**CPP文档**

# 文档管理

**更改历史**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **状态** | **日期** | **负责人** | **更改原有** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 目录

1 文档管理 2

2 目录 3

3 参考文章 4

4 配置MinGW64 4

4.1 下载 4

4.2 解压 5

4.3 配置MinGW64环境变量 5

4.4 查看gcc版本 5

5 安装vscode 6

5.1 下载 6

5.2 解压 6

5.3 安装Chinese、C/C++插件 7

5.4 重启 7

6 开发C/C++代码 7

6.1 创建项目文件夹 7

6.2 编写test.c 7

6.3 设置c\_cpp\_properties.json（非必须） 8

6.4 设置tasks.json（必须） 9

6.5 设置launch.json（非必须） 10

6.6 编译和运行一个c代码 11

6.6.1 编译 11

6.6.2 运行 12

6.7 编译和运行多个c代码 12

7 git 13

7.1 VSCode连接GitHub报错 13

7.2 VSCode Git提交失败 14

7.3 VsCode使用git提交一直显示在提交 14

8 Makefile 14

8.1 下载cmake 14

8.2 将make中的bin添加到环境变量 15

8.3 vscode中安装Makefile tools插件 15

8.4 编译 15

9 cmake（vscode编译多个.c文件和.h文件） 16

9.1 下载cmake 16

9.2 将cmake中的bin添加到环境变量 16

9.3 vscode中安装cmake和cmake tool插件 16

# 参考文章

<https://blog.csdn.net/weixin_52159554/article/details/134406628>

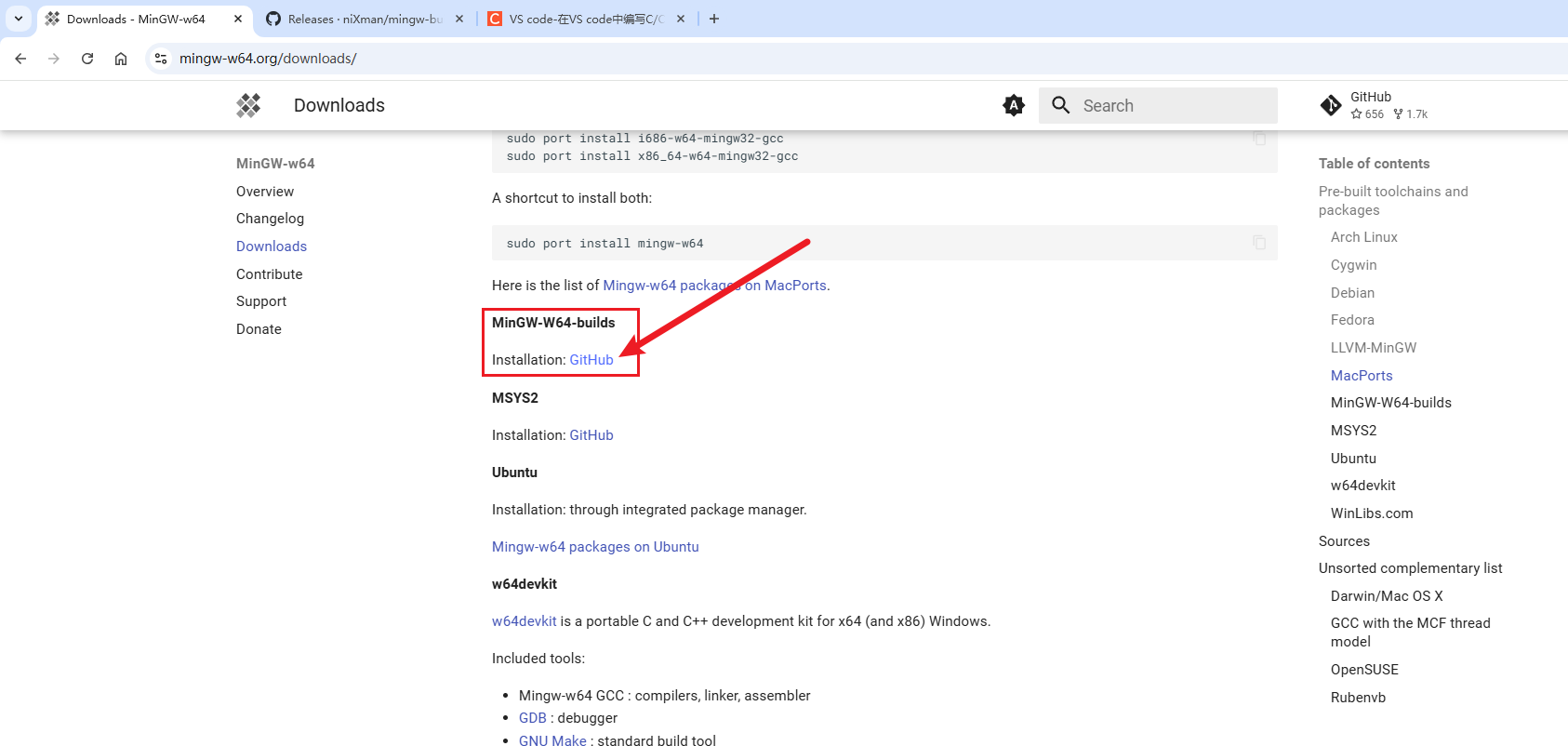
# 配置MinGW64

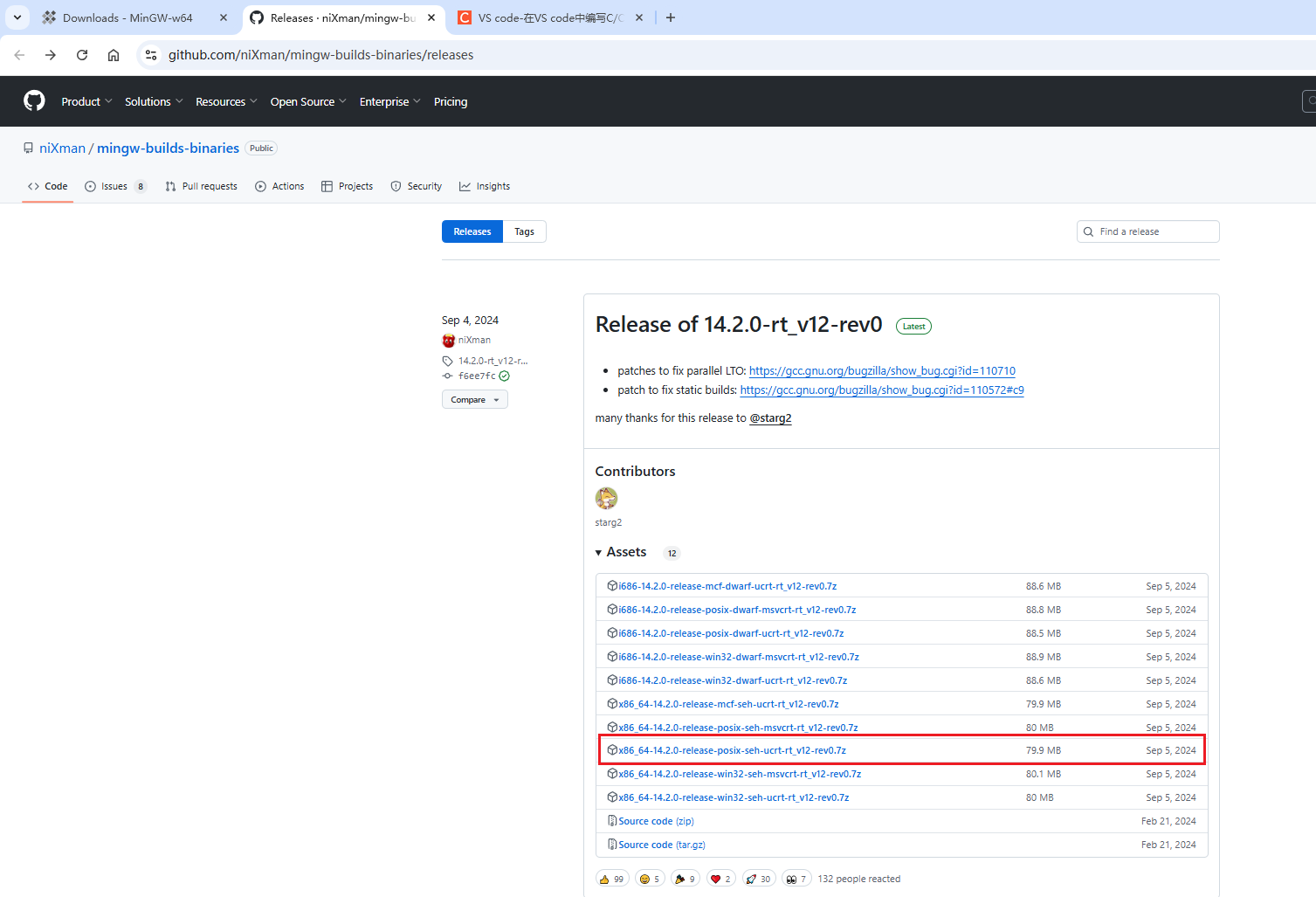
## 下载

<https://www.mingw-w64.org/downloads/>

<https://github.com/niXman/mingw-builds-binaries/releases>

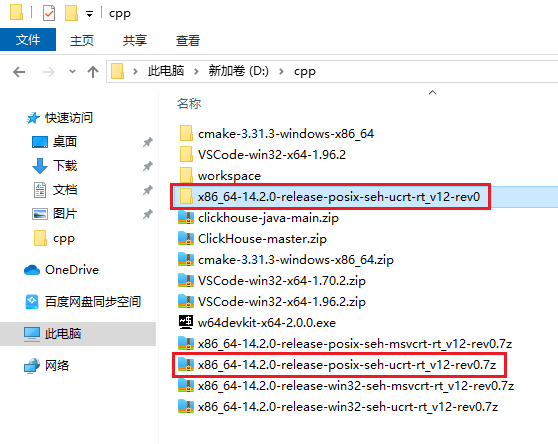
下载：[x86\_64-14.2.0-release-posix-seh-ucrt-rt\_v12-rev0.7z](https://github.com/niXman/mingw-builds-binaries/releases/download/14.2.0-rt_v12-rev0/x86_64-14.2.0-release-posix-seh-ucrt-rt_v12-rev0.7z)





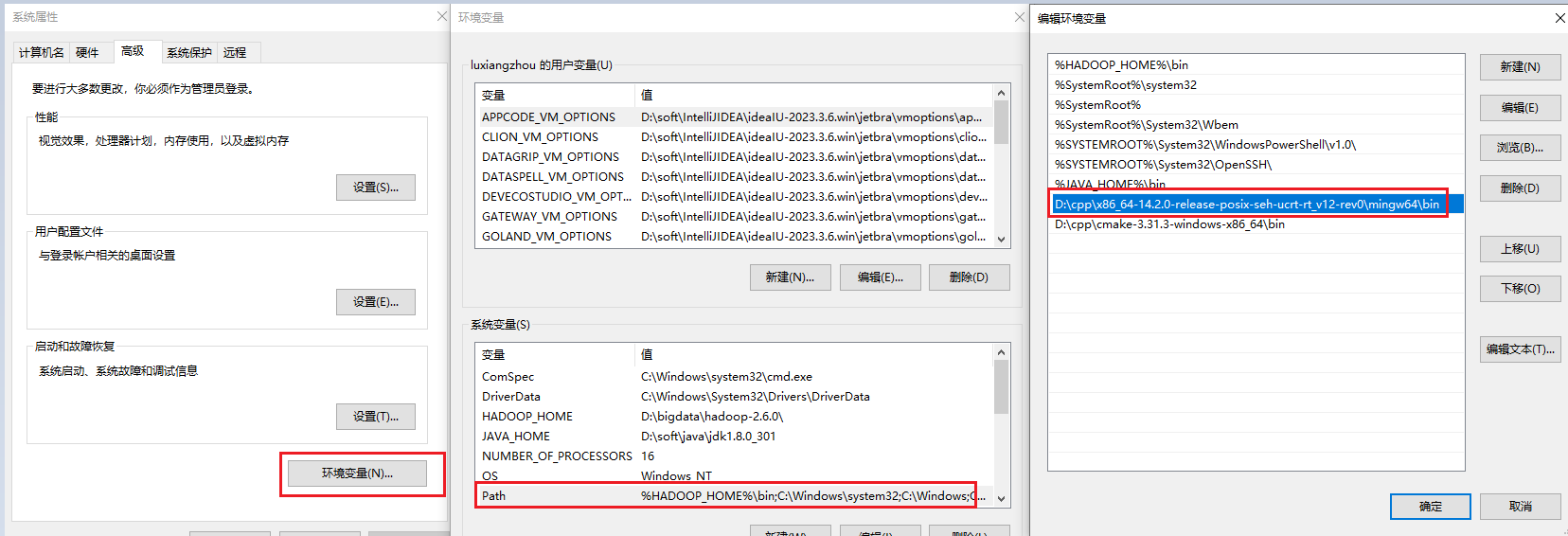
## 解压

**解压x86\_64-14.2.0-release-posix-seh-ucrt-rt\_v12-rev0.7z**



## 配置MinGW64环境变量

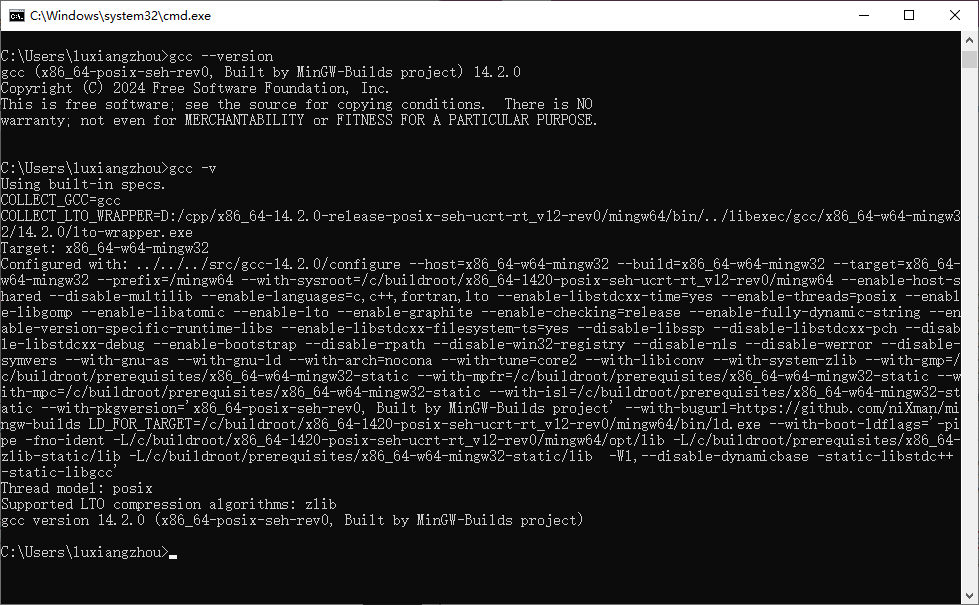
D:\cpp\x86\_64-14.2.0-release-posix-seh-ucrt-rt\_v12-rev0\mingw64\bin



## 查看gcc版本

gcc --version

gcc -v



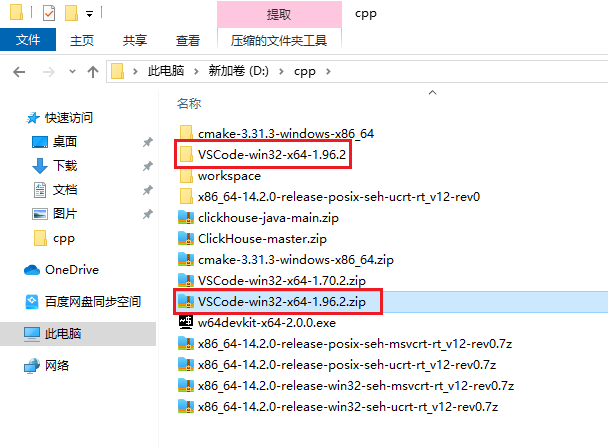
# 安装vscode

## 下载

<https://code.visualstudio.com/>

## 解压

**解压VSCode-win32-x64-1.96.2.zip，并打开vscode**



## 安装Chinese、C/C++插件



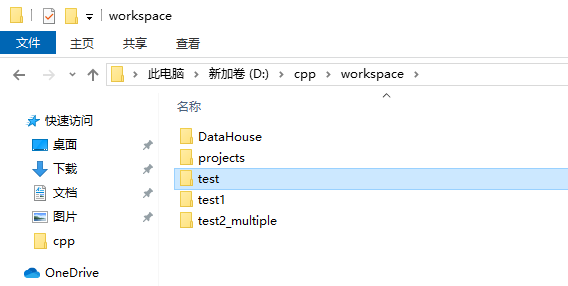


## 重启

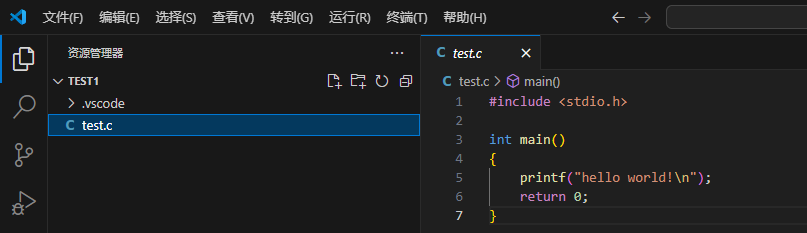
重启VSCode，这一步很重要，让前面的设置生效，要不然后面会出现问题的。

# 开发C/C++代码

## 创建项目文件夹



## 编写test.c



## 设置c\_cpp\_properties.json（非必须）

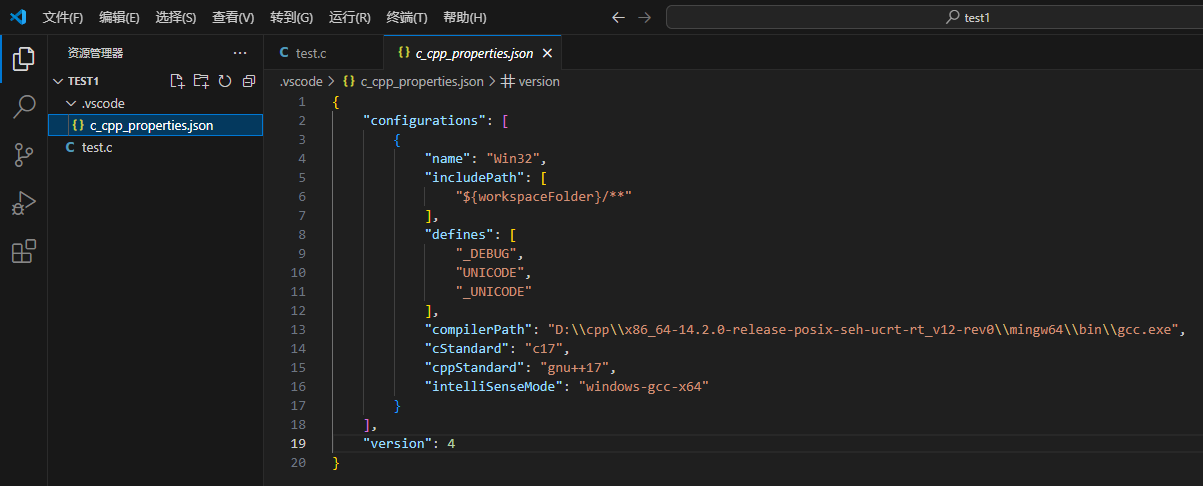
**设置C/C++编译的选项：c\_cpp\_properties.json**

 鼠标点在.c的源文件内部，按ctrl+shift+p快捷键，在弹出的界面中选择：【C/C++：编辑配置(UI)】，然后进入设置页面。





其他的选项就不再修改。当修改完以上内容后，就在你打开的文件夹下生成了一个.vscode的文件夹，这个文件夹很重要，里边放的都是配置文件，以后是可以复用的。同时你看到在.vscode文件夹下生一个新的文件c\_cpp\_properties.json,这个文件中是关于C/C++的配置属性。



## 设置tasks.json（必须）

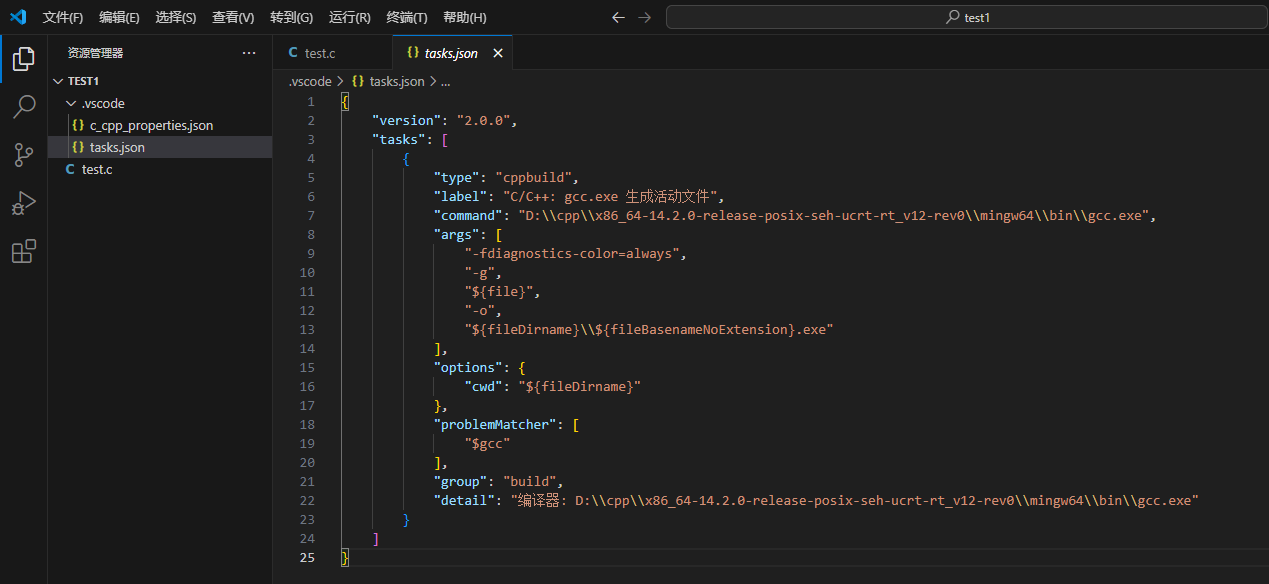
**创建执行任务：tasks.json**

1、打开test.c，打开“终端--->配置任务”，才能在配置文件框中找到【C/C++:gcc.exe 生成活动文件】；

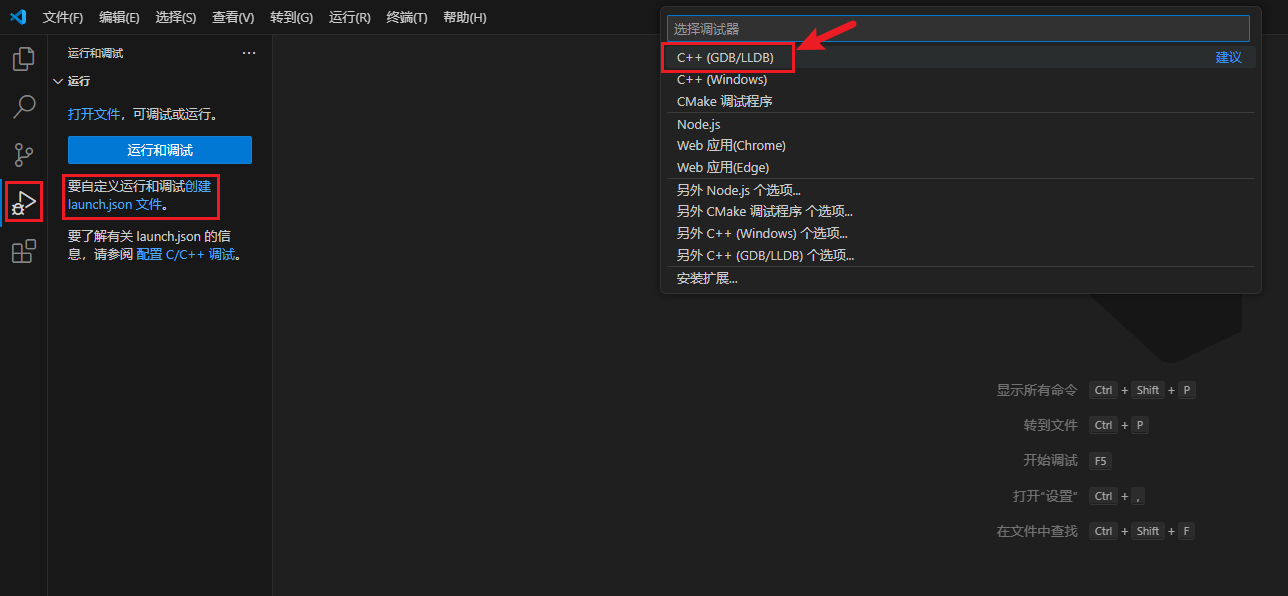
2、在.vscode文件夹中生成tasks.json文件；

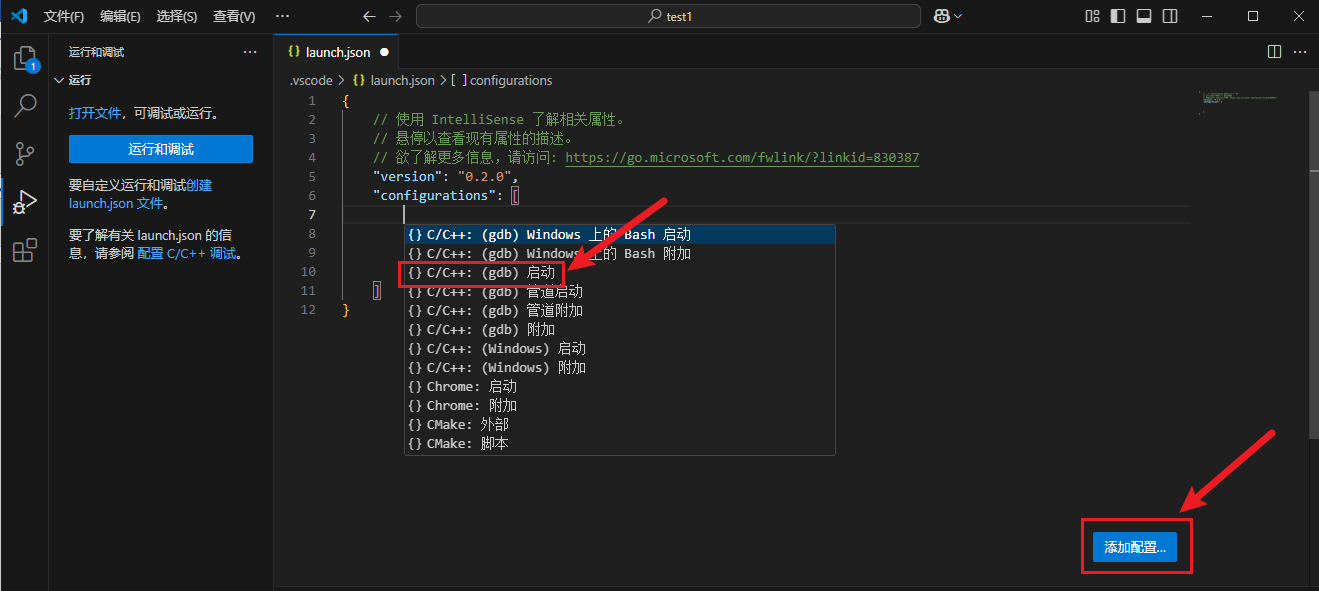


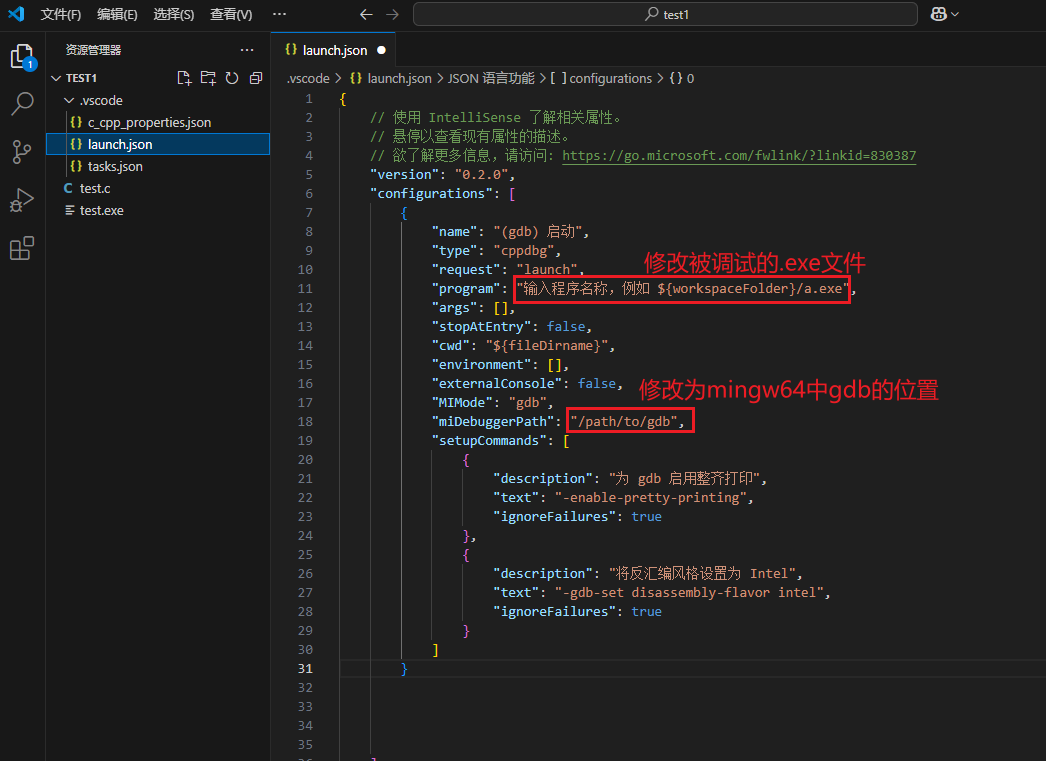




## 设置launch.json（非必须）







修改"program"为："**${workspaceFolder}\\${workspaceRootFolderName}.exe**"，这个内容和tasks.json中的可支持文件的路径匹配。

修改"miDebuggerPath"为："**D:\\cpp\\x86\_64-14.2.0-release-posix-seh-ucrt-rt\_v12-rev0\\mingw64\\bin\\gdb.exe**

"，这个路径根据自己实际放MinGW-w64的文件路径调整。

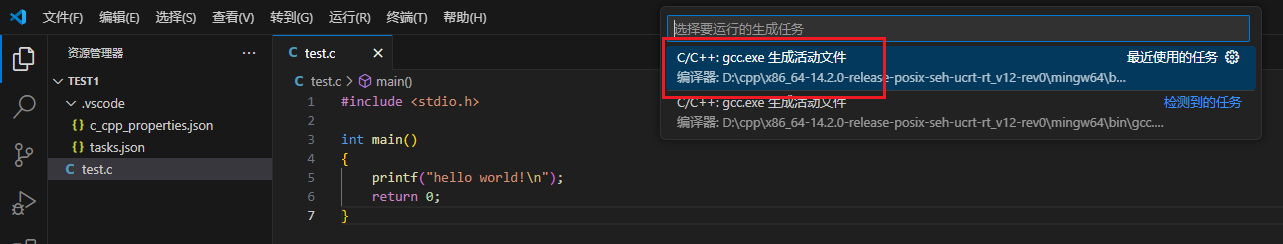
## 编译和运行一个c代码

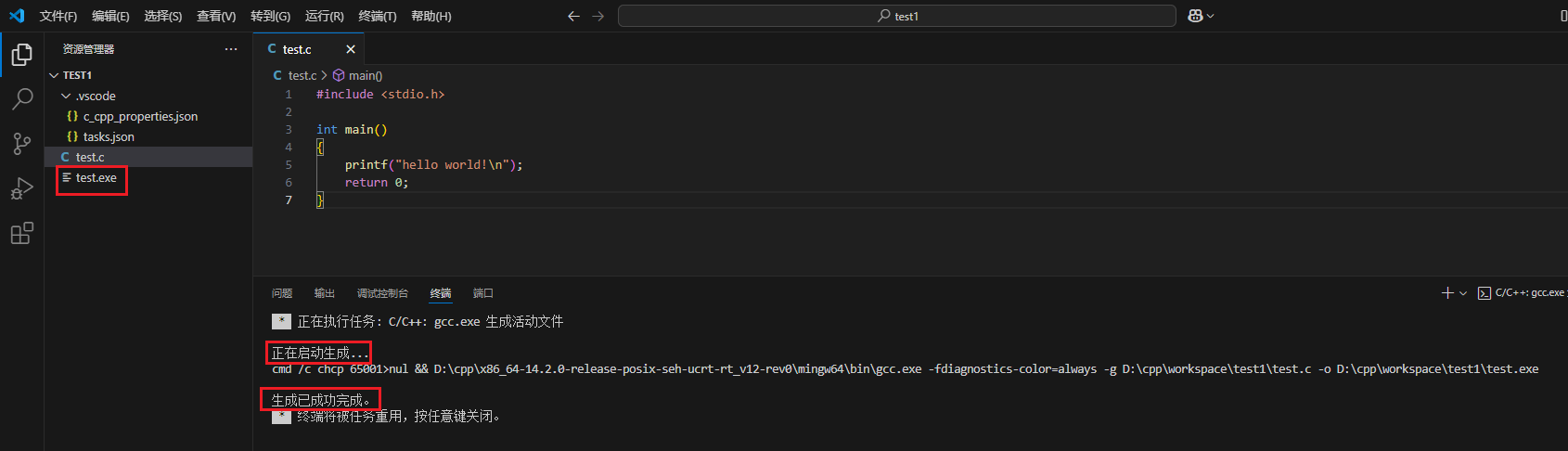
### 编译

1、终端--->运行生成任务--->C/C++:gcc.exe 生成活动文件

2、生成可执行文件 “.exe ”文件；



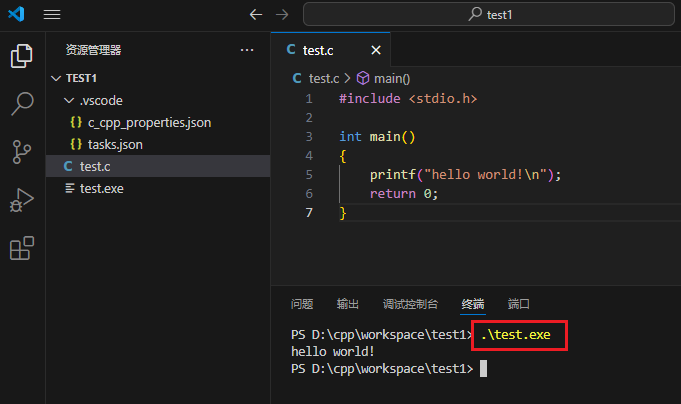




### 运行

按键盘的ctrl+`快捷键，打开VSCode自带的终端，就可以运行代码了。

这时输入：.\可执行程序的名字，即可以执行代码。



## 编译和运行多个c代码

修改tasks.json文件：

编译多个源代码文件：

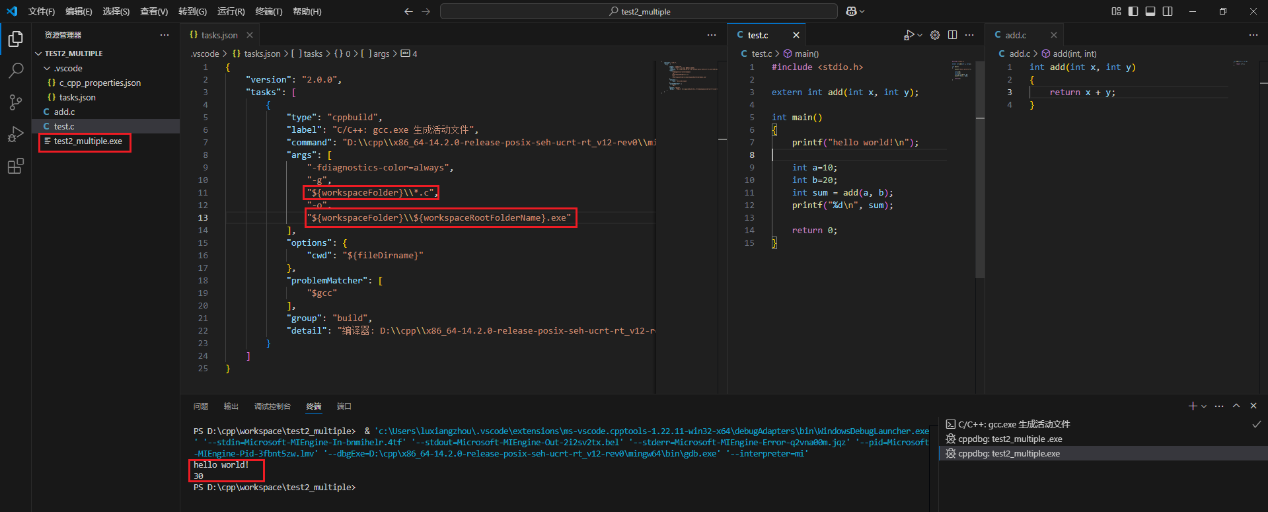
通过把"${file}"修改为"${workspaceFolder}\\\*.c"，可以构建当前工作区中的所有C文件。

通过把"${file}"修改为"**${workspaceFolder}\\\*.cpp**"，可以构建当前工作区中的所有C++文件。

修改编译输出的文件名：

通过把"${fileDirname}\\${fileBasenameNoExtension}.exe"，修改为:"${workspaceFolder}\\myProgram.exe",可以把编译输出的文件名硬编码为myProgram.exe，而不是与C文件同名。

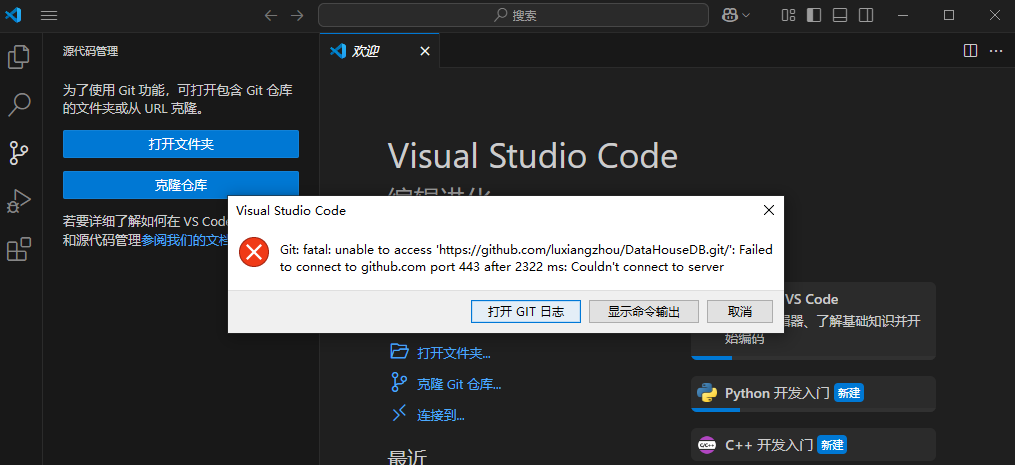
通过把"${fileDirname}\\${fileBasenameNoExtension}.exe"，修改为:"**${workspaceFolder}\\${workspaceRootFolderName}.exe**",可以把编译输出的文件名为：打开的工作区文件夹名字.exe。



# git

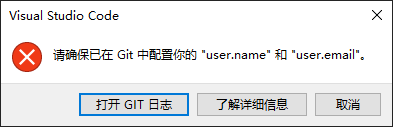
## VSCode连接GitHub报错

修改host文件，把140.82.114.4 github.com添加到C:\Windows\System32\drivers\etc里的hosts文件的最后一行，然后保存。



## VSCode Git提交失败

请确保已在 Git 中配置你的"user,name"和"user,email"。

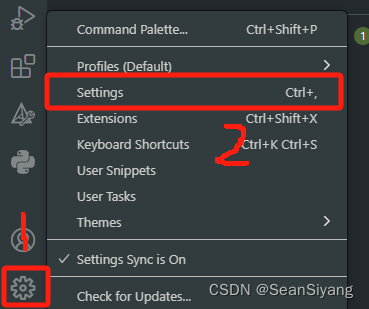


git config --global user.name "luxiangzhou"

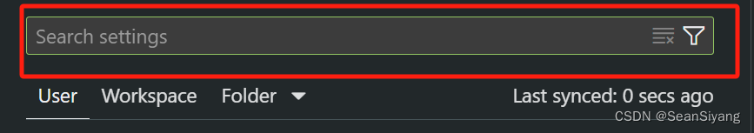
git config --global user.email [472773804@qq.com](mailto:472773804@qq.com)

## VsCode使用git提交一直显示在提交

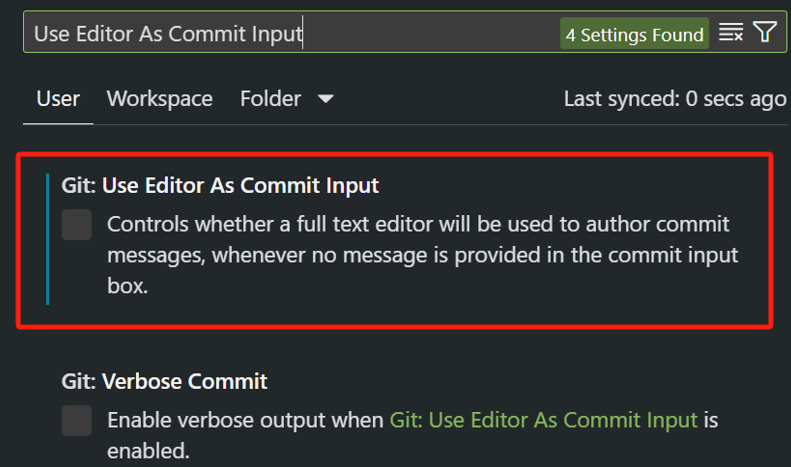
打开VSCode的设置：



在下面的窗口中输入：Use Editor As Commit Input



然后将下方红色框中的勾选取消：



此时再次尝试提交，就可以发现正常提交Github了。

# Makefile

## Makefile介绍

Makefile文件是make工具使用的配置文件，它定义了如何自动化构建项目的规则和命令。Makefile文件的主要作用是指定如何编译和链接程序，以及管理文件之间的依赖关系，从而高效的构建过程。

### Makefile基本结构

一个典型的Makefile文件是由目标（target）、依赖（dependencies）和命令（commands）组成。

基本语法如下：

target：denpendencies

command1

command2

…

#### **target**

要生成的文件，通常是可执行文件、目标文件、或者其他文件，也可以是一个**伪目标**（例如clean，不实际生成文件，只是执行某些命令）。

#### **denpendencies**

生成目标文件所需的文件。如果这些依赖文件有**任何变化**，make就会执行相应的命令来更新目标。

#### **commands**

生成目标的命令。通常是编译、链接命令或者是其他操作系统命令。这些命令必须以Tab键开头。

### Makefile变量和变量赋值

变量可以对许多地方使用，比如目标，依赖。或者命令。  
变量的赋值可以使用：  =   ?=   :=   +=  
变量的使用：通过**$()** 来完成变量的引用。

### Makefile自动化变量

自动化变量就是不用定义且会随着上下程序的不同而发生变化的变量叫做自动化变量。 这里介绍三个最常用的自动化变量：  
$@: 表示所有目标  
$< :表示第一个依赖文件，如果依赖模式是%，那么他就表示一系列文件。 （%为通配符，类似 linux 上的 \*）  
$^ :表示所有依赖。

### Makefile常用函数

#### **wildcard函数**

格式： $ (wildcard PATTENR)  
功能： 展开指定的目录

#### **notdir 函数**

 格式： $ (notdir $ (var) )  
功能：去掉路径。

#### **dir 函数**

格式： $(dir )  
功能：取出目录，这里的目录指的是最后一个反斜杠/ 之前的部分，如果没有反斜杠/就返回当前。

#### **patsubst 函数**

格式： $(patsubst 原文件，目标文件，文件列表）  
功能：替换文件后缀

#### **foreach 函数**

格式：$（foreach <var>,<list>,<text>）  
功能:把参数 <list> 中的单词逐一取出放到参数 <var> 所指定的变量中，然后再执行 <text> 所包含的表达式。每一次 <text> 会返回一个字符串。

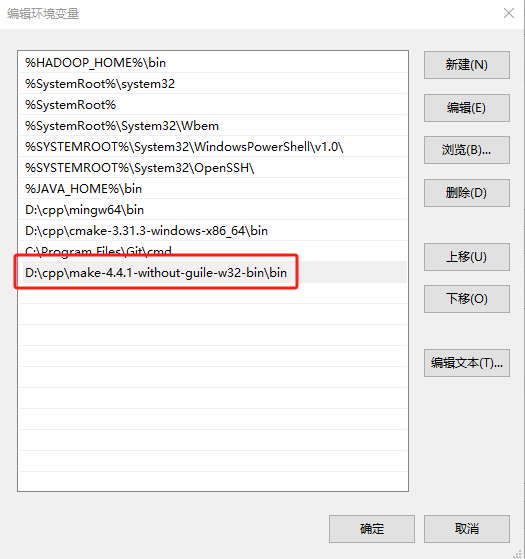
## make安装

### 下载make

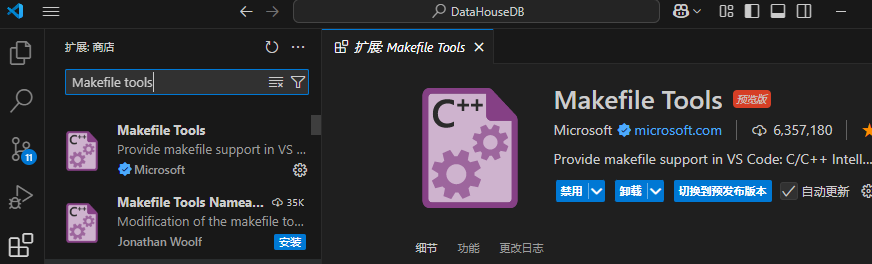
<https://sourceforge.net/projects/ezwinports/files/>

下载[make-4.4.1-without-guile-w32-bin.zip](https://sourceforge.net/projects/ezwinports/files/make-4.4.1-without-guile-w32-bin.zip/download)

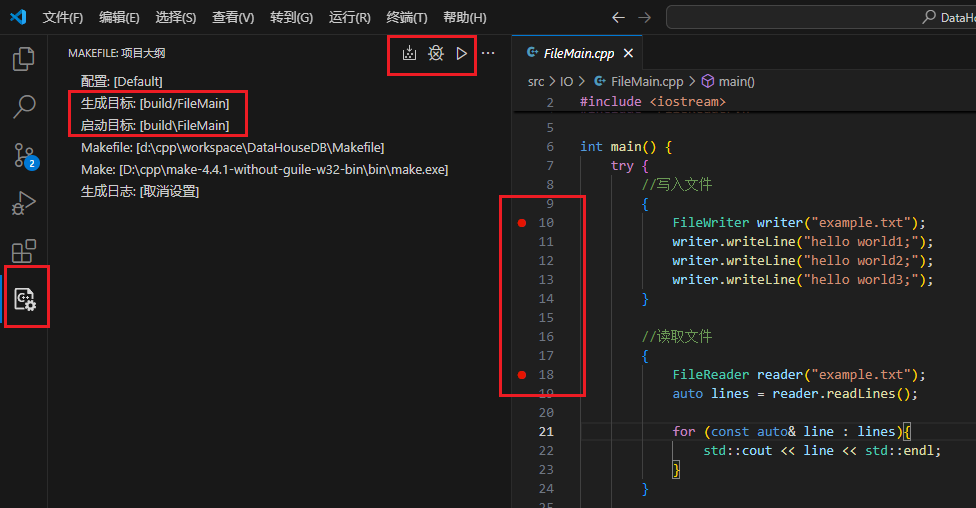
### 将make中的bin添加到环境变量

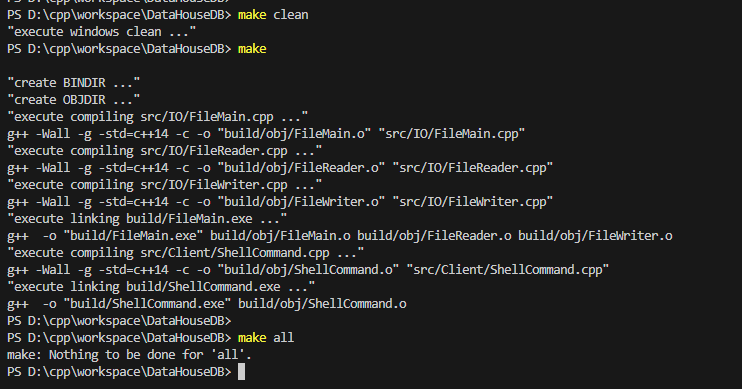


### vscode中安装Makefile tools插件



### 编译



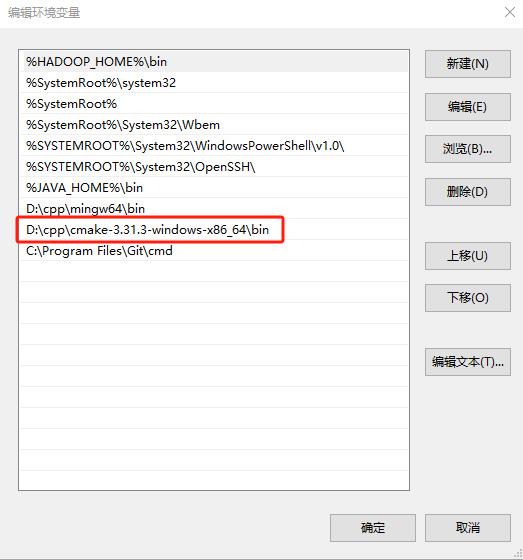


# cmake（vscode编译多个.c文件和.h文件）

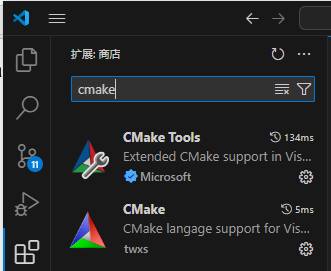
## 下载cmake

<https://cmake.org/download/>

## 将cmake中的bin添加到环境变量



## vscode中安装cmake和cmake tool插件



# 编译、链接、构建

在C++编程中，编译和构建是两个关键步骤，它们将源代码转换为可执行的程序。以下是一个简要的介绍和步骤指南：

## 编译（Compilation）

编译是将源代码（.cpp 文件）转换为目标文件（.o 或 .obj 文件）的过程。目标文件是机器代码的一种中间形式，通常包含二进制指令，但还不能直接执行。

使用编译器：

最常用的C++编译器是 g++（GNU Compiler Collection 的 C++ 编译器）。

编译命令示例：g++ -c main.cpp -o main.o

-c 选项表示只编译和汇编，不进行链接。

-o 选项指定输出文件名。

## 链接（Linking）

链接是将多个目标文件（以及必要的库文件）组合成一个可执行文件的过程。链接器会处理外部函数和变量的引用，并生成最终的可执行文件。

链接命令：

链接示例：g++ main.o -o my\_program

-o 选项指定输出可执行文件的名称。

## 构建（Build）

构建是一个更广泛的概念，通常包括编译、链接以及可能的其他步骤（如预处理、资源文件处理等）。在现代开发环境中，构建过程通常由构建系统或构建工具来管理。

构建系统：

Makefile：使用 make 工具来自动化编译和链接过程。Makefile 文件中定义了规则和依赖关系。

CMake：一个跨平台的构建系统，可以生成适用于不同编译器的 Makefile 或其他构建文件。

IDE（集成开发环境）：如 Visual Studio、CLion、Eclipse CDT 等，这些 IDE 通常内置了构建工具，用户只需点击按钮即可完成构建过程。

示例

假设我们有一个简单的 C++ 程序 main.cpp：

cpp

#include <iostream>

int main() {

std::cout << "Hello, World!" << std::endl;

return 0;

}

### 使用g++编译和链接

编译：

sh

g++ -c main.cpp -o main.o

链接：

sh

g++ main.o -o my\_program

现在，你可以运行生成的可执行文件 my\_program：

sh

./my\_program

输出：

Hello, World!

### 使用Makefile编译和链接

创建一个简单的 Makefile：

makefile

# Makefile

# 编译器

CXX = g++

# 编译选项

CXXFLAGS = -Wall -g

# 目标文件

OBJS = main.o

# 可执行文件

TARGET = my\_program

# 链接目标文件

$(TARGET): $(OBJS)

$(CXX) $(OBJS) -o $(TARGET)

# 编译源文件

%.o: %.cpp

$(CXX) $(CXXFLAGS) -c $< -o $@

# 清理

clean:

rm -f $(OBJS) $(TARGET)

然后，你可以使用 make 命令来构建程序：

sh

make

清理构建文件：

sh

make clean

总结

编译：将源代码转换为目标文件。

链接：将目标文件组合成可执行文件。

构建：包括编译、链接以及其他构建步骤，通常由构建系统或 IDE 管理。

通过理解这些步骤和工具，你可以更高效地开发和构建 C++ 程序。

# gdk