使用vscode编写调试c程序

1. **概述**

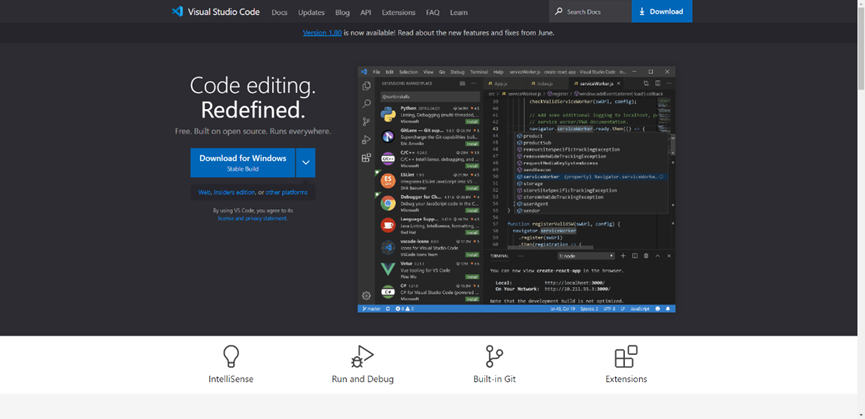
Visual Studio Code (简称：VS Code) 是一个轻量级但功能强大的源代码编辑器，可以在桌面上运行，并且适用于 Windows、macOS 和 Linux。VS Code支持语法高亮、代码自动补全、代码重构、查看定义功能，并且内置了命令行工具和 Git 版本控制系统。

VS Code本质上只是一个编辑器，要想让其编译程序必须安装第三方的插件。好在VS Code提供了强大的第三方扩展，这也是VS Code越来越受欢迎的原因。

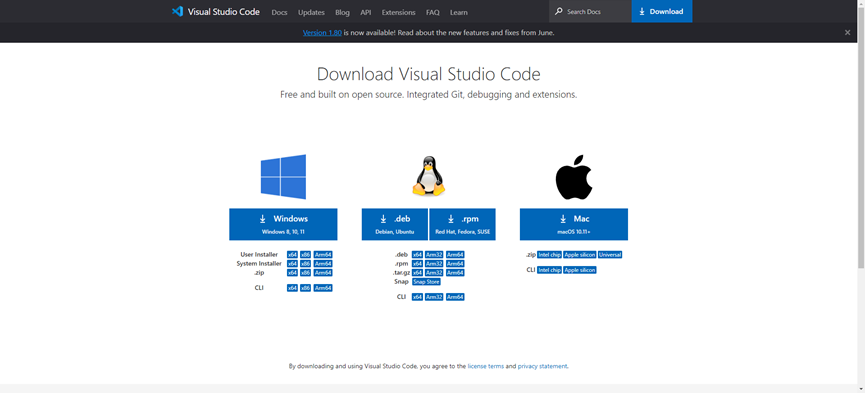
1. **下载 VS Code**

VS官方网址：<https://code.visualstudio.com/>

打开链接后如下所示，网页会根据你的操作系统匹配对应的版本，直接点击左边的下载按钮下载适合你电脑的版本。

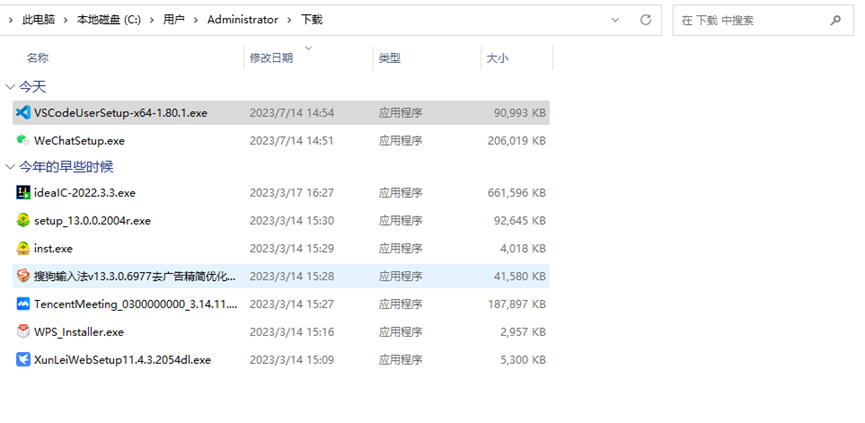


如果需要查看其他版本，可以点击右上角的Download按钮，如下所示，选择你需要的版本下载。

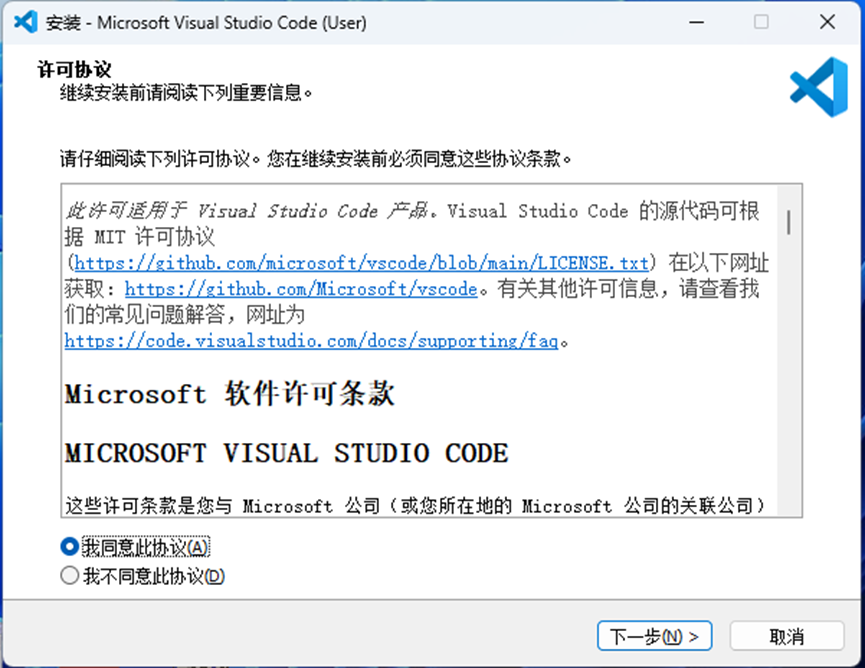


1. **安装VS Code**

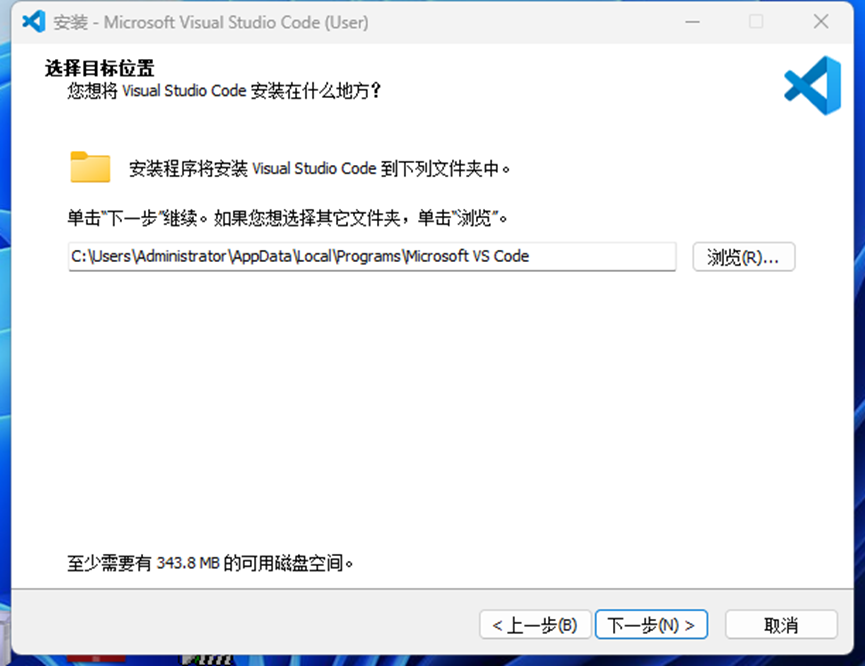
以 Windows 平台为例，下载下来的安装器名字类似于 VSCodeUserSetup-x64-1.80.1.exe。



双击安装文件，跳出以下页面，选择我同意此协议

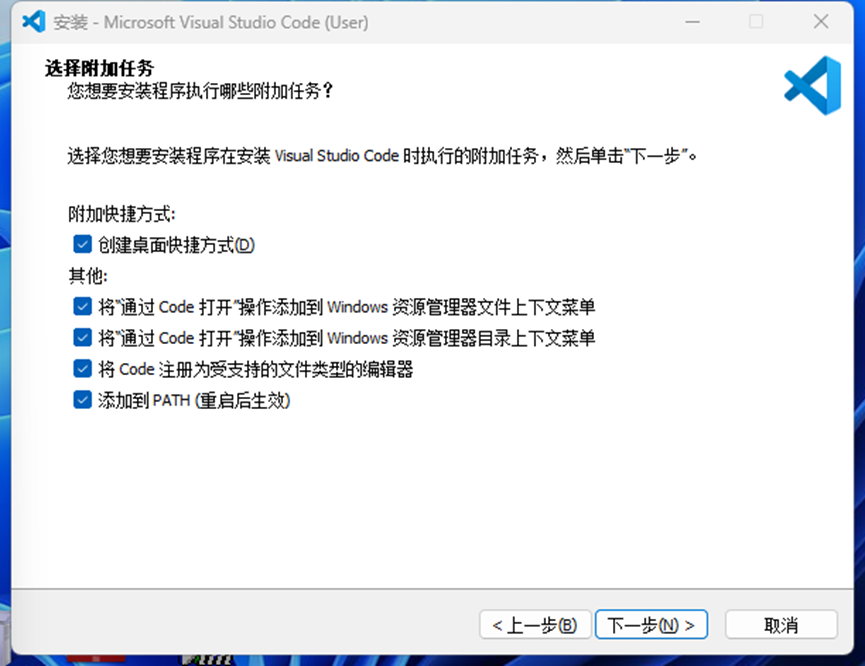


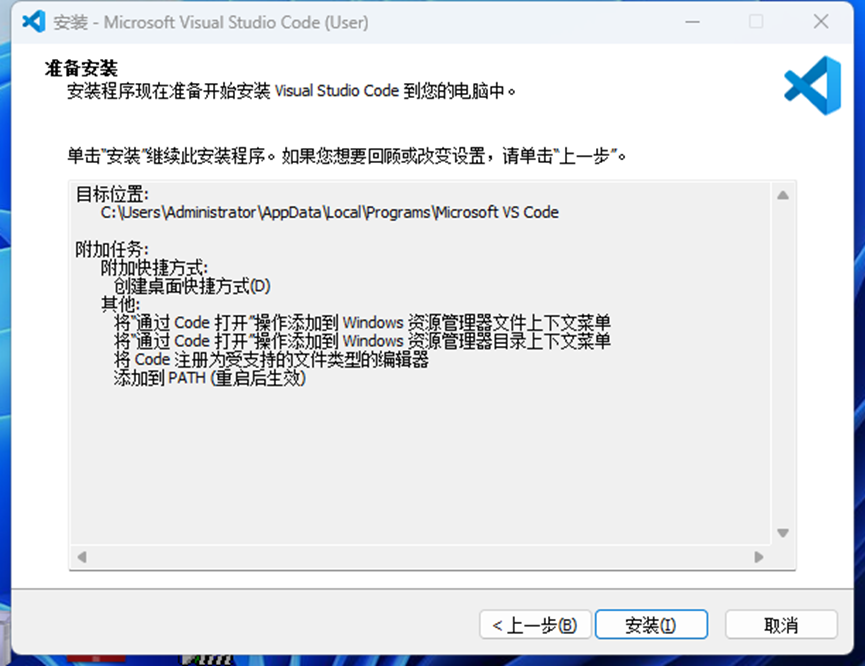
选择安装位置，点击下一步。



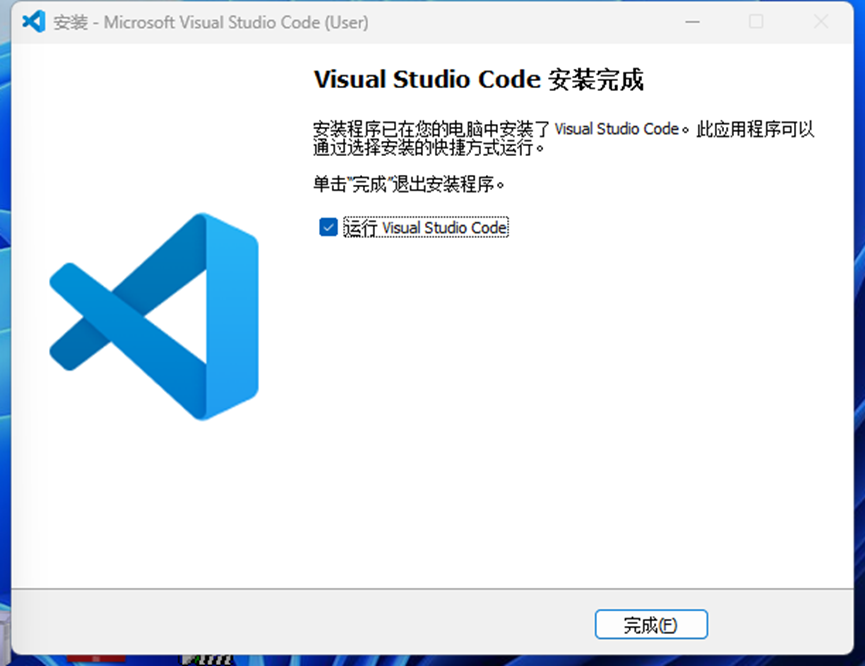


勾选创建桌面快捷方式，和其他下的选项，继续下一步。

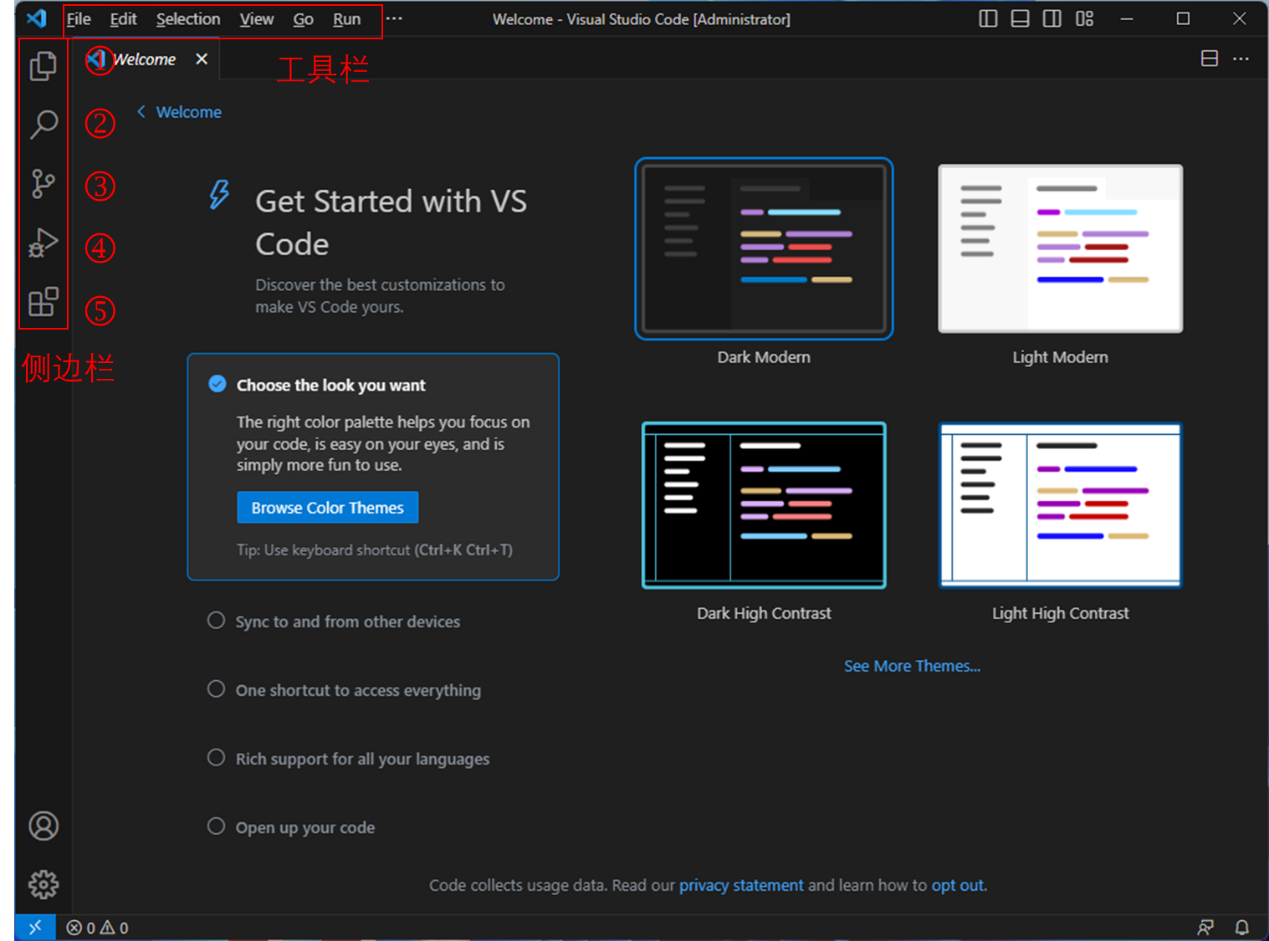




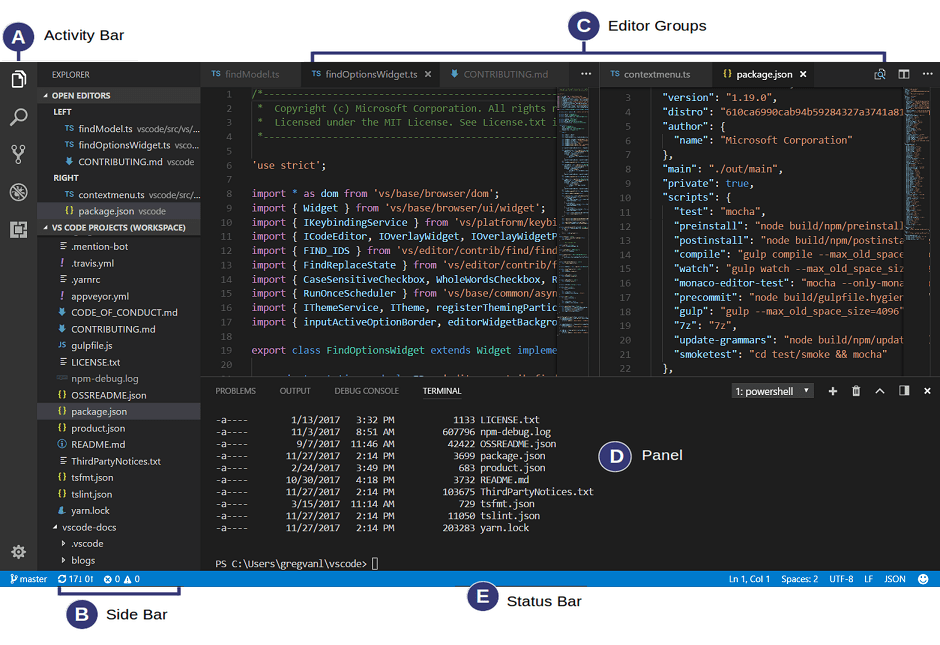
点击安装



安装完成后第一次启动页面如下：



1. **VS Code基本布局**

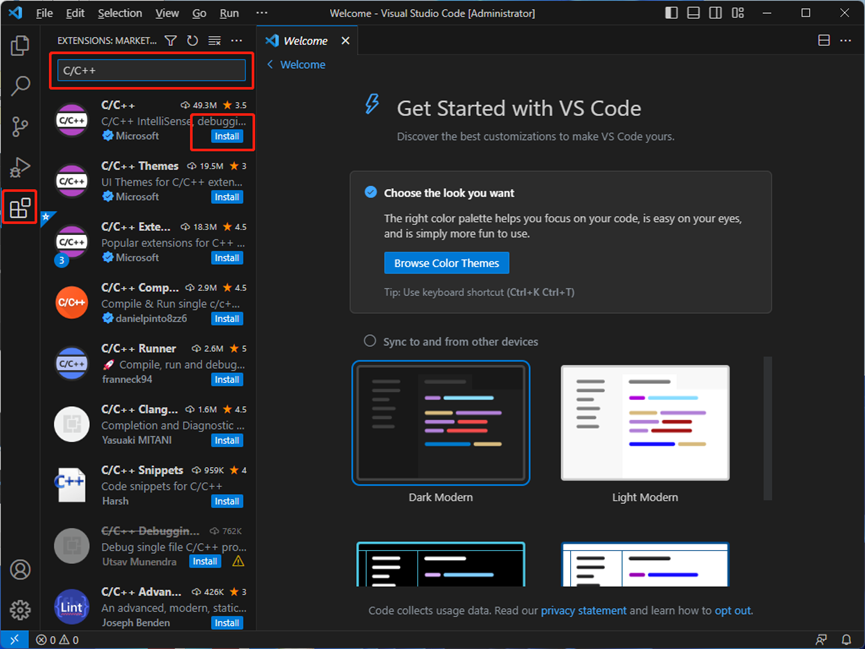


VS Code 具有简单直观的布局，可最大化为编辑器提供的空间，同时留出足够的空间来浏览和访问文件夹或项目的完整上下文。UI 分为五个主要区域：

* A：活动栏- 位于最左侧，可让您在视图之间切换。
* B：主侧边栏- 包含不同的视图（例如资源管理器），以在您处理项目时为您提供帮助。
* C：编辑器- 编辑文件的主要区域。您可以垂直和水平并排打开任意多个编辑器。
* D: 面板- 编辑器区域下方用于查看视图的附加空间。默认情况下，它包含输出、调试信息、错误和警告以及集成终端。面板还可以向左或向右移动以获得更多垂直空间。
* E：状态栏- 有关打开的项目和您编辑的文件的信息。

1. **安装 C/C++ 语言插件**

点击左侧活动栏第⑤个按钮，打开对应的侧边栏，在搜索框中搜索安装“C/C++”语言插件，注意代表官方的“Microsoft”标识。点击Install，安装完成后请重启VS Code



1. **获取 MinGW-w64 编译器软件包**

**6.1 下载MinGW-w64**

VS Code (搭配 "C/C++" 插件) 并不足以使 C/C++ 语言编写的代码能跑起来。C语言需要先经过编译，构建出可直接执行的程序文件，才能跑起来。C/C++ 语言的编译器有几大派系，不同操作系统流行使用的也有所不同，这里以比较有代表性的 GCC 为例。GCC 在 Windows 系统上的其中一个实现叫做 MinGW-w64，下载地址为：

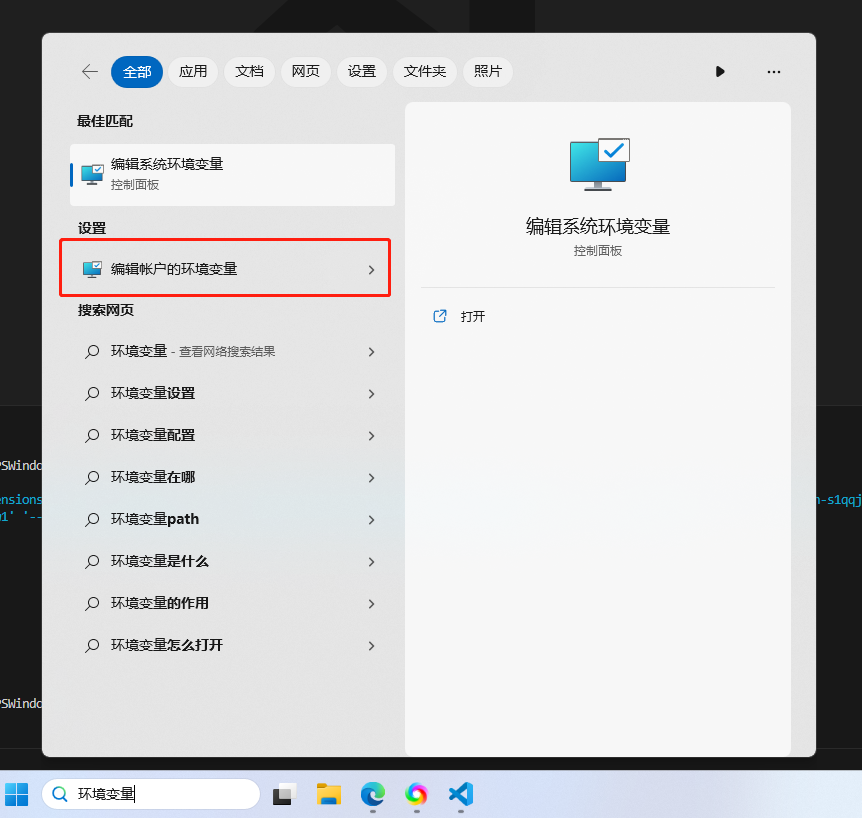
https://github.com/niXman/mingw-builds-binaries/releases

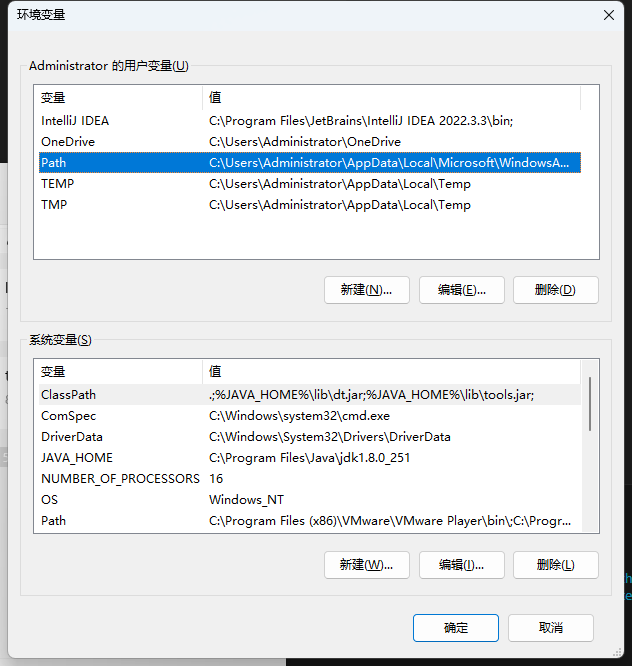
下载的文件类似：x86\_64-13.1.0-release-win32-seh-ucrt-rt\_v11-rev1.7z，解压后里面有一个mingw64的文件夹

将其剪切复制到一个易于管理的目录，建议直接选择 C 盘根目录

**6.2 配置环境变量**

在系统搜索栏中输入“环境变量”，点击“编辑账户的环境变量”。



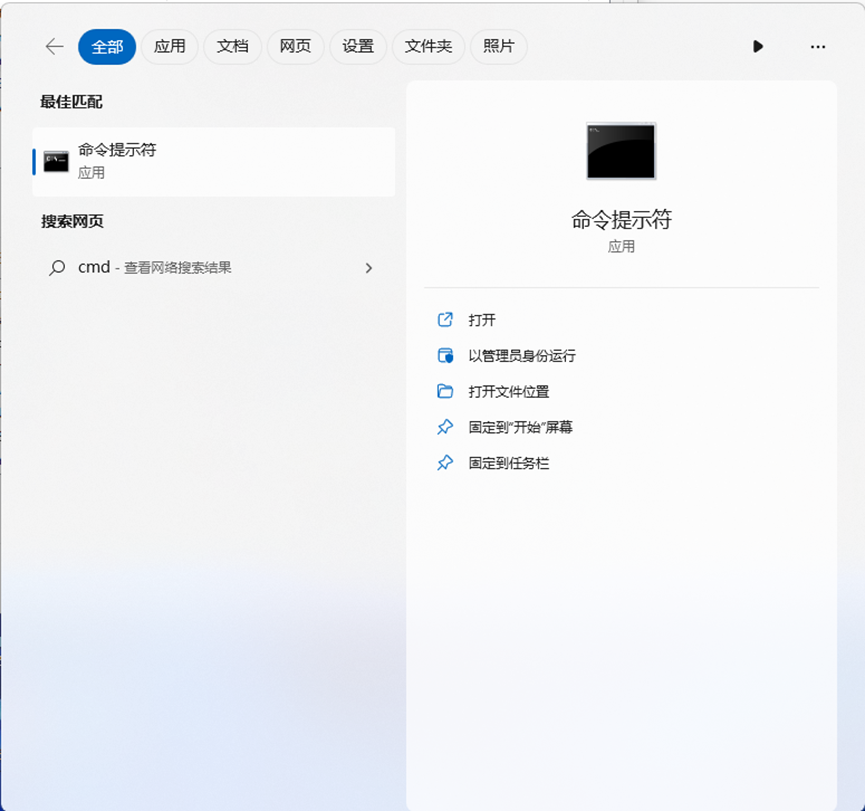


双击Path，添加ming64的bin目录路径，如下图所示：



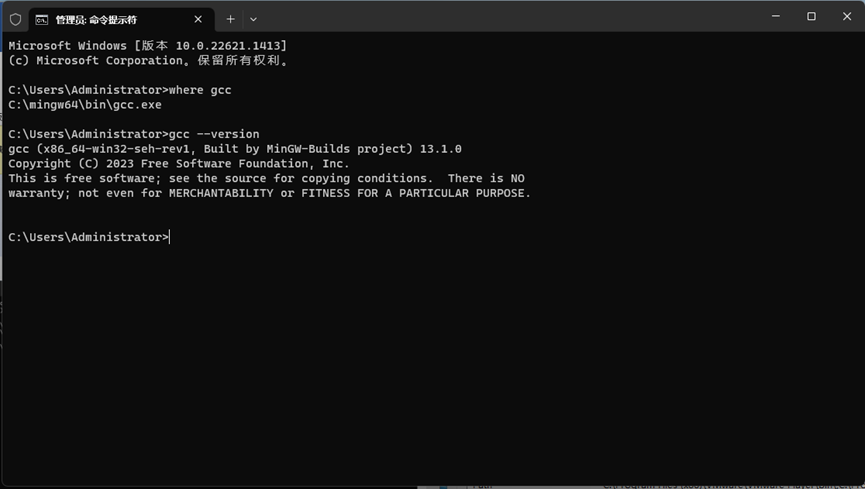
**6.3 确认环境变量生效**

像之前一样打开搜索栏，但这次搜索“cmd”，会有个匹配项叫做“命令提示符”，打开它即可看到在 Windows 系统中输入命令的经典界面。



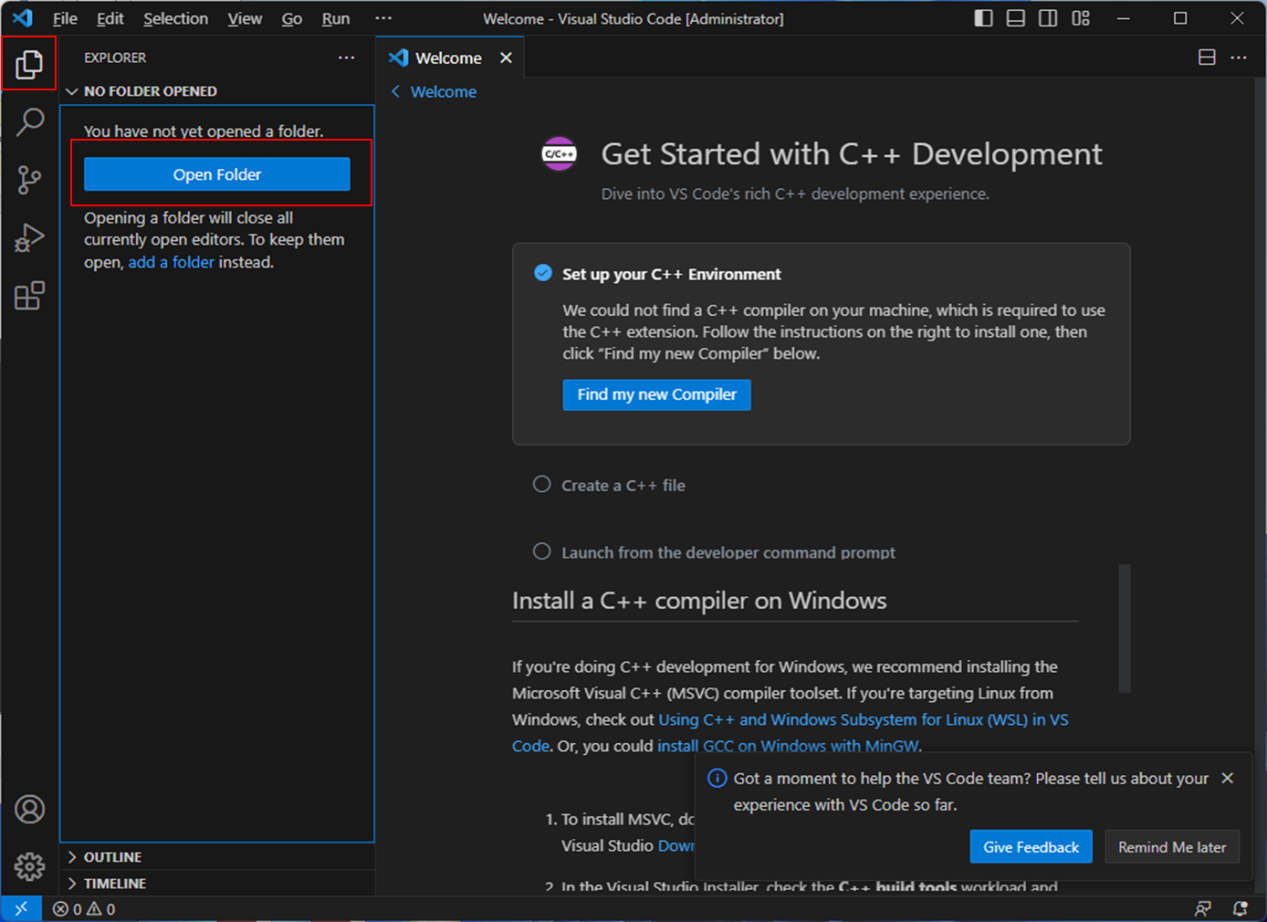
输入where gcc 查看gcc路径

输入gcc --version查看版本



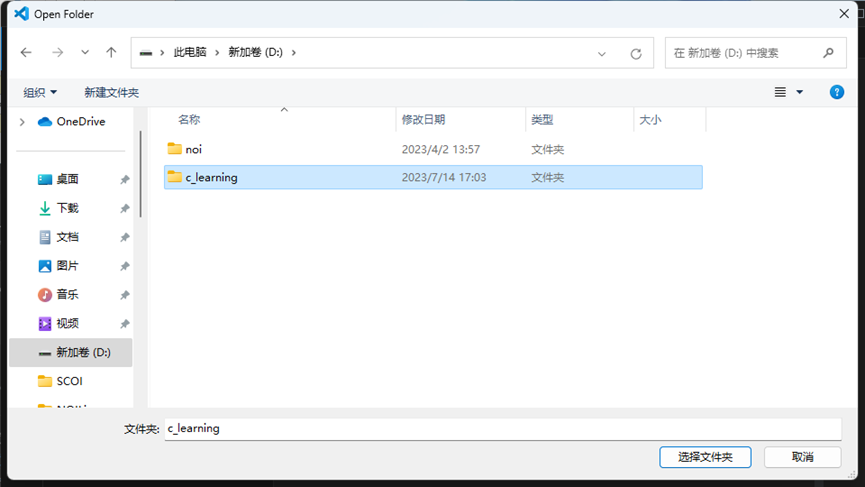
1. **编译运行第一个C程序**
2. **打开文件夹**

点击左侧活动栏第1个图标，打开对应的侧边栏，点击Open Folder

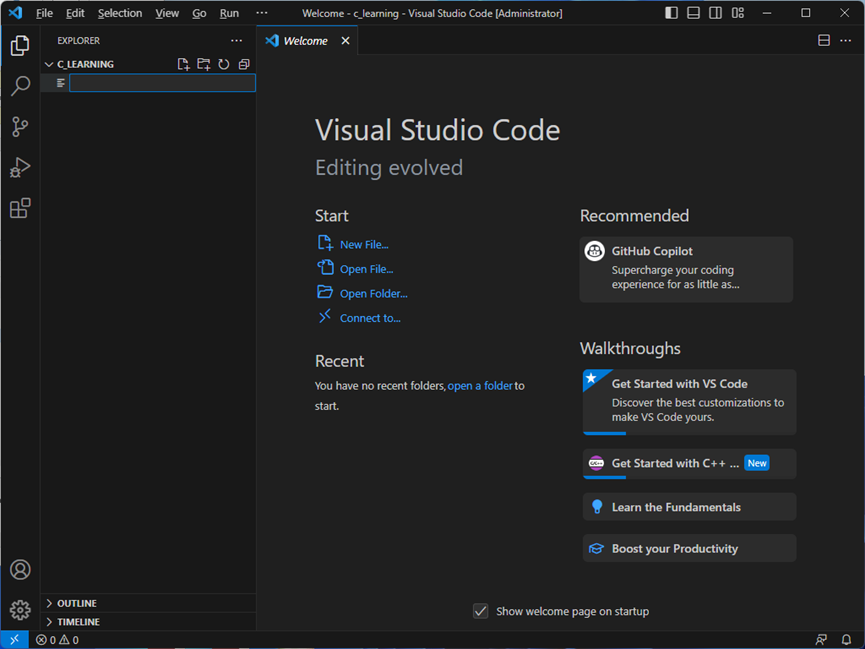


在弹出的导航对话框中，选中一个你系统中方便的文件夹来打开它。注意，这个文件夹的名字和它所在的任何上级文件夹名字中都不能有任何中文或空格！

这里设置的是D:/c\_learning目录

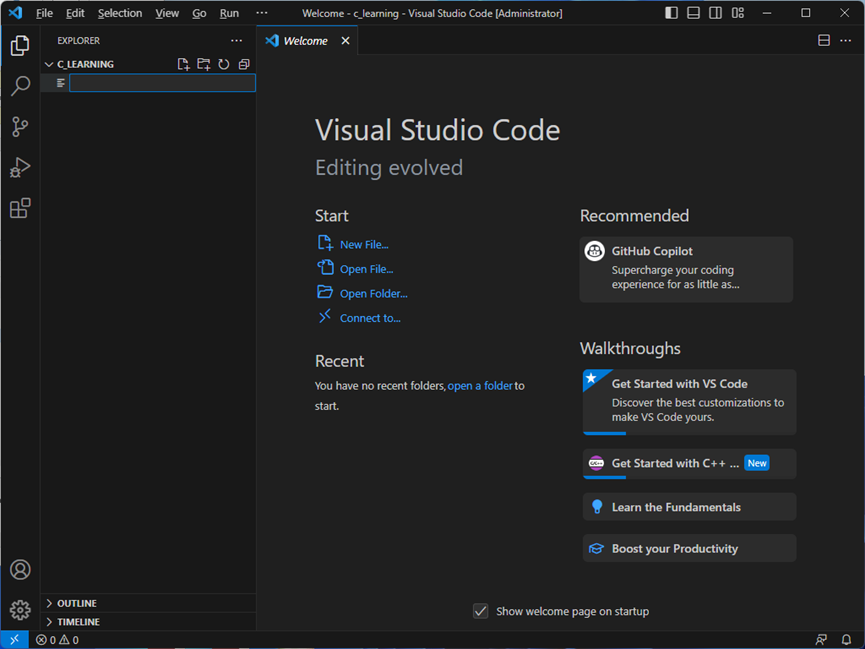


选中之后主界面如下所示，侧边栏中可以看到加载的C\_LEARNING目录（大写）

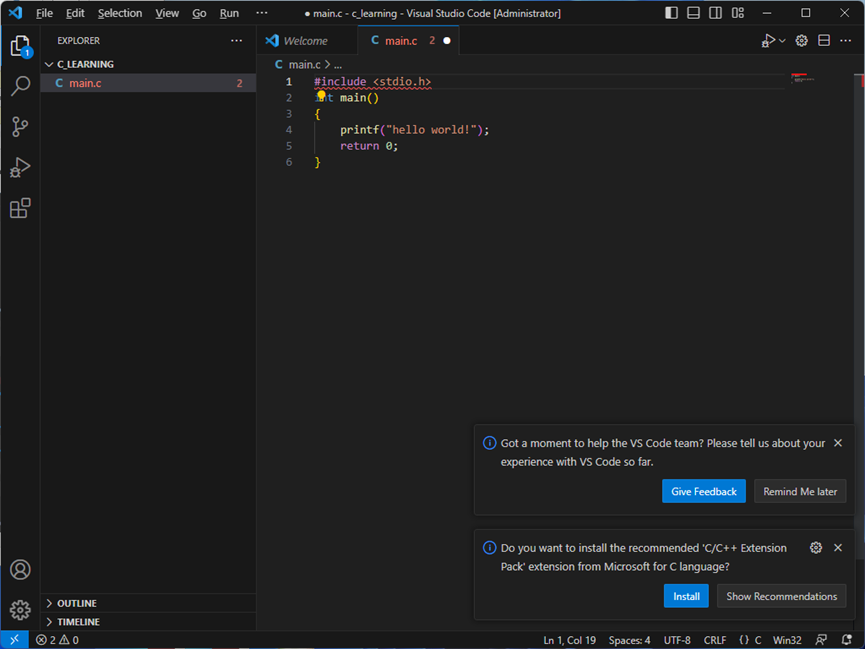


1. **新建c文件**

在目录下双击，输入文件名就可以直接新建文件



这里新建了一个main.c的文件，双击，在右侧主编辑框中输入代码。没有保存前文件标题栏右侧有个白色的小点。

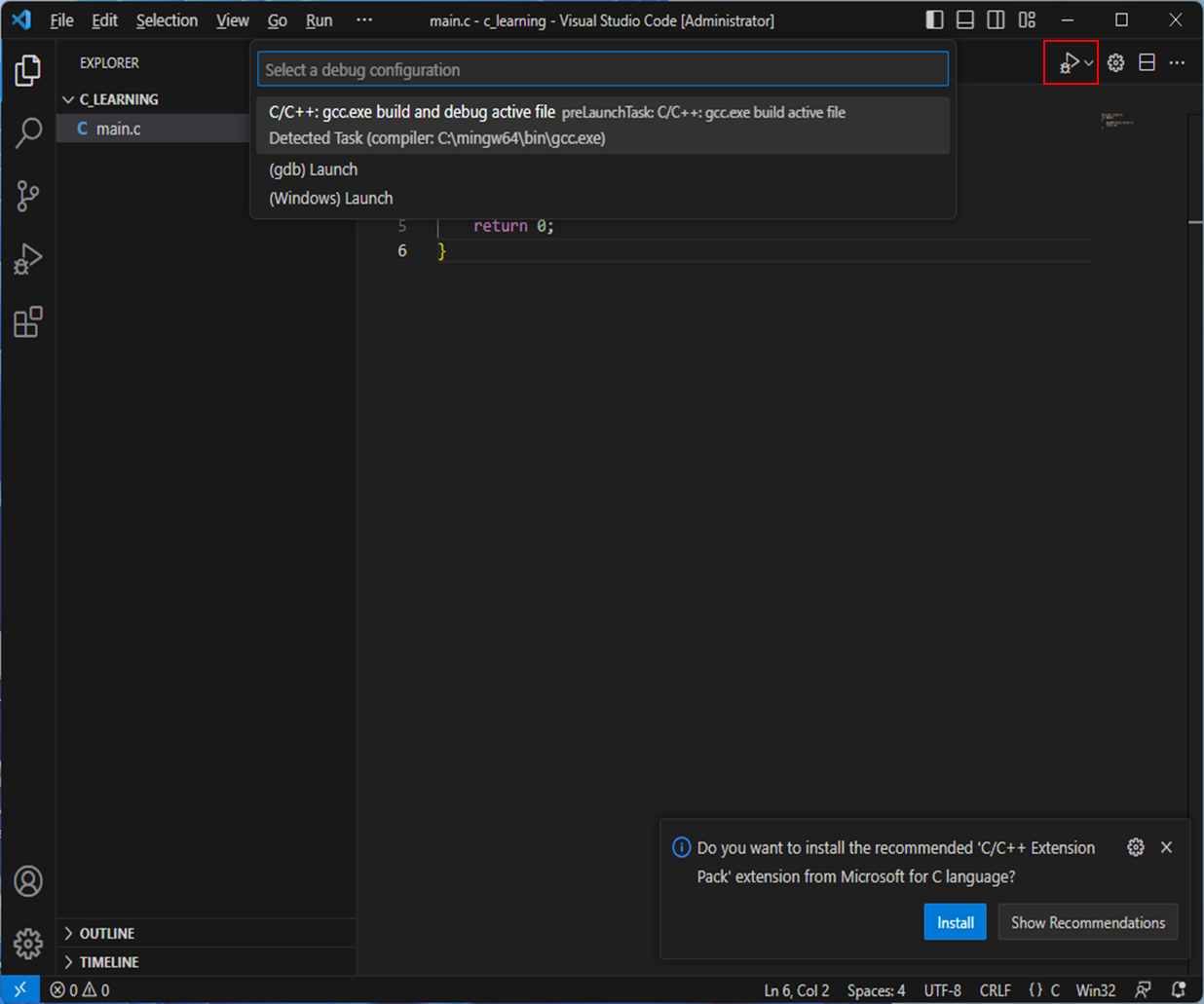


Ctrl+S 保存后白色小点消失

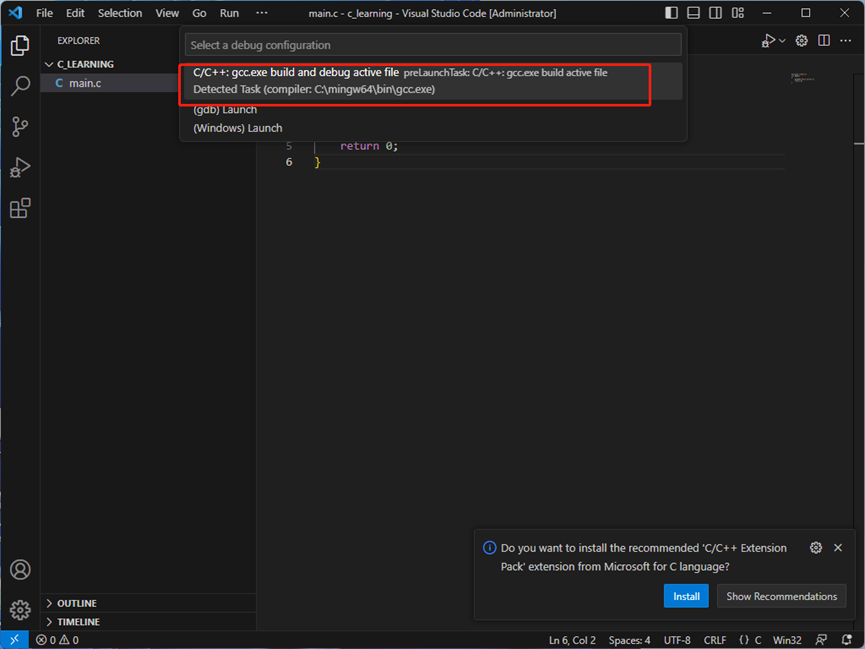


1. **编译运行**

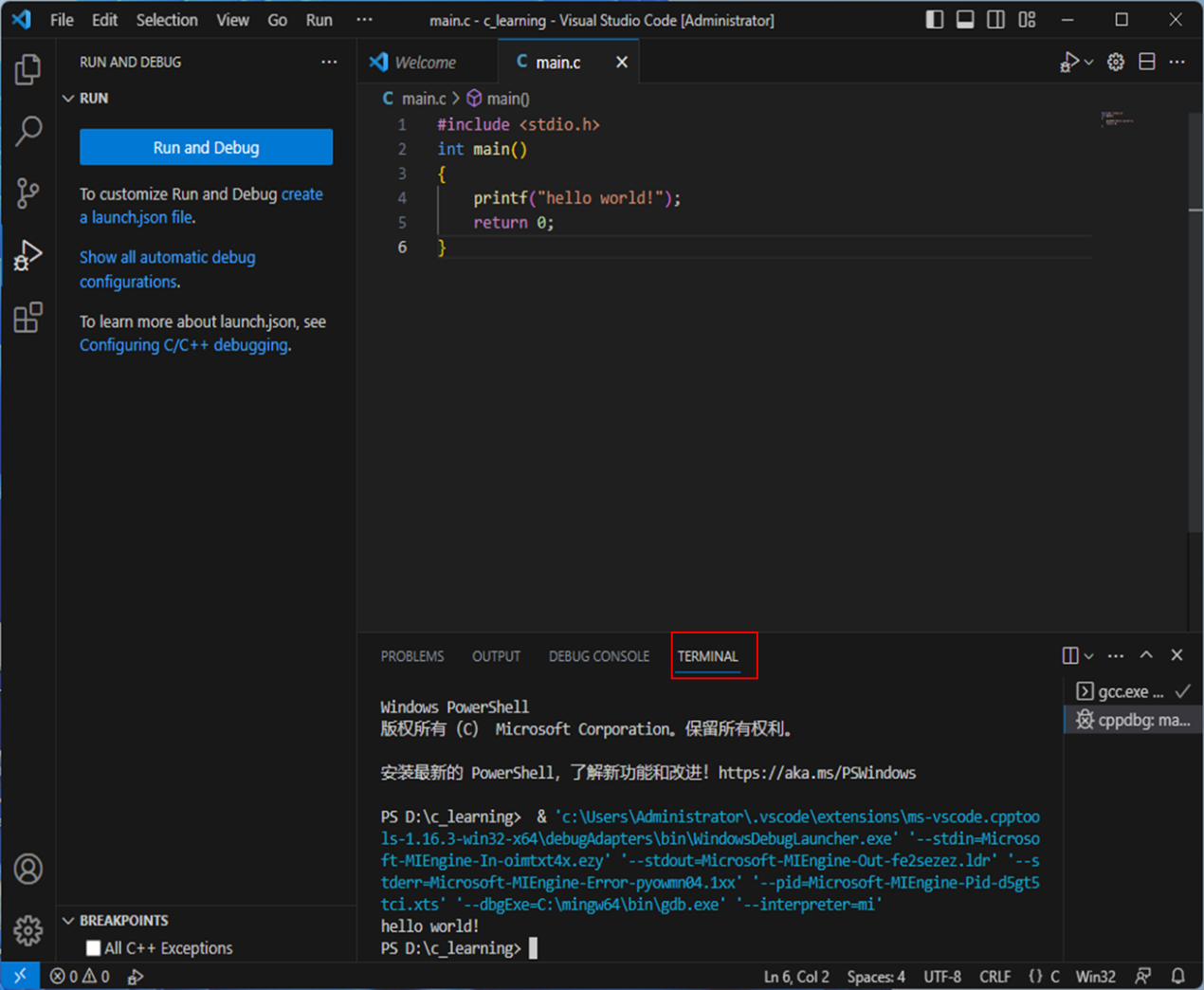
点击右侧三角形图标



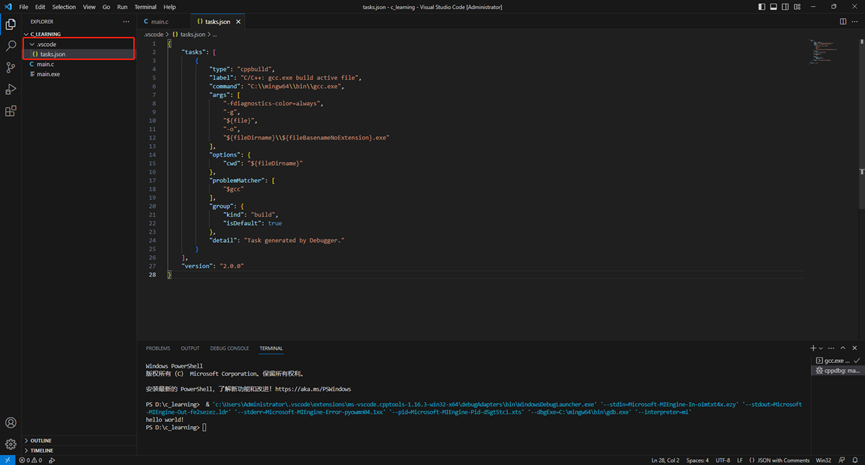
选择C/C++:gcc.exe



第一次运行会自动跳转到以下界面，在下面面板界面，点解TERMINAL，能看到输出结果



点击左侧活动栏第1个图标，打开对应的侧边栏，可以看到C\_LEARNING目录下自动添加了.vscode目录，且该目录下多了一个task.json文件。同时C\_LEARNING目录下还增加了main.c编译后的可执行文件main.exe



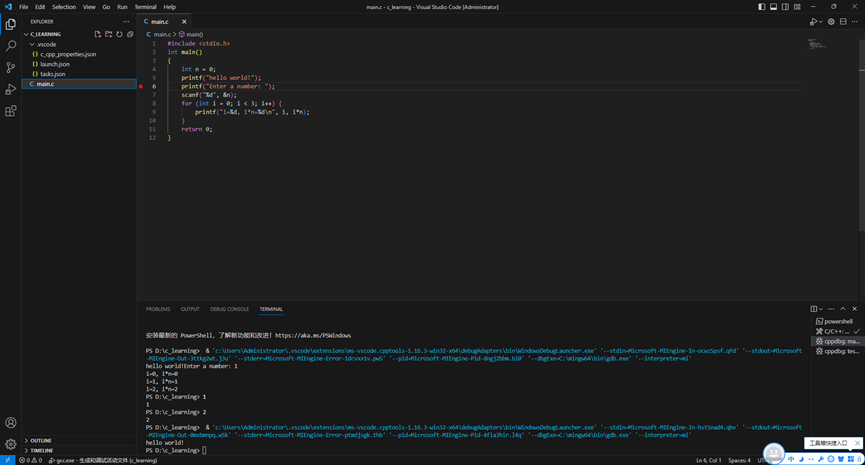
1. **调试配置**

VS Code要实现调试，必须正确的配置.vscode下面的3个文件：

c\_cpp\_properties.json

launch.json

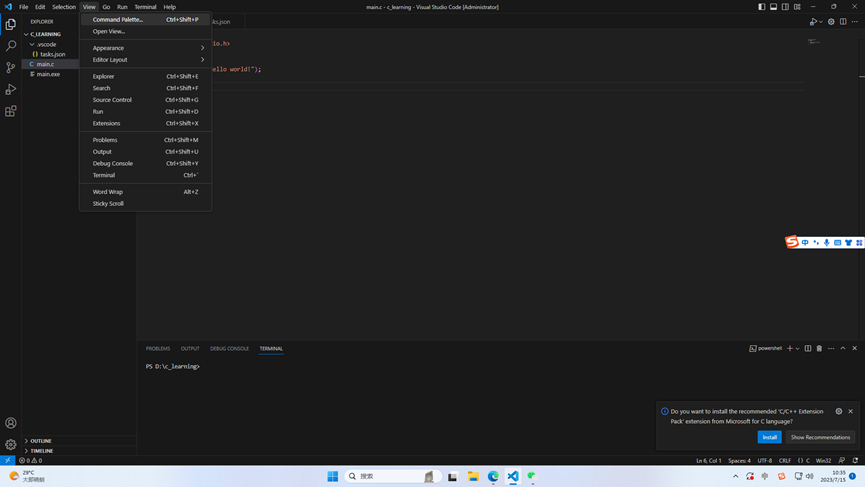
tasks.json



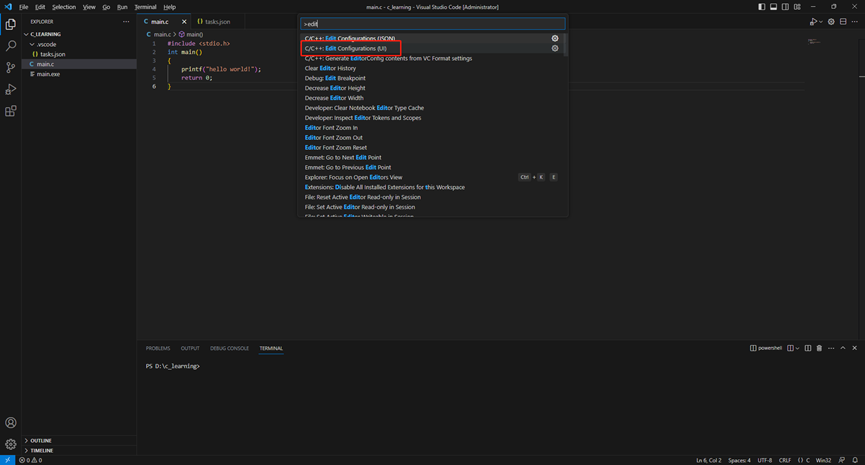
最简单的方式就是直接在.vscode目录下新建这三个文件，然后拷贝对应的配置（前提是前面的配置与本文一致，否则需要修改相应的路径）。

1. **设置配置文件c\_cpp\_properties.json**

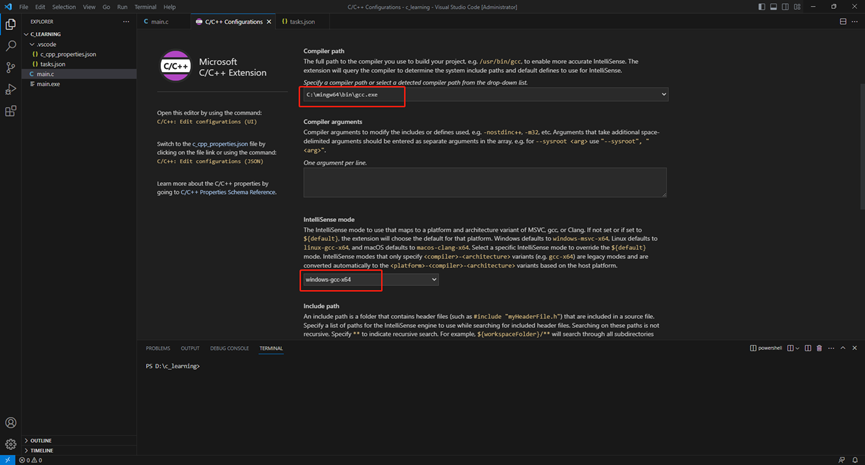
选择菜单栏【View】【Command Palette…】（快捷键【Ctrl+Shift+P】）打开命令面板



输入关键词"C/C++ edit"，选择匹配到的命令"C/C++: Edit Configurations (UI)"



在打开的C/C++基本配置界面中，确保编辑环境配置的正确。



同时，在侧边栏.vscode文件夹下，会存在一个名为c\_cpp\_properties.json的配置文件，里面实际上就是上面UI界面中的内容。

{

"configurations": [

{

"name": "Win32",

"includePath": [

"${workspaceFolder}/\*\*"

],

"defines": [

"\_DEBUG",

"UNICODE",

"\_UNICODE"

],

"compilerPath": "C:\\mingw64\\bin\\gcc.exe",

"cStandard": "c17",

"cppStandard": "gnu++17",

"intelliSenseMode": "windows-gcc-x64"

}

],

"version": 4

}

1. **设置配置文件launch.json**

{

// 使用 IntelliSense 了解相关属性。

// 悬停以查看现有属性的描述。

// 欲了解更多信息，请访问: https://go.microsoft.com/fwlink/?linkid=830387

"version": "0.2.0",

"configurations": [

{

"name": "gcc.exe - 生成和调试活动文件", // 该调试任务的名字，启动调试时会在待选列表中显示

"type": "cppdbg",

"request": "launch",

"program": "${fileDirname}\\${fileBasenameNoExtension}.exe",

"args": [],

"stopAtEntry": false, // 这一项控制是否在入口处暂停，默认false不暂停，改为true暂停

"cwd": "${workspaceFolder}",

"environment": [],

"externalConsole": false, // 这一项控制是否启动外部控制台（独立的黑框）运行程序，默认false表示在集成终端中运行

"MIMode": "gdb",

"miDebuggerPath": "C:\\mingw64\\bin\\gdb.exe", // 调试器路径，必须与你自己的电脑相符

"setupCommands": [

{

"description": "为 gdb 启用整齐打印",

"text": "-enable-pretty-printing",

"ignoreFailures": true

}

],

"preLaunchTask": "C/C++: gcc.exe build active file" // 调试前的预执行任务，这里的值是tasks.json文件中对应的编译任务，也就是调试前需要先编译

}

]

}

1. **设置配置文件tasks.json**

{

"tasks": [

{

"type": "shell",

"label": "C/C++: gcc.exe build active file", // 编译任务的名称，跟上面preLaunchTask的值对应

"command": "C:/mingw64/bin/gcc.exe", // 编译器的路径，须跟你自己的电脑相符

"args": [ // 编译器执行时的参数，跟手动编译时输入的内容基本一致，主要是多了-g参数，以加入调试信息

"-g",

"${file}",

"-o",

"${fileDirname}\\${fileBasenameNoExtension}.exe"

],

"options": {

"cwd": "C:/mingw64/bin"

},

"problemMatcher": [

"$gcc"

],

"group": {

"kind": "build",

"isDefault": true

}

}

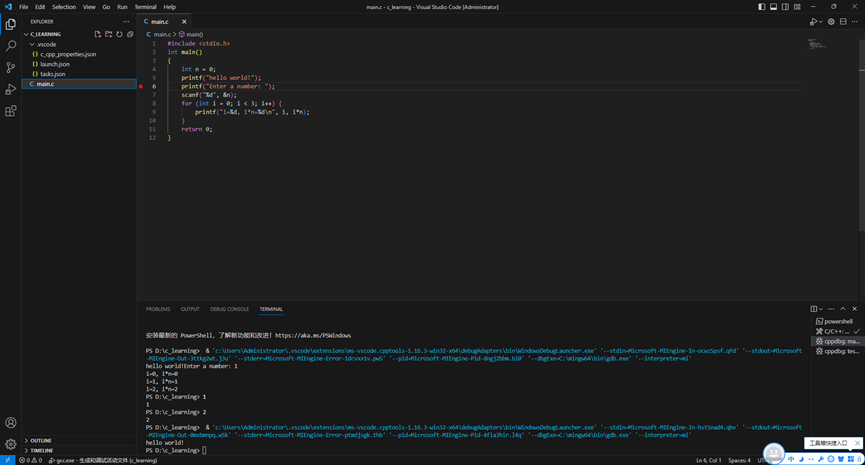
],

"version": "2.0.0"

}

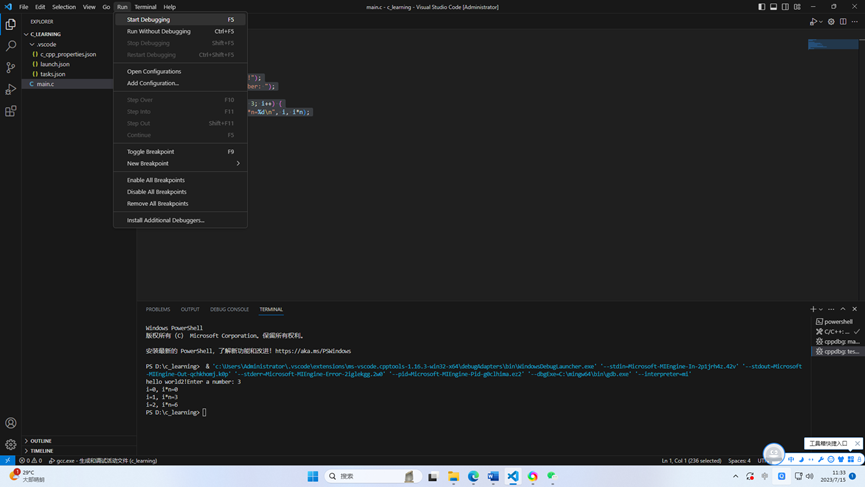
1. **调试C程序**
2. **设置断点**

鼠标单击代码行左边的数字就可以在该行设置断点，设置后会显示一个红色的圆点。

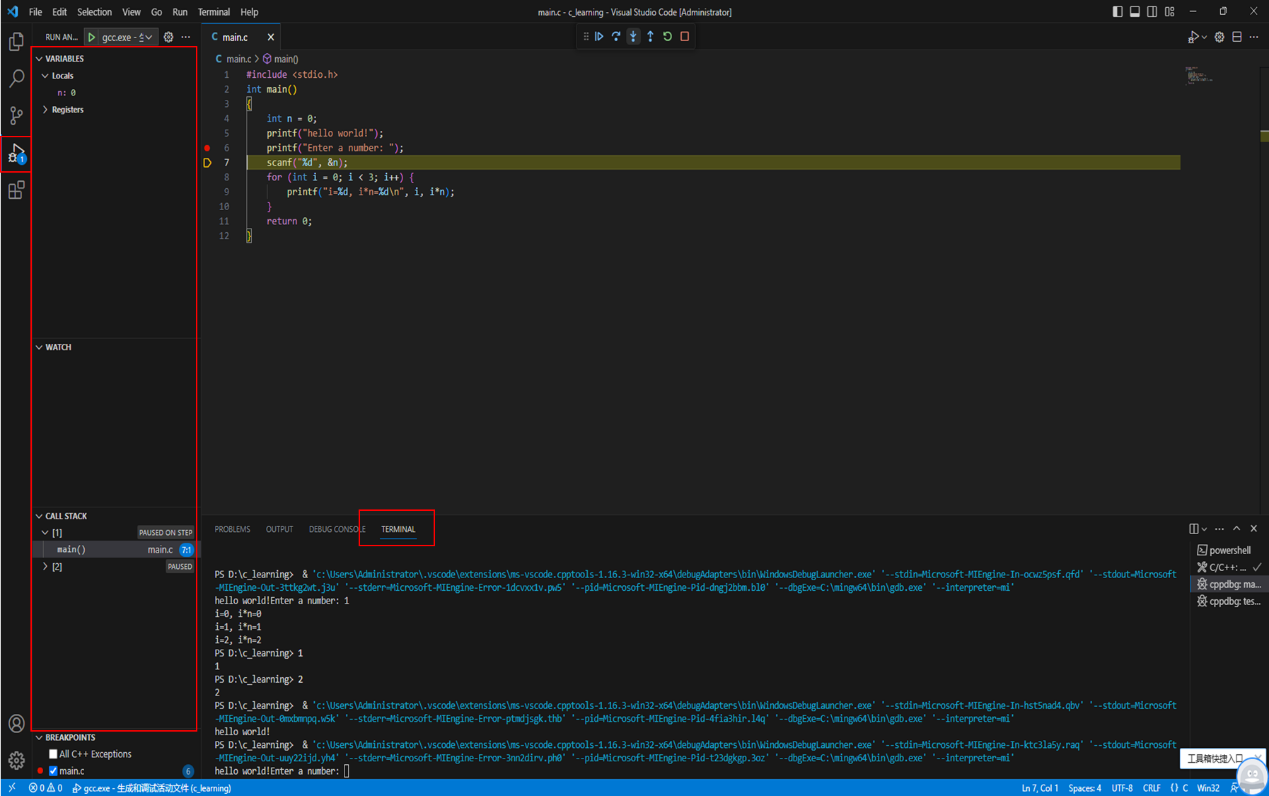


1. **开始调试**

然后选择菜单栏【Run】【Start Debugging】（快捷键【F5】）开始调试



程序经过自动编译并启动调试后，将运行至第一个断点暂停。左侧可以实时观察变量值的变化。



1. **调试工具栏**

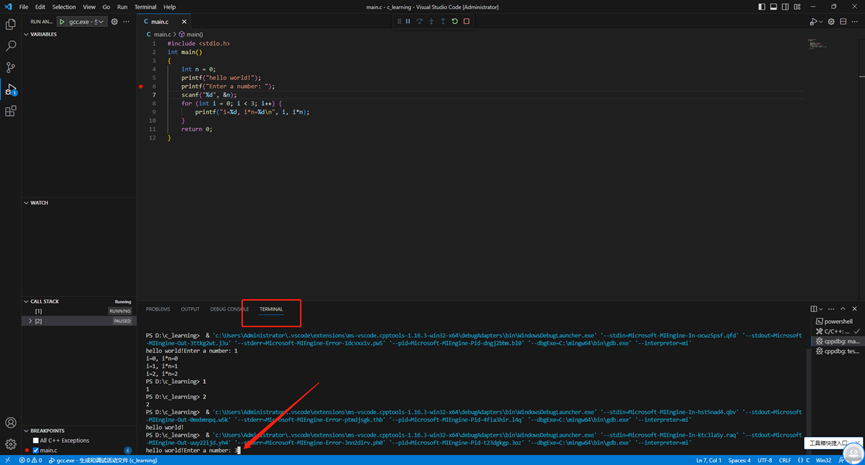
调试会话启动后，调试工具栏将出现在编辑器的顶部。6个按钮如下所示：



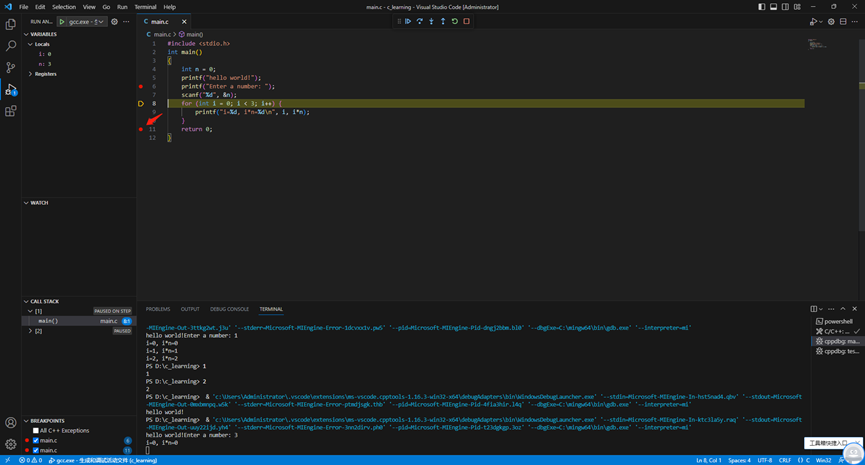
|  |  |
| --- | --- |
| **按钮** | **解释** |
| 继续/暂停  F5 | 继续：恢复正常的程序/脚本执行（直到下一个断点）。  暂停：检查当前行执行的代码并逐行调试。 |
| 单步跳过  F10 | 将下一个方法作为单个命令执行，而不检查或遵循其组成步骤。 |
| 单步进入  F11 | 输入下一个方法来逐行跟踪其执行。 |
| 单步跳出  Shift+F11 | 在方法或子例程内部时，通过完成当前方法的剩余行返回到较早的执行上下文，就像它是单个命令一样。 |
| 重新启动  Ctrl+Shift+F5 | 终止当前程序执行并使用当前运行配置再次开始调试。 |
| 停止  Shift+F5 | 终止当前程序的执行。 |

1. **调试示例**

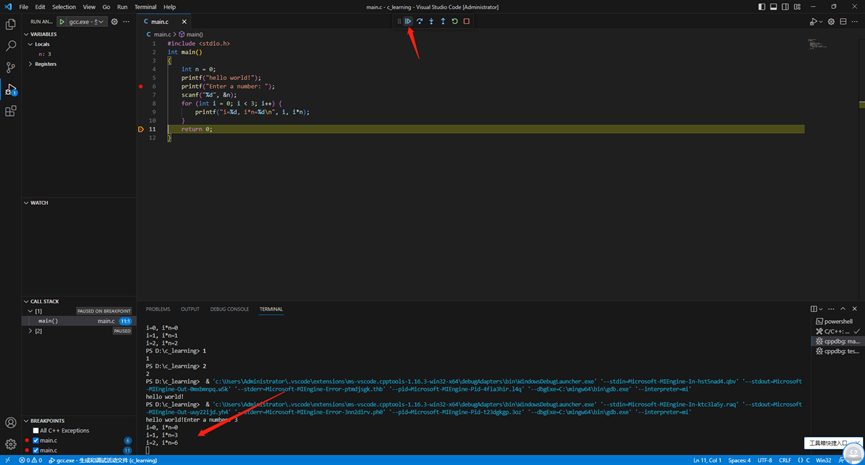
点击单步进入，逐行调试，遇到输入时，需要在TERMINAL中键入用户输入



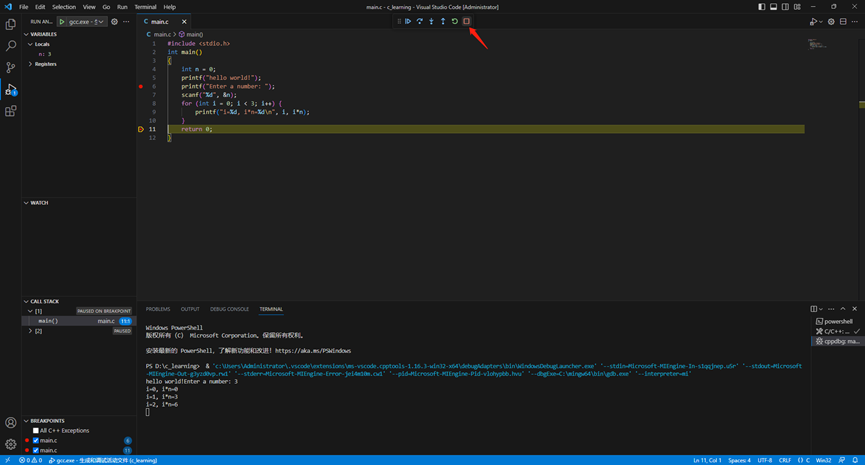
调试过程中也可以增加断点



点击继续跳到下一个断点。

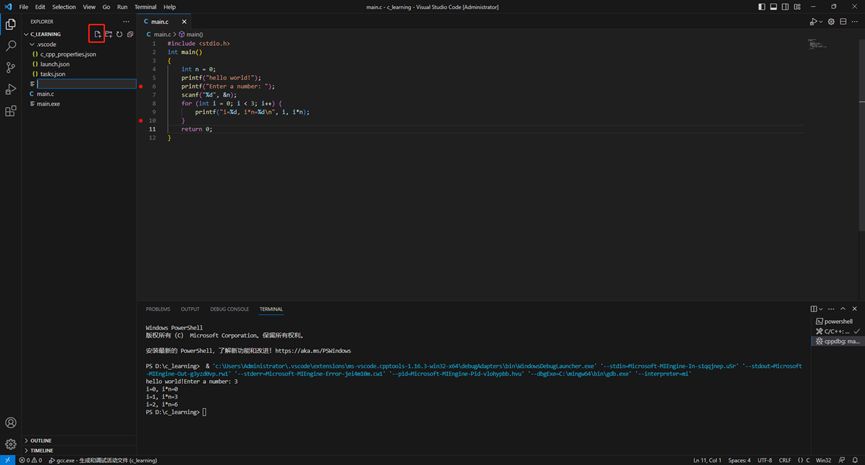


点击停止，结束调试

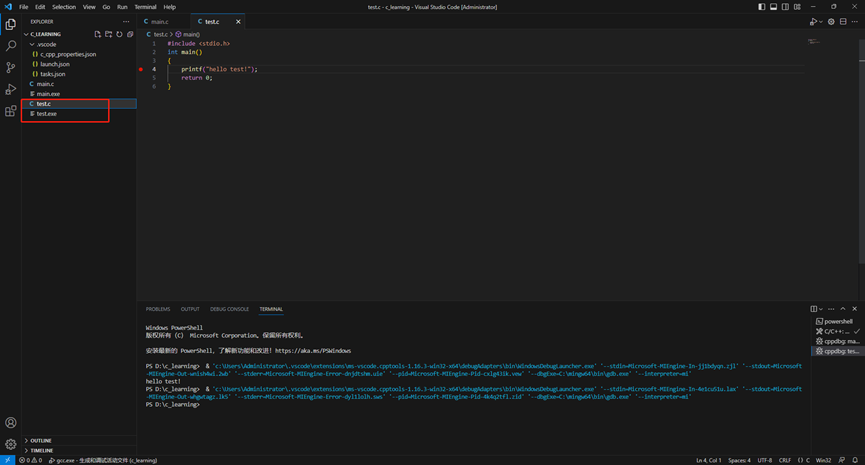


1. **新建其他源文件**

在c\_learning目录下可以创建多个文件（独立的源文件），如下所示新建了一个test.c的文件



F5运行后会自动生成test.exe的执行程序



PS：在创建大型工程时，可能涉及到多个文件相互编译链接（这些文件相互关联，但只有一个main文件），如果要编译运行这种大型工程，需要修改launch.json和tasks.json。