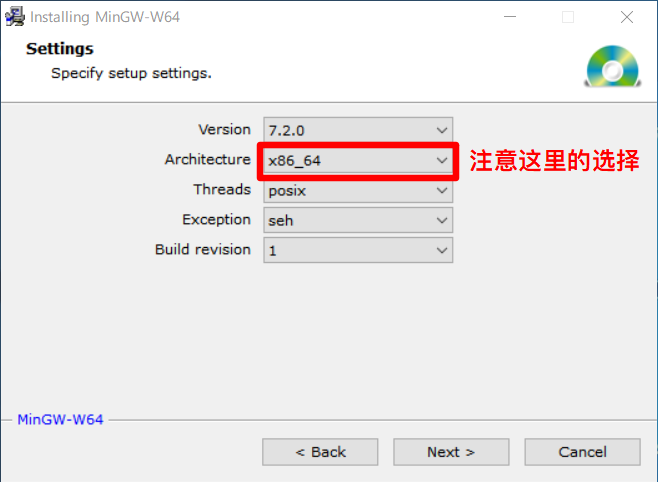
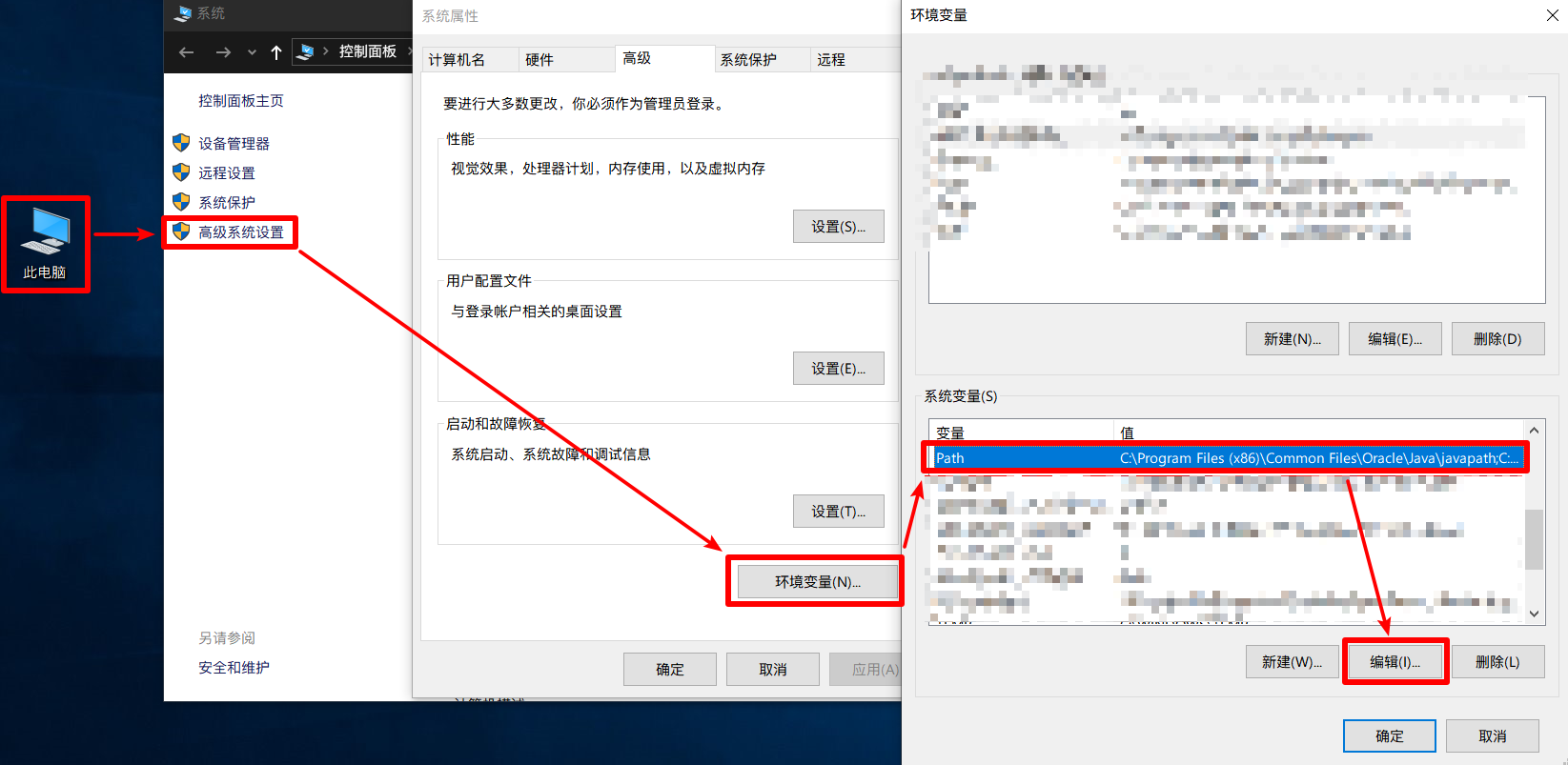
**安装MINGW-64**

[https://sourceforge.net/projects/mingw-w64/files/mingw-w64/](https://sourceforge.net/projects/mingw-w64/files/mingw-w64/" \t "_blank)

打开就是 Next >，默认安装路径即可（注意安装路径不要有中文）



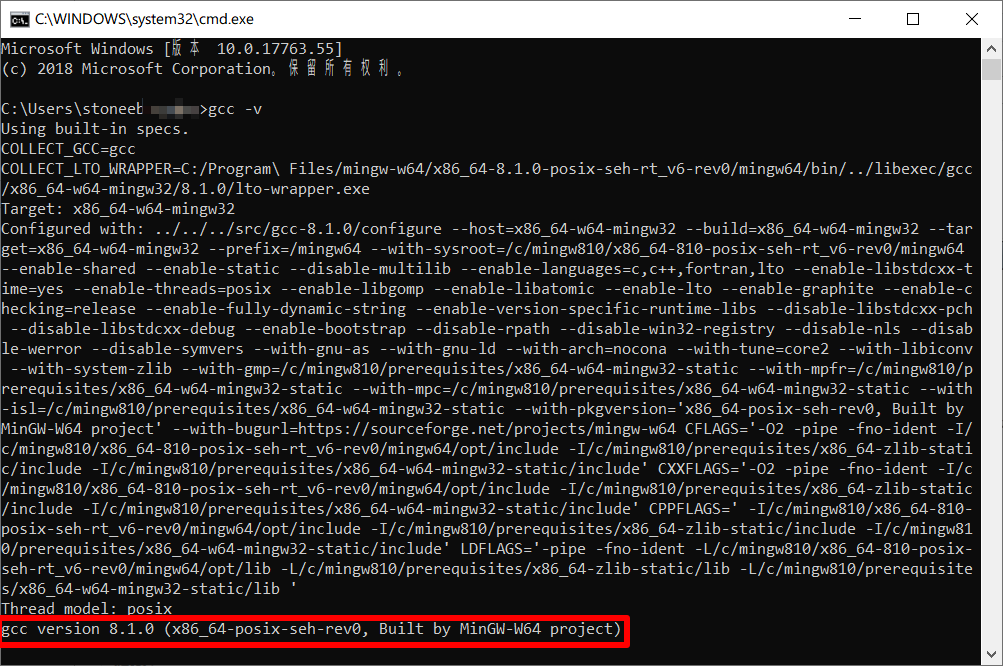
**配置环境变量**



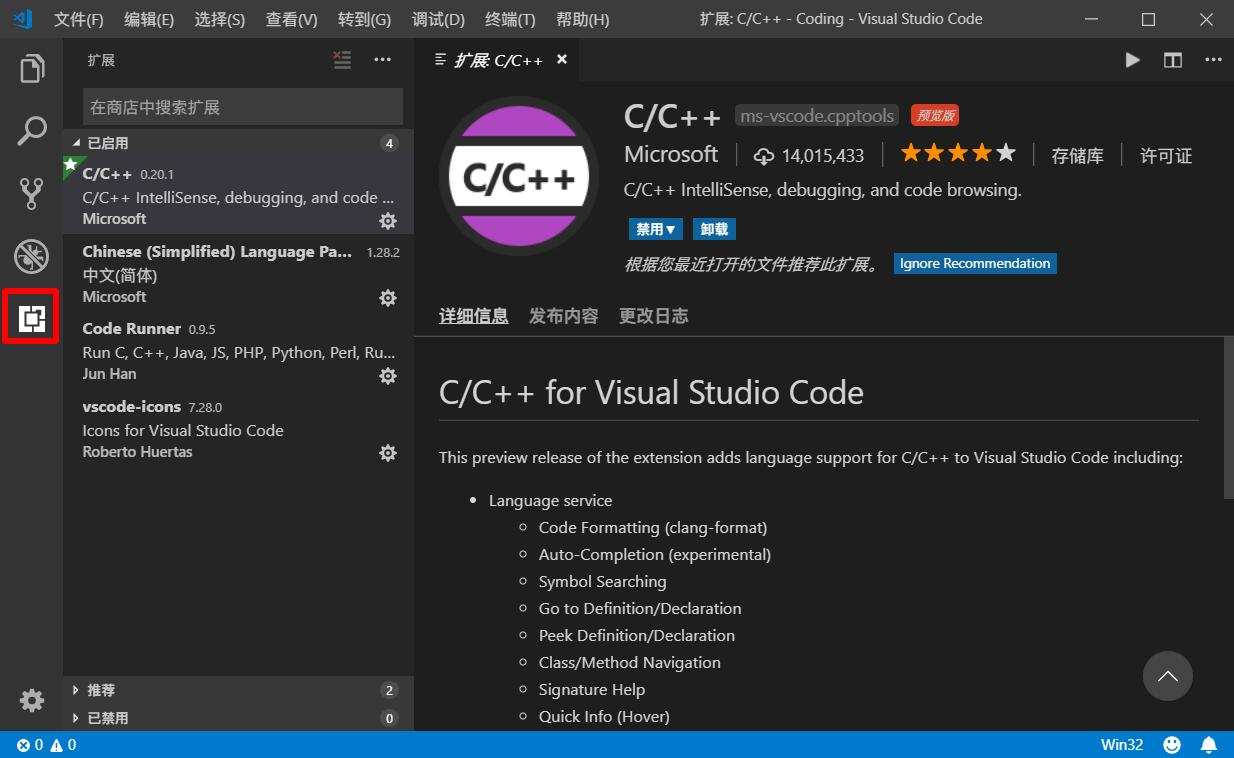
在编辑里添加mingw-64的bin目录

**测试mingw-64安装是否成功**

　按下[Win+R]，然后输入cmd，点击运行，输入双引号里面的内容“gcc -v”



**VSCode安装插件**



C/C++、Code Runner

## 编写你的第一个 C 语言程序

在工作区新建一个 C 语言源文件命名为 hello.cpp ，输入以下内容：

#include <stdio.h>

#include <windows.h>

#include <iostream>

#include <stdlib.h>

int main()

{

std::cout << "hello world!" << std::endl;

int a;

std::cin >> a;

return 0;

}

## **配置智能感知，配置导入的头文件参数 c\_cpp\_properties.json**

在编写完毕并保存之后，你可能会看到 #include 这句下面会有绿色波浪线，这是由于编译器没办法找到你所使用的头文件的所在位置。通过快捷方式Ctrl+Shift+P运行C/CPP: Edit configuration ...命令添加缺少的信息并生成c\_cpp\_properties.json文件。将文件修改成下面内容：

// 下面 C:/Program Files/mingw64 为 MinGw-w64 安装路径

{

    "configurations": [

        {

            "name": "MinGW",

            "intelliSenseMode": "gcc-x64",

            "compilerPath": "C:/Program Files/mingw64/bin/x86\_64-w64-mingw32-g++.exe",

            "cStandard": "c11",

            "cppStandard": "c++17"

        }

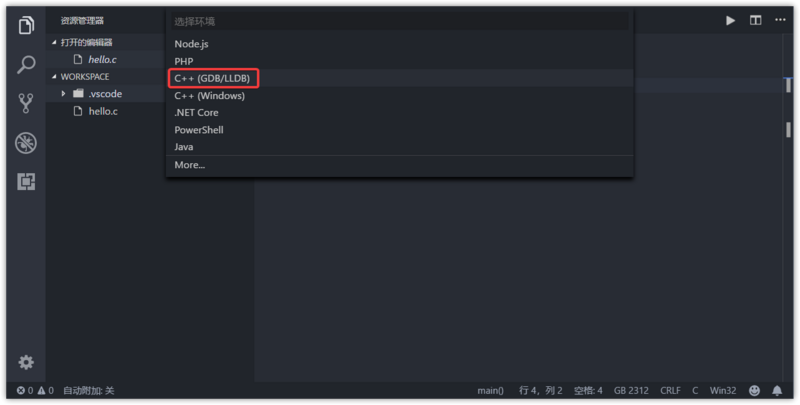
    ],

    "version": 4

}

## 配置调试程序 launch.json

打开已经编写好的 hello.c ，然后按 F5 调试。因为是第一次调试，系统会弹出 选择环境 面板，这里选择 C++(GDB/LLDB)。



选择运行环境后，VS Code 会在工作区 .vscode 文件夹下创建 launch.json 模板文件并打开，将文件内容清空，复制下面的内容到文件中并保存：

{

"version": "0.2.0",

"configurations": [{

// 配置 VS Code 调试行为：

"name": "GDB Debug", // 设置在启动配置下拉菜单中显示调试配置的名称。

"preLaunchTask": "Compile", // 调试会话开始前要运行的任务。

"type": "cppdbg", // 设置要使用的基础调试器。使用 GDB 或 LLDB 时必须是 cppdbg 。

"request": "launch", // 设置启动程序还是附加到已经运行的实例。启动或附加 ( launch | attach ).

"program": "${fileDirname}/${fileBasenameNoExtension}.exe", // 调试器将启动或附加的可执行文件的完整路径。

"externalConsole": true, // 设置是否显示外部控制台。

"logging": { // 用于确定应该将哪些类型的消息记录到调试控制台。

"exceptions": true, // 是否应将异常消息记录到调试控制台。默认为真。

"moduleLoad": false, // 是否应将模块加载事件记录到调试控制台。默认为真。

"programOutput": true, // 是否应将程序输出记录到调试控制台的可选标志。默认为真。

"engineLogging": false, // 是否应将诊断引擎日志记录到调试控制台。默认为假。

"trace": false, // 是否将诊断适配器命令跟踪记录到调试控制台。默认为假。

"traceResponse": false // 是否将诊断适配器命令和响应跟踪记录到调试控制台。默认为假。

},

// 配置目标应用程序：

"args": [], // 设置调试时传递给程序的命令行参数。

"cwd": "${workspaceFolder}", // 设置调试器启动的应用程序的工作目录。

"environment": [], // 设置调试时添加到程序环境中的环境变量，例如: [ { "name": "squid", "value": "clam" } ]。

// 自定义 GDB 或者 LLDB：

"windows": {

"MIMode": "gdb", // 指定 VS Code 连接的调试器，必须为 gdb 或者 lldb。

"miDebuggerPath": "C:/Program Files/mingw64/bin/gdb.exe" // 调试器的路径，修改为你的安装路径

},

"miDebuggerArgs": "", // 传递给调试器的附加参数

"stopAtEntry": false, // 设置调试器是否停止在目标的入口（附加时忽略）。默认值为 false。

"setupCommands": [{ // 执行下面的命令数组以设置 GDB 或 LLDB

"description": "Enable pretty-printing for gdb",

"text": "-enable-pretty-printing", // 鼠标悬停查看变量的值，需要启用 pretty-printing 。

"ignoreFailures": true // 忽略失败的命令，默认为 false 。

}]

}]

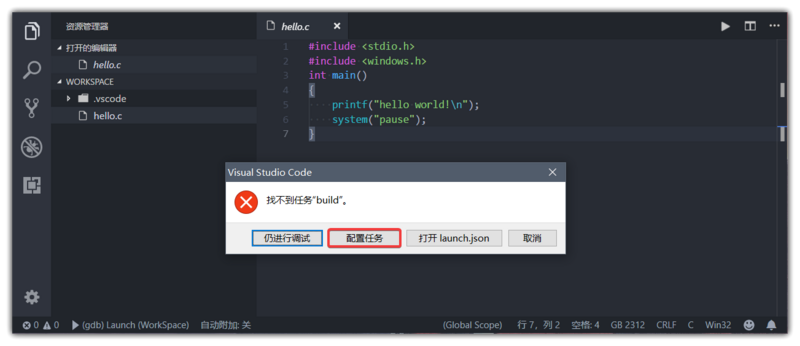
}

**留意** luanch.json **中的注释内容**，记得把 "miDebuggerPath" 参数修改成你自己安装位置里的 gdb.exe

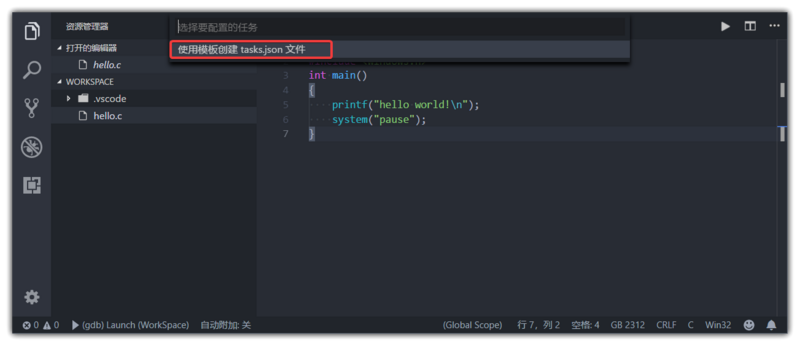
gdb.exe 位于 {MinGW-w64安装位置}\mingw64\bin 下面。

## 配置调试前执行的任务 task.json

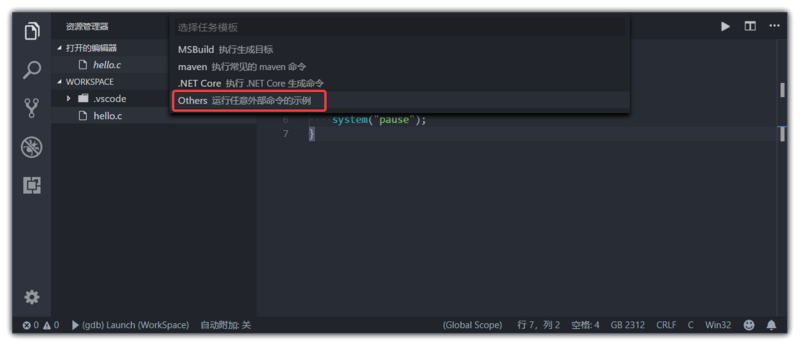
再按一次 F5 ，会弹出“找不到任务”的提示窗口，点击 配置任务 按钮，如下图所示：



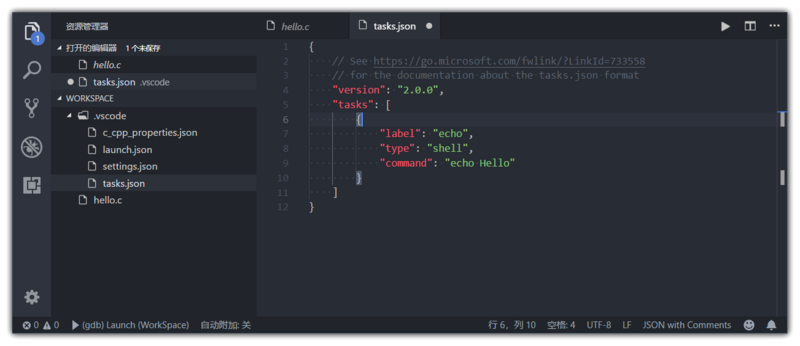
然后在弹出的命令面板选择 使用模板创建 task.json 文件 ，如下图所示：



继续选择 Others 运行任意外部命令的示例 ，如下图所示：



完成以上步骤之后，会在工作区的 .vscode 目录下生成 tasks.json 文件，并自动打开 task.json 文件。



接下来我们将 task.json 文件内容清空，复制下面的内容到文件中并保存：

{

// 有关 tasks.json 格式的参考文档：https://go.microsoft.com/fwlink/?LinkId=733558 。

"version": "2.0.0",

"tasks": [{

"label": "Compile",

"type": "shell", // { shell | process }

// 适用于 Windows 的配置：

"windows": {

"command": "g++",

"args": [

"-g",

"\"${file}\"",

"-o",

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

// 设置编译后的可执行文件的字符集为 GB2312：

// "-fexec-charset", "GB2312"

// 直接设置命令行字符集为 utf-8：

// chcp 65001

]

},

// 定义此任务属于的执行组：

"group": {

"kind": "build", // { build | test }

"isDefault": true // { true | false }

},

// 定义如何在用户界面中处理任务输出：

"presentation": {

// 控制是否显示运行此任务的面板。默认值为 "always"：

// - always: 总是在此任务执行时显示终端。

// - never: 不要在此任务执行时显示终端。

// - silent: 仅在任务没有关联问题匹配程序且在执行时发生错误时显示终端

"reveal": "silent",

// 控制面板是否获取焦点。默认值为 "false"：

"focus": false,

// 控制是否将执行的命令显示到面板中。默认值为“true”：

"echo": false,

// 控制是否在任务间共享面板。同一个任务使用相同面板还是每次运行时新创建一个面板：

// - shared: 终端被共享，其他任务运行的输出被添加到同一个终端。

// - dedicated: 执行同一个任务，则使用同一个终端，执行不同任务，则使用不同终端。

// - new: 任务的每次执行都使用一个新的终端。

"panel": "dedicated"

},

// 使用问题匹配器处理任务输出：

"problemMatcher": {

// 代码内问题的所有者为 cpp 语言服务。

"owner": "cpp",

// 定义应如何解释问题面板中报告的文件名

"fileLocation": [

"relative",

"${workspaceFolder}"

],

// 在输出中匹配问题的实际模式。

"pattern": {

// The regular expression.

"regexp": "^(.\*):(\\d+):(\\d+):\\s+(warning|error):\\s+(.\*)$",

// 第一个匹配组匹配文件的相对文件名：

"file": 1,

// 第二个匹配组匹配问题出现的行：

"line": 2,

// 第三个匹配组匹配问题出现的列：

"column": 3,

// 第四个匹配组匹配问题的严重性，如果忽略，所有问题都被捕获为错误：

"severity": 4,

// 第五个匹配组匹配消息：

"message": 5

}

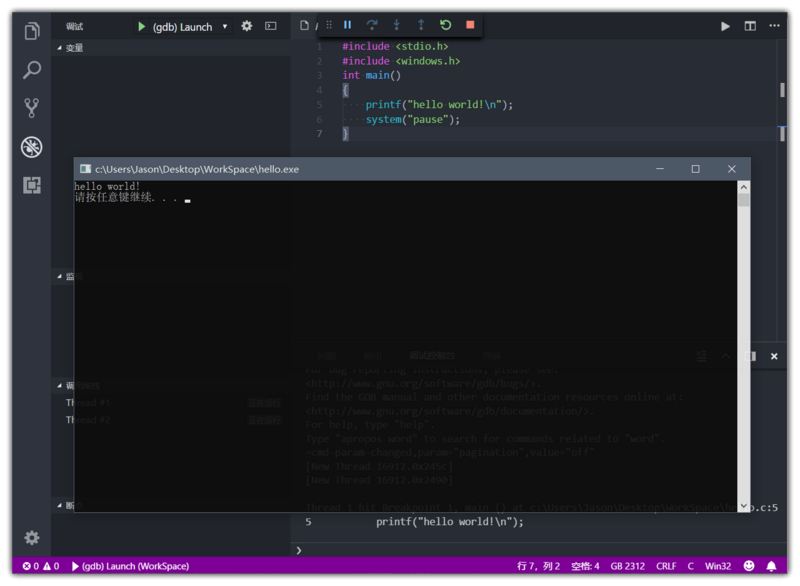
}

}]

}

## 完成

到这里，开发环境就已经配置完毕。接下来我们在 hello.cpp 的编辑窗口按 F5 运行下，看下效果。



如果你还想再创建其他的 C 语言开发工作区，我们只需要新建一个文件夹，再把现在已有工作区目录下的 .vscode 文件夹复制到新建的文件夹即可。