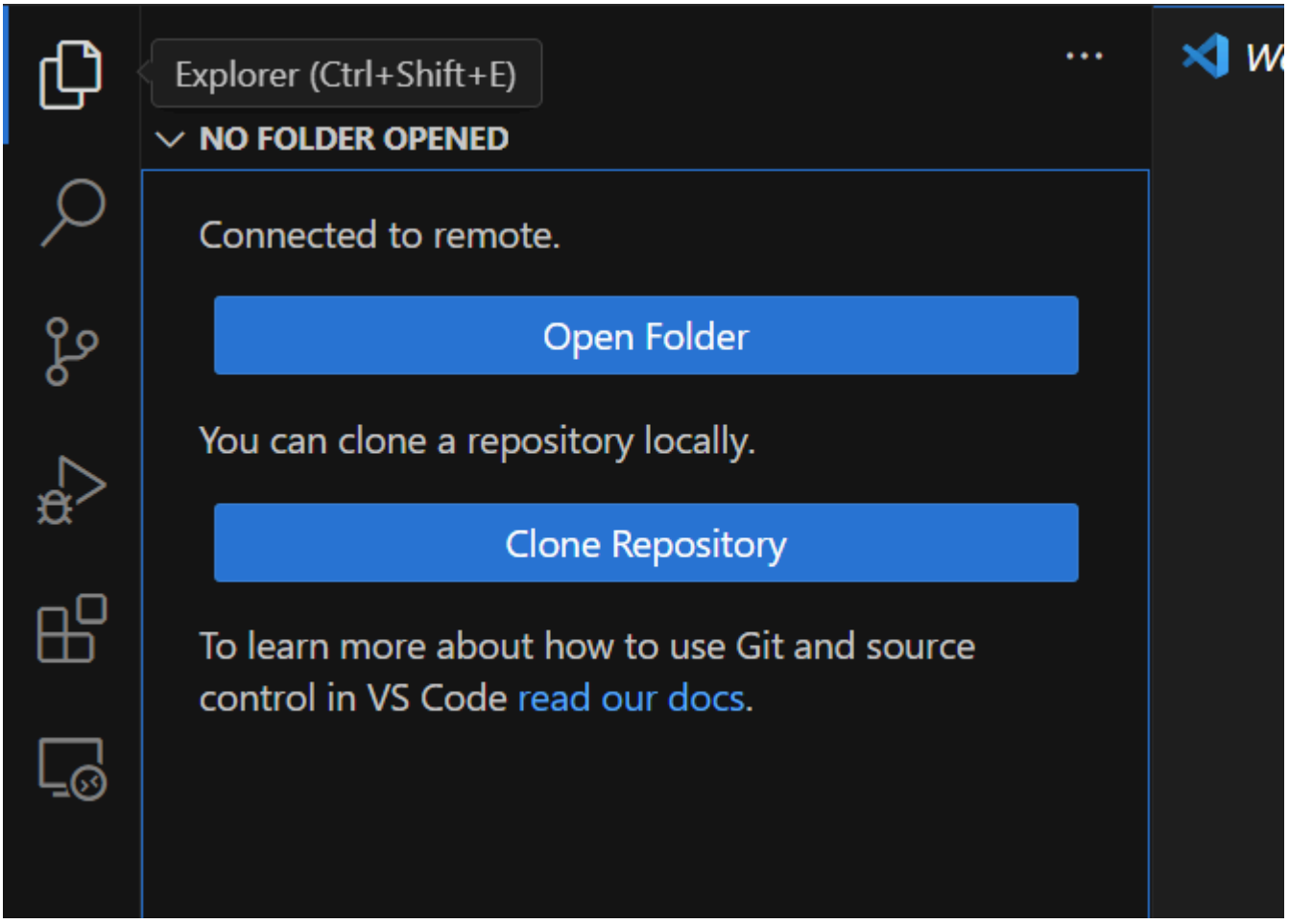
4.安装cmake tools插件



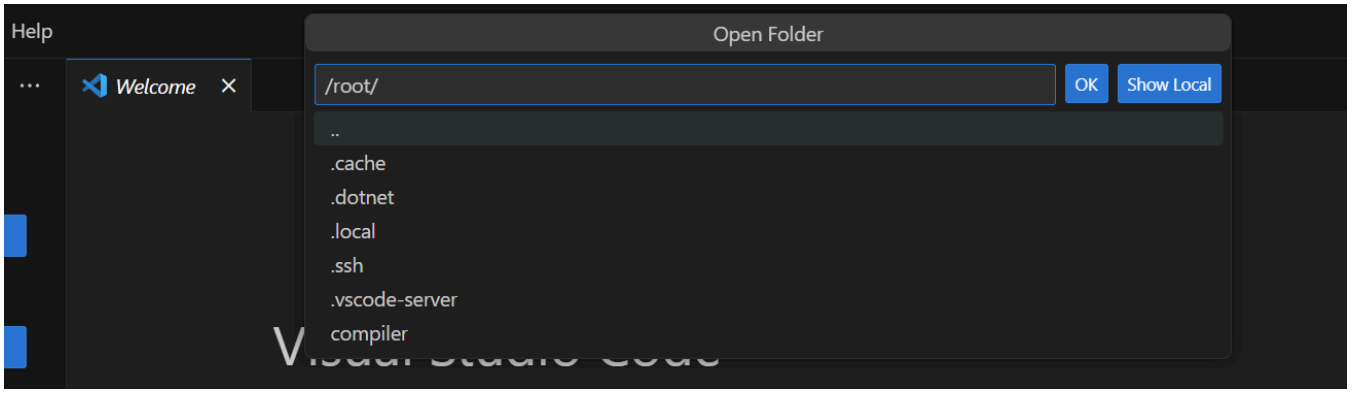
5.编写cmakelists.txt文件

首先创建文件夹

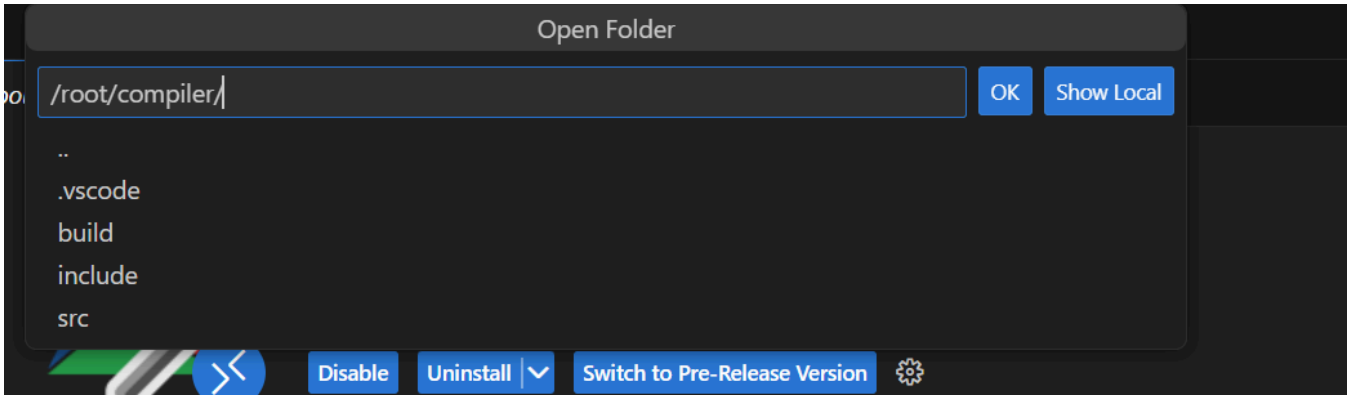
```
mkdir compiler
```



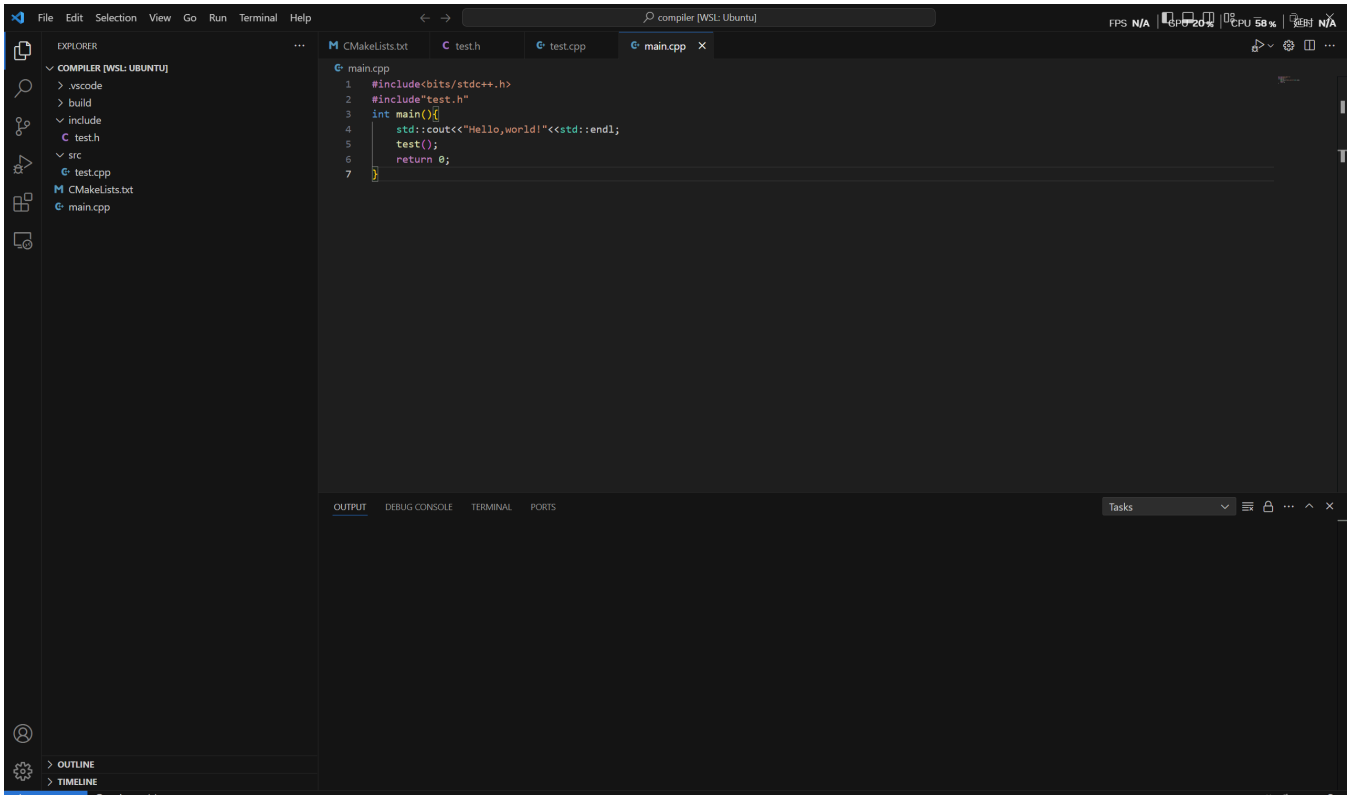
通过vscode打开文件夹，点击"打开文件夹"



选择compiler目录



点击OK



依次创建`src/test.cpp` `include/test.h` `CMakeLists.txt` `main.cpp`四个文件

```h
//test.h
#ifndef TEST_H
#define TEST_H
#include<unordered_map>
#include<string>
void test();
void f();
#endif
```

```cpp
//test.cpp
#include"test.h"
#include<iostream>
#include<vector>
void test(){
    std::cout<<"include ok"<<std::endl;
}
```

```cpp
void f(){
    std::vector<int> vec = {1,2,3,4,5,6,7,8};
    for(auto&i:vec){
        std::cout<<i<<std::endl;
    }
    return;
}
```

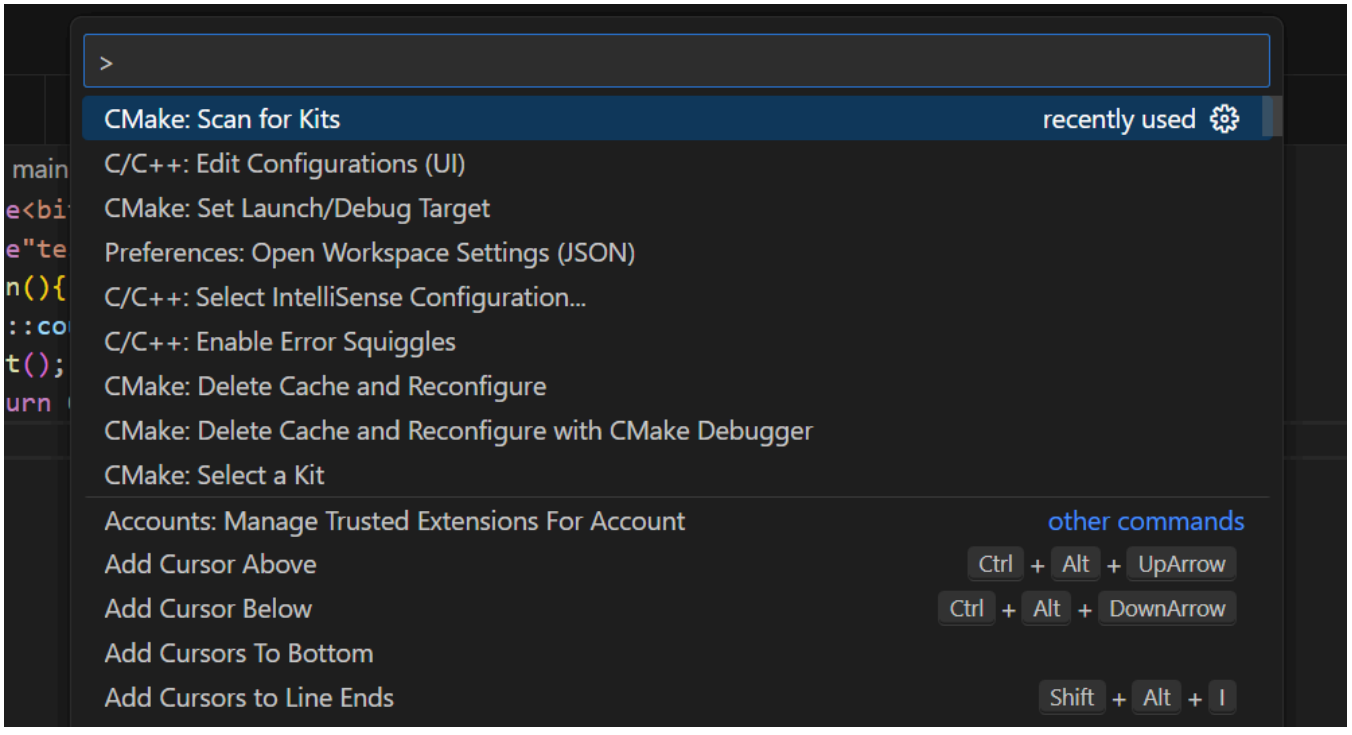
```cpp
//main.cpp
#include<bits/stdc++.h>
#include"test.h"
int main(){
    std::cout<<"Hello,world!"<<std::endl;
    test();
    f();
    return 0;
}
```

```cmake
# CMakeLists.txt
cmake_minimum_required(VERSION 3.11)
project(compiler)
set(CMAKE_CXX_STANDARD 23)
include_directories(${PROJECT_SOURCE_DIR}/include)
add_executable(main main.cpp src/test.cpp)
set(CMAKE_CXX_FLAGS_DEBUG "-g")
```

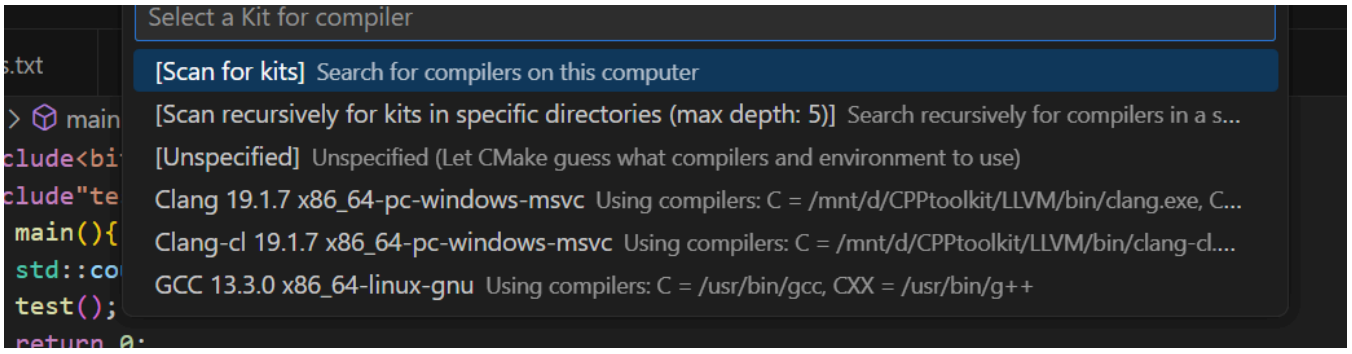
依次创建`src/test.cpp` `include/test.h` `CMakeLists.txt` `main.cpp`四个文件

6.启动项目

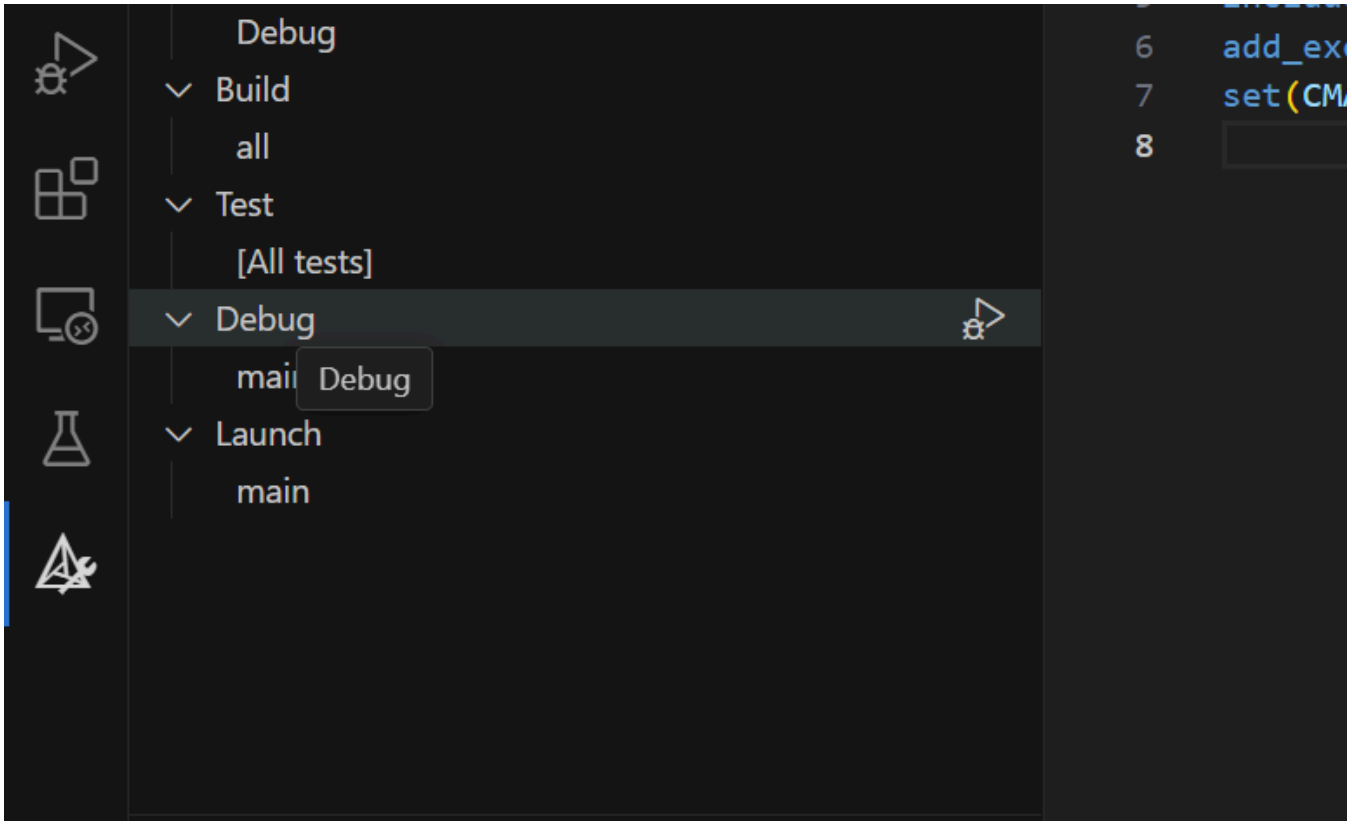
Ctrl + Shift + p 输入查找CMake Select for Kits



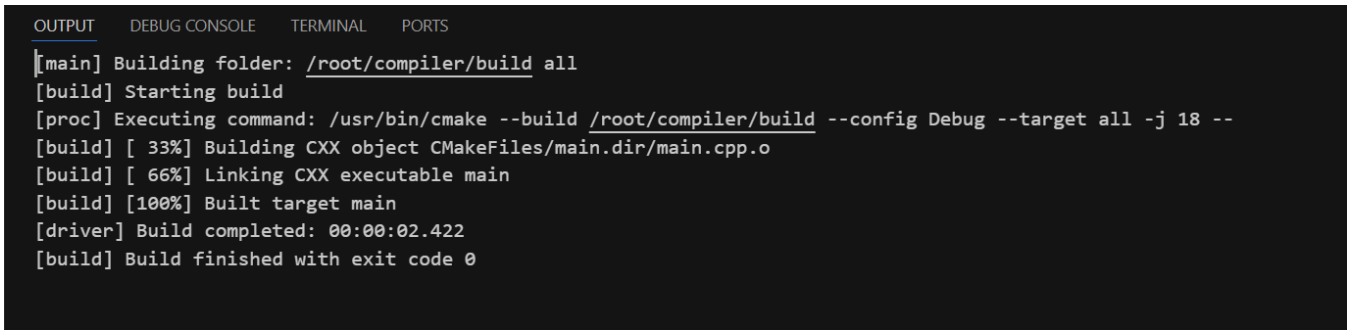
选择GCC



点击左侧小三角



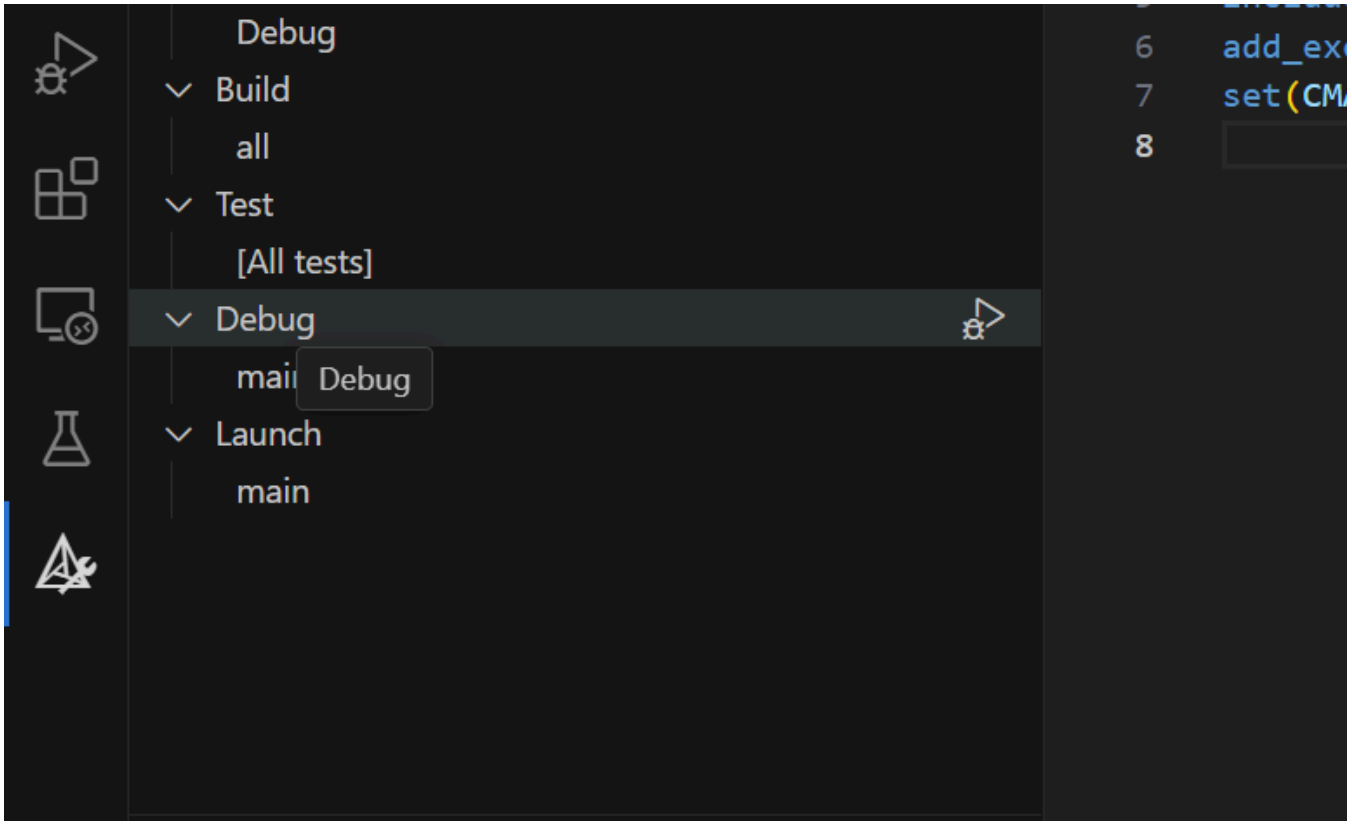
点击launch，即可构建运行main.cpp

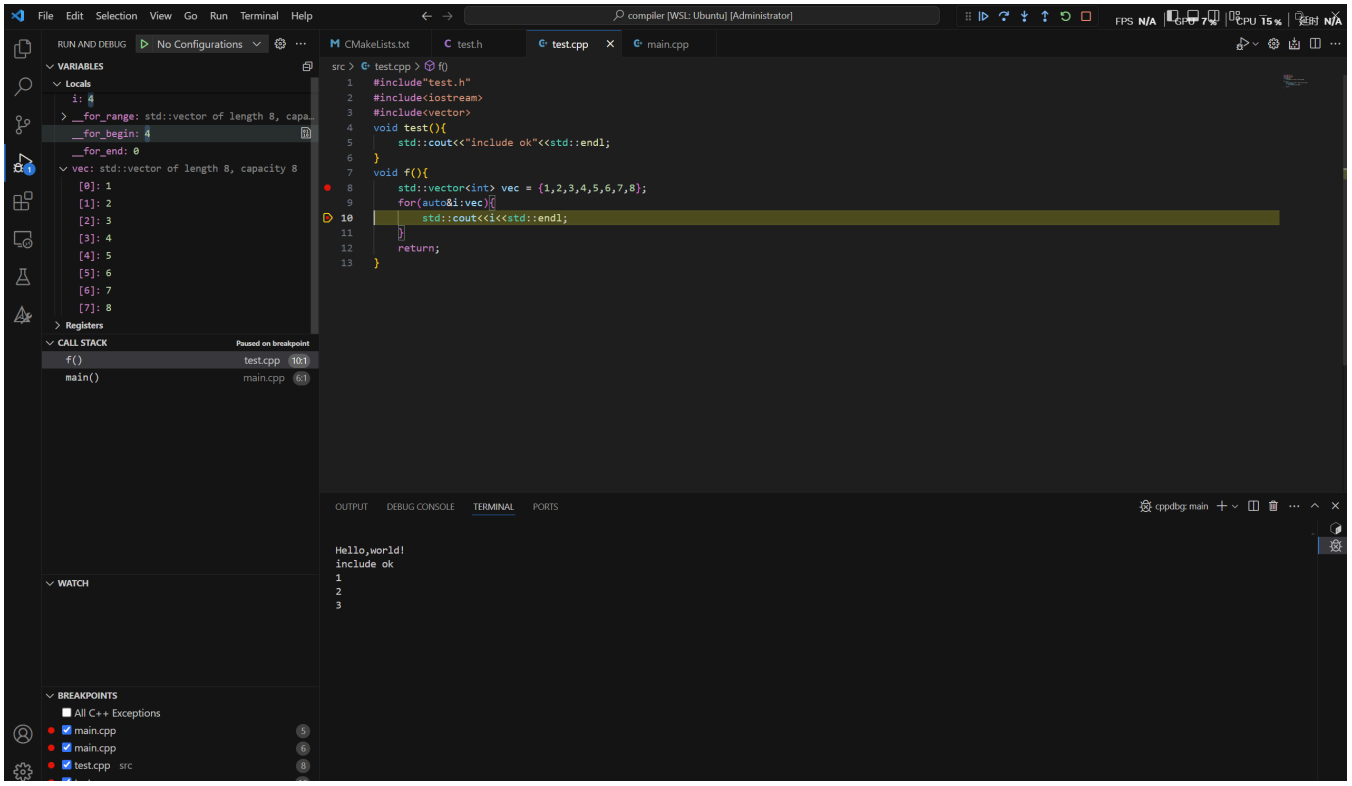


```
Hello,world!
include ok
root@Naputao:~/compiler/build# /root/compiler/build/main
Hello,world!
include ok
root@Naputao:~/compiler/build# /root/compiler/build/main
Hello,world!
include ok
root@Naputao:~/compiler/build#
```

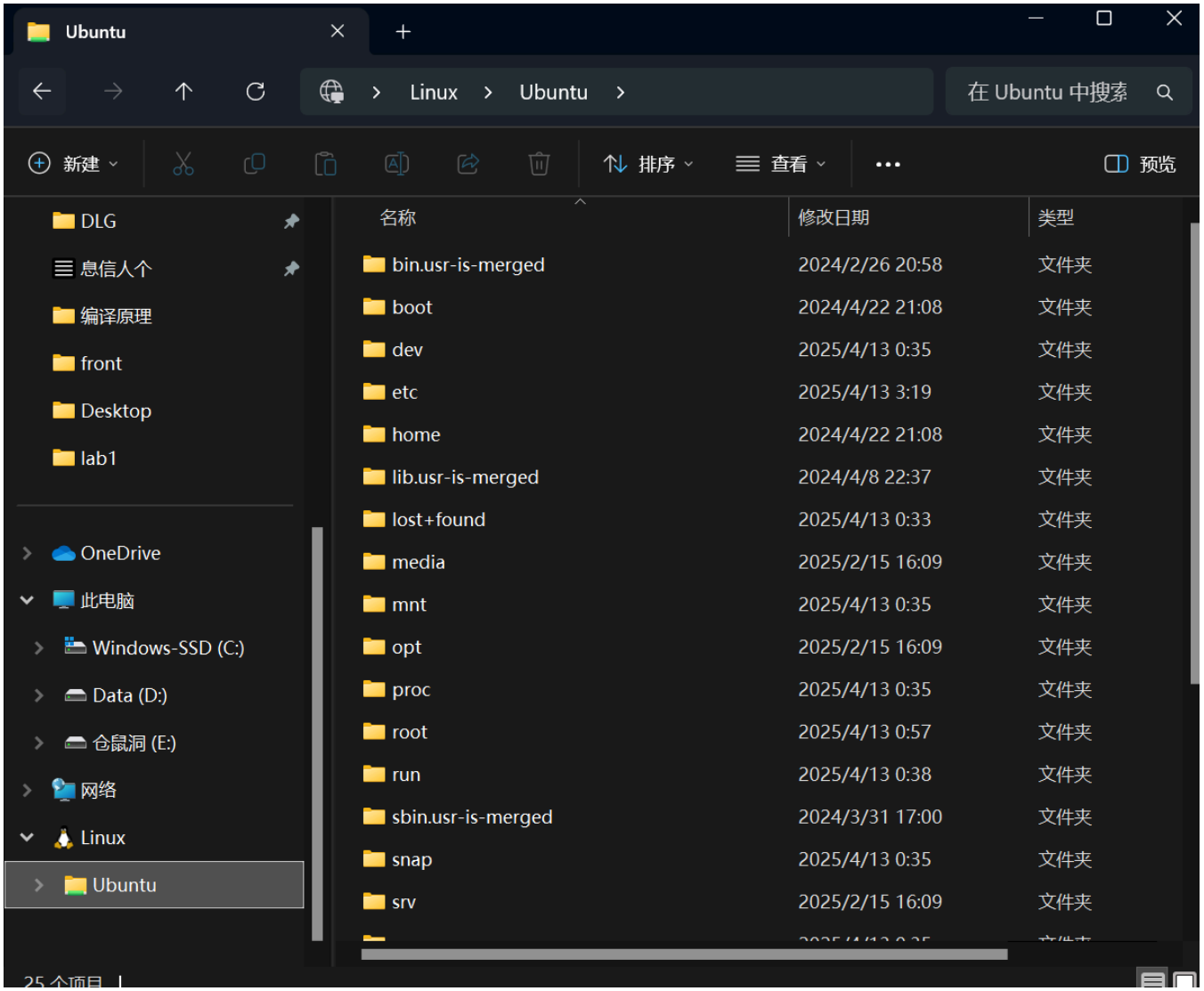
点击文本左侧添加断点

点击debug即可调试



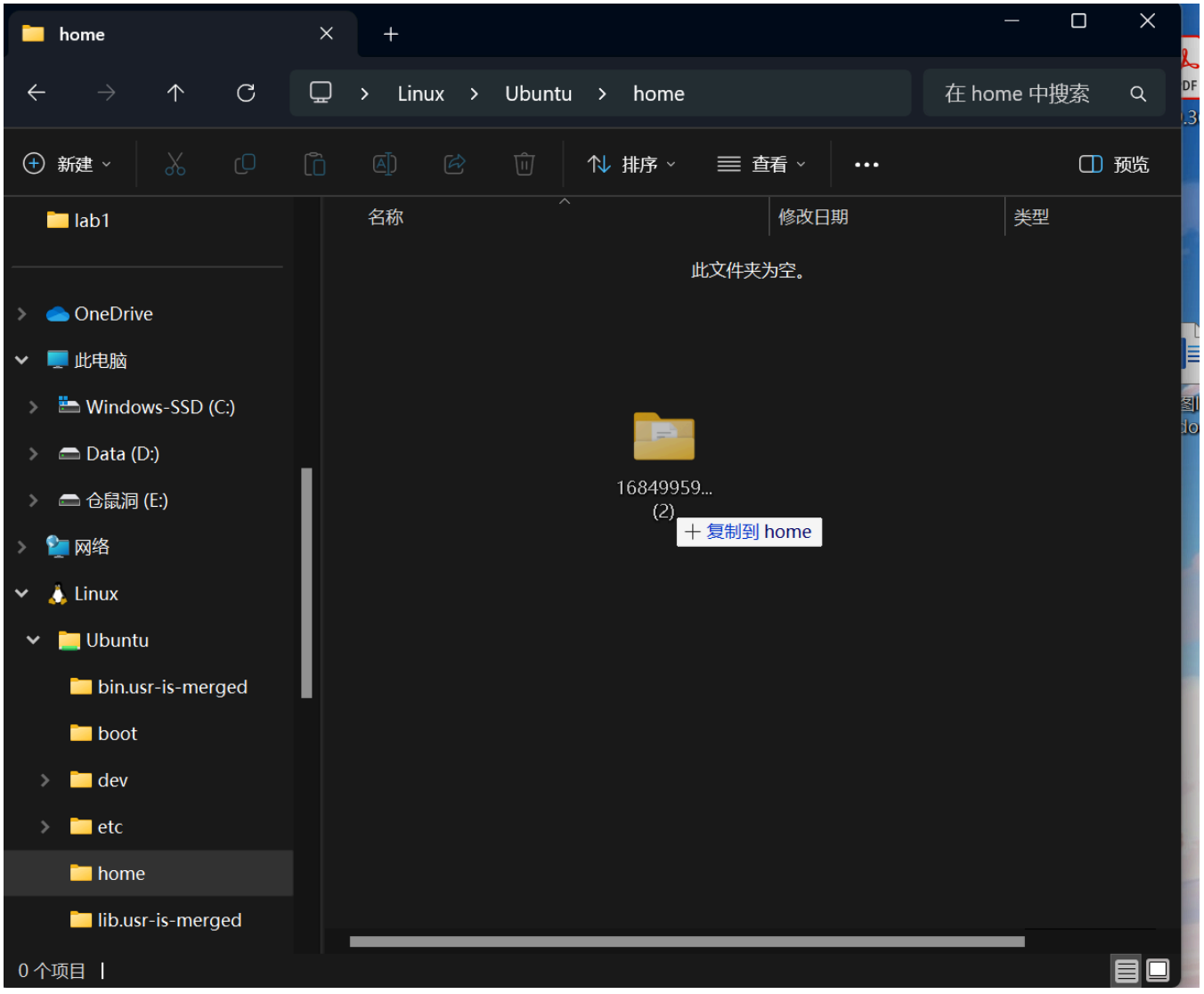


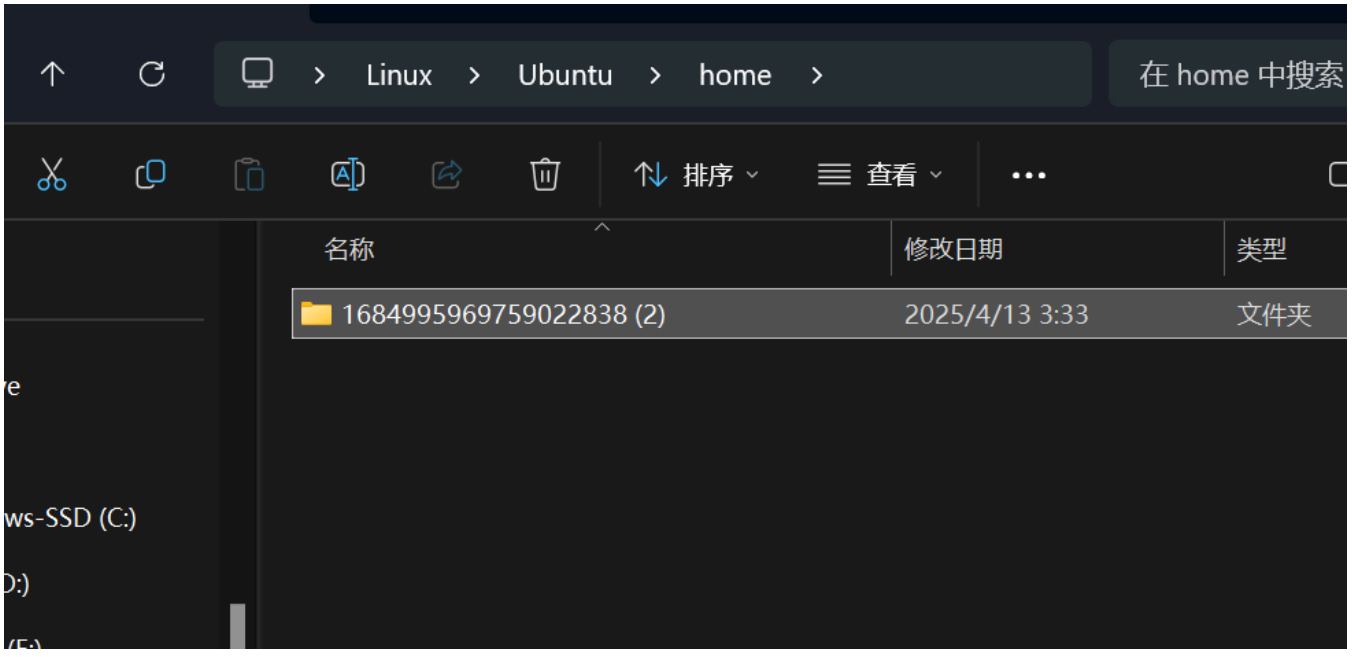
7.以上环境构建就完成了，将实验框架拖入vscode即可开始编写代码

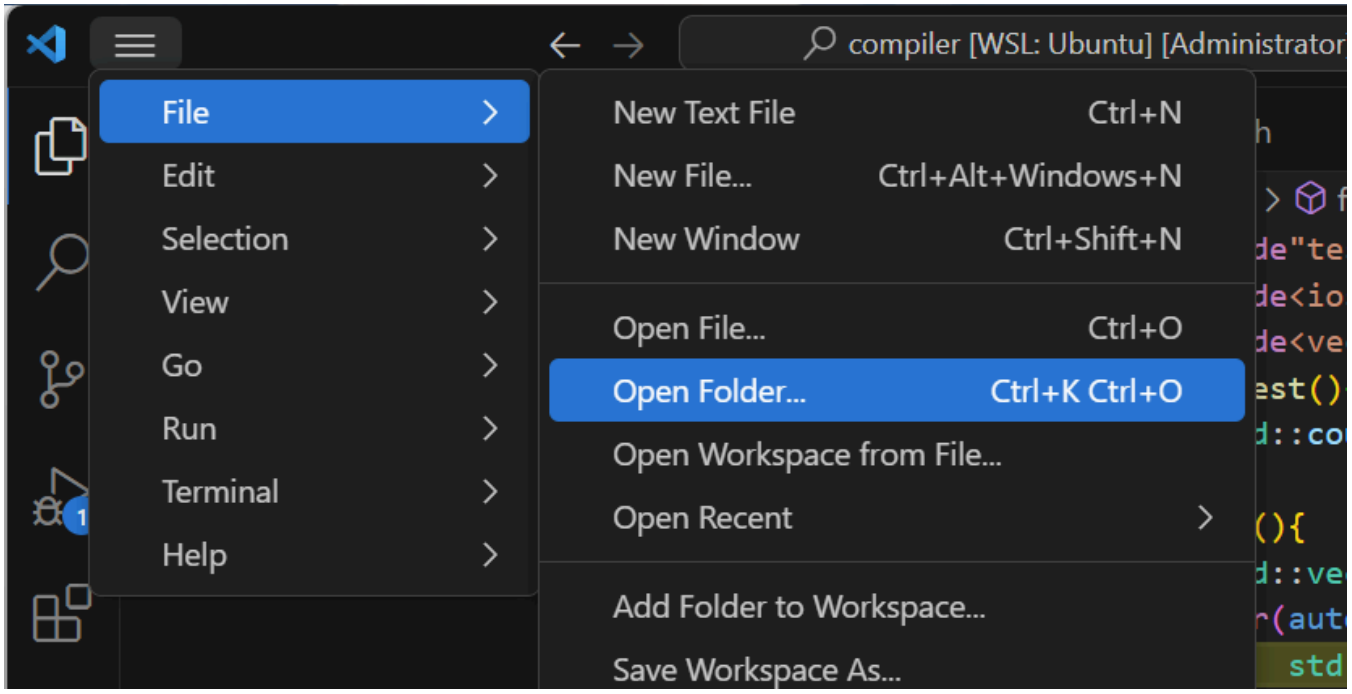


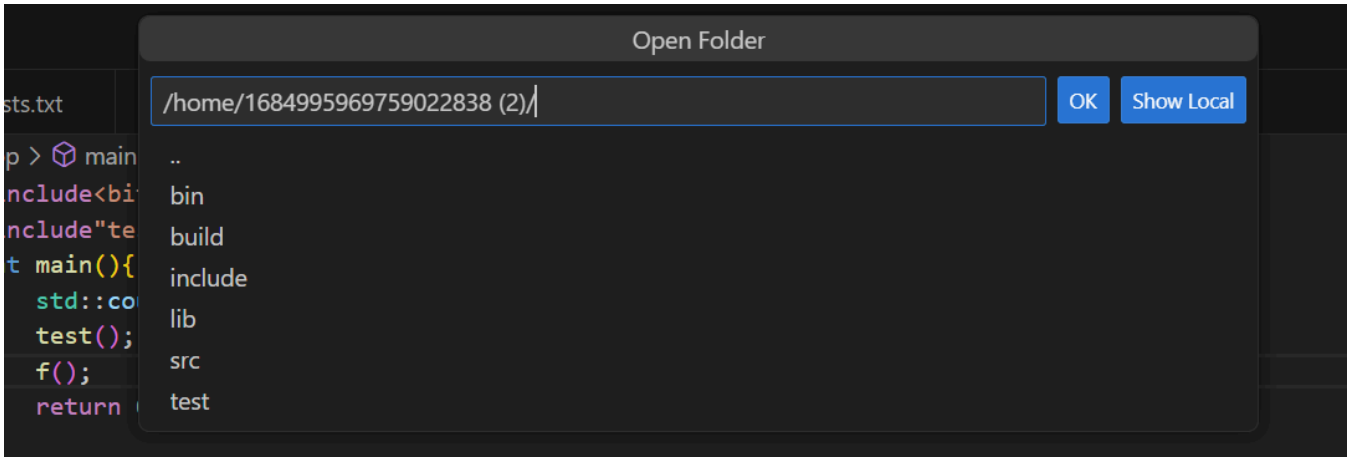
可以使用windows的资源管理器浏览WSL2的文件，以及复制拷贝文件。

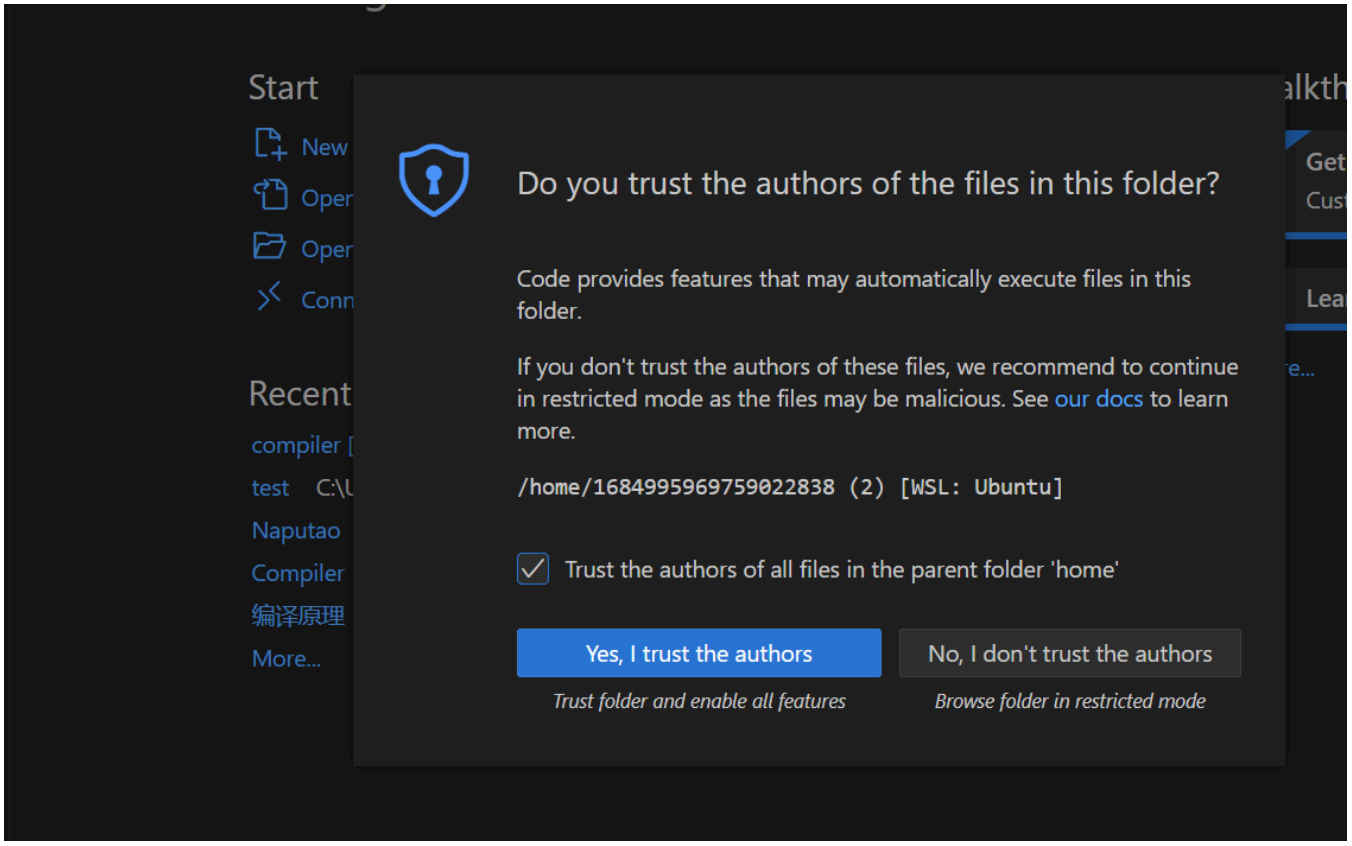
将框架代码拖入到home中，在使用vscode打开即可开始快乐的写代码了

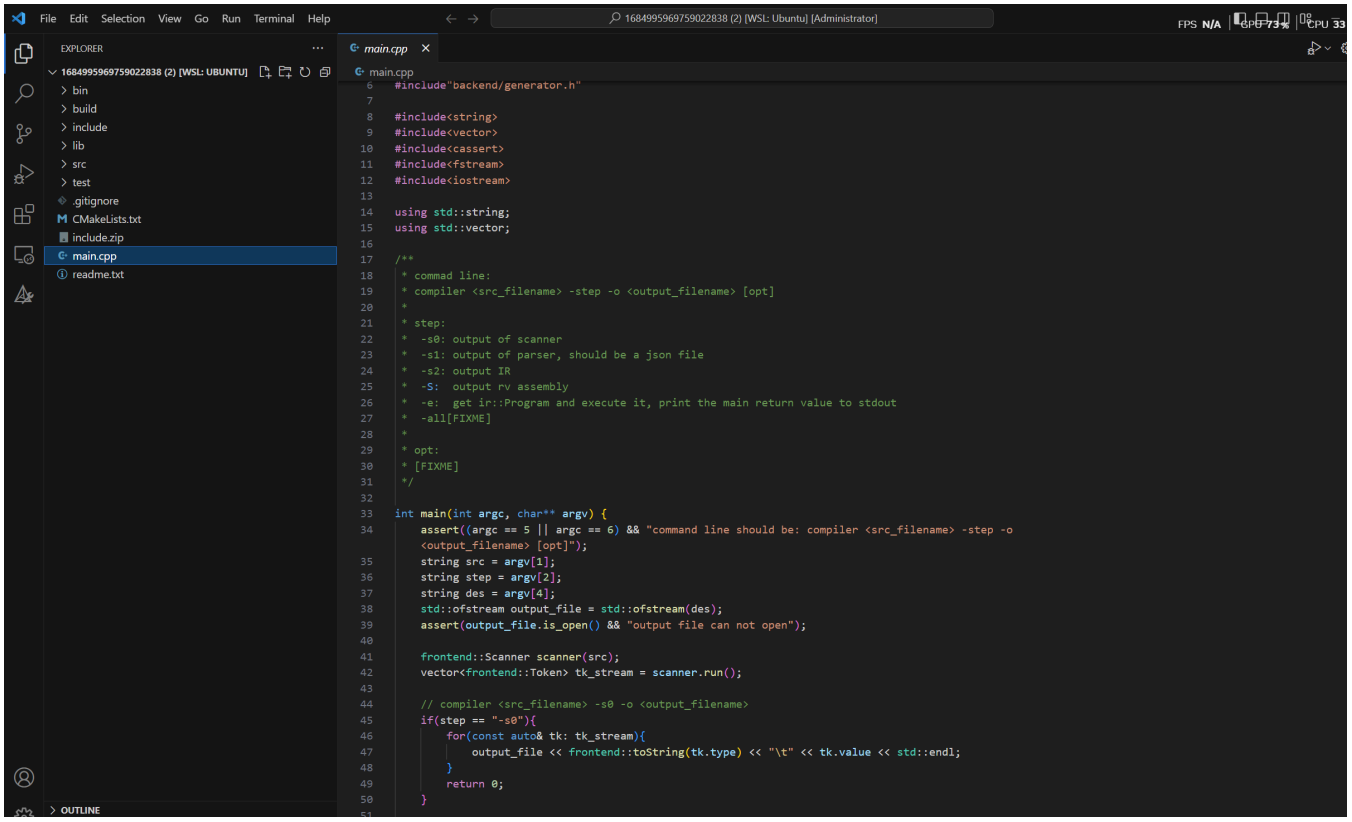





compiler [WSL: Ubuntu] [Administrator
File
Edit
Selection
View
Go
Run
Terminal
Help
New Text File Ctrl+N
New File... Ctrl+Alt+Windows+N
New Window Ctrl+Shift+N
Open File... Ctrl+O
Open Folder... Ctrl+K Ctrl+O
Open Workspace from File...
Open Recent
Add Folder to Workspace...
Save Workspace As...


Open Folder
/home/1684995969759022838 (2)/
OK
Show Local
..
bin
build
include
lib
src
test


Start
New
Oper
Oper
Conn
Recent
compiler [
test C:\U
Naputao
Compiler
编译原理
More...
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.
/home/1684995969759022838 (2) [WSL: Ubuntu]
Trust the authors of all files in the parent folder 'home'
Yes, I trust the authors
No, I don't trust the authors
Trust folder and enable all features
Browse folder in restricted mode


File Edit Selection View Go Run Terminal Help
1684995969759022838 (2) [WSL: Ubuntu] [Administrator]
EXPLORER
1684995969759022838 (2) [WSL: UBUNTU]
bin
build
include
lib
src
test
.gitignore
CMakeLists.txt
include.zip
main.cpp
readme.txt
OUTLINE
main.cpp
#include"backend/generator.h"
#include<string>
#include<vector>
#include<cassert>
#include<fstream>
#include<iostream>
using std::string;
using std::vector;
/**
* commad line:
* compiler <src_filename> -step -o <output_filename> [opt]
*
* step:
* -s0: output of scanner
* -s1: output of parser, should be a json file
* -s2: output IR
* -S: output rv assembly
* -e: get ir::Program and execute it, print the main return value to stdout
* -all[FIXME]
*
* opt:
* [FIXME]
*/
int main(int argc, char** argv) {
assert((argc == 5 || argc == 6) && "command line should be: compiler <src_filename> -step -o <output_filename> [opt]");
string src = argv[1];
string step = argv[2];
string des = argv[4];
std::ofstream output_file = std::ofstream(des);
assert(output_file.is_open() && "output file can not open");
frontend::Scanner scanner(src);
vector<frontend::Token> tk_stream = scanner.run();
// compiler <src_filename> -s0 -o <output_filename>
if(step == "-s0"){
for(const auto& tk: tk_stream){
output_file << frontend::toString(tk.type) << "\t" << tk.value << std::endl;
}
return 0;
}

8.也可以直接将文件拖出，提交作业