0、C++开发环境相关理论

0、代码编辑器：Visual Studio Code

编辑器就是处理文本（源码）的程序，写代码写的就是文本，编辑器可能提供智能提示、代码高亮等辅助功能，但不负责源码到二进制文件的操作;

1、编译器下载：MinGW-w64

编译器就是负责将源码文本翻译成计算机能够理解和执行的二进制文件的程序;

2、VsCode&C++调试器：安装C\C++扩展

3、配置环境变量 = 编译器+ 调试器

4、程序从编辑到可执行的过程:

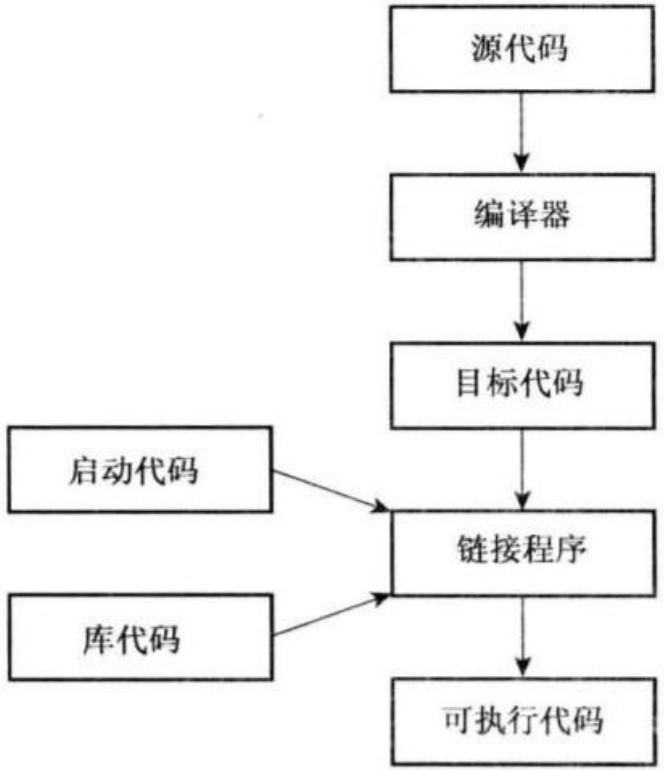
首先用文本编辑器编写源代码 -> 编译源代码 生成目标代码-> 将目标代码与其它代码（如库函数代码、标准启动代码）链接起来 -> 生成可执行代码

5、集成开发环境（IDE，Integrated Development Environment ）是用于提供程序开发环境的应用程序，包括了代码编辑器、编译器、调试器和图形用户界面工具。集成了代码编写、分析、编译、调试等一整套工具链。

6、搭建环境:

vscode定位代码编辑器，不是IDE，不包含编译功能，因此需要我们自己安装编译器、调试器等编译器套件，并使两者有效的配合起来，以实现快捷操作。把这一整套工具链整合到一起的过程就是我们所说的搭建环境。

7、搭建步骤：获取编辑器 -> 获取编译套装(编译器、调试器、头文件库等) -> 做好两者之间的沟通工作(配置文件)



1、下载安装

0、代码编辑器：Visual Studio Code → System Installer

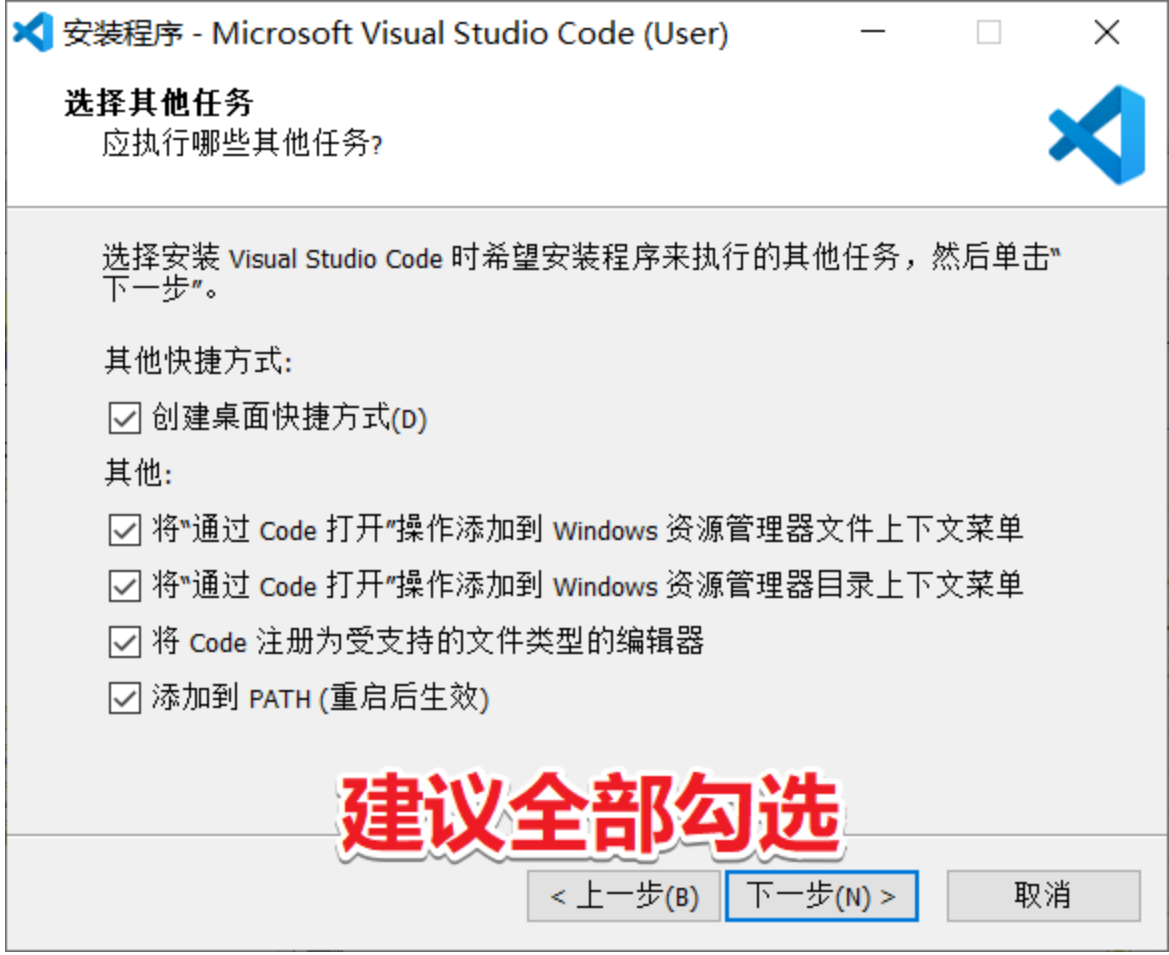
https://visualstudio.microsoft.com/zh-hans/
https://code.visualstudio.com/

1、完全卸载

C:\Users\$用户名\.vscode
C:\Users\$用户名\AppData\Roaming\Code

2、编译器安装

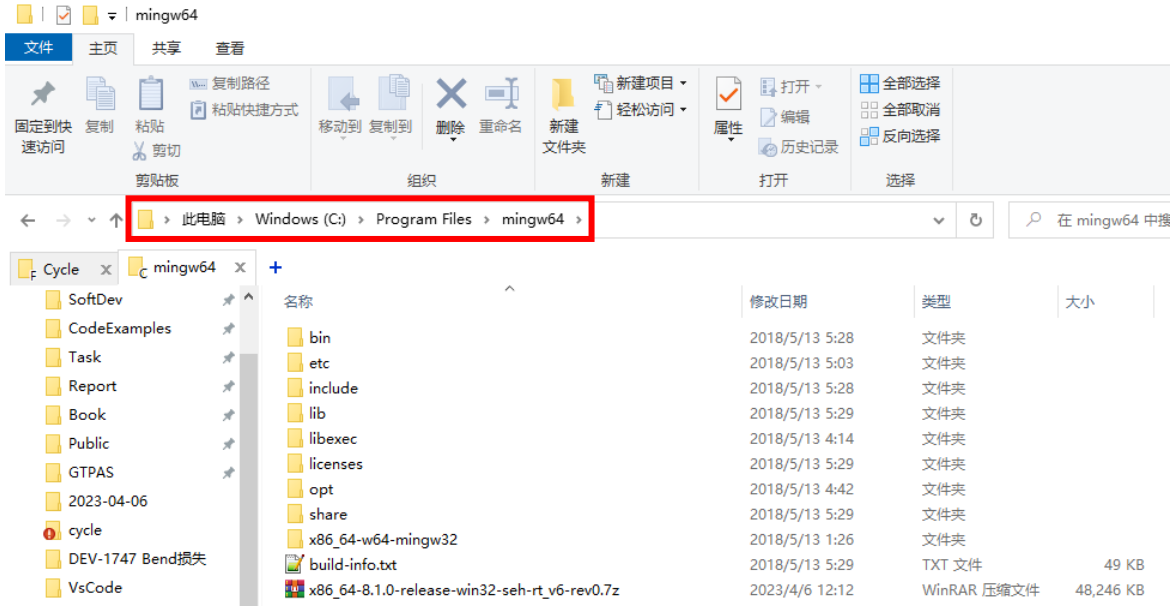
https://sourceforge.net/projects/mingw-w64/files/mingw-w64/mingw-w64-release/



gcc->MinGW-w64->C:\Program Files\mingw64\bin->g++.exe(c++编译器)/gdb.exe(调试器)
编译工具选用**gcc**（全称GNU Compiler Collection GNU编译器套件），不过不是原版的gcc，而是它在Windows下的特制版**MinGW-w64**(全称Minimalist GNU on Windows）。
它实际上是将GCC 移植到了 Windows 平台下，并且包含了 Win32API，因此可以将源代码编译为可在 Windows 中运行的可执行程序。
还有一些头文件也里面，如stdio.h的位置是C:\Program Files\mingw64\x86_64-w64-mingw32\include

1、下载安装

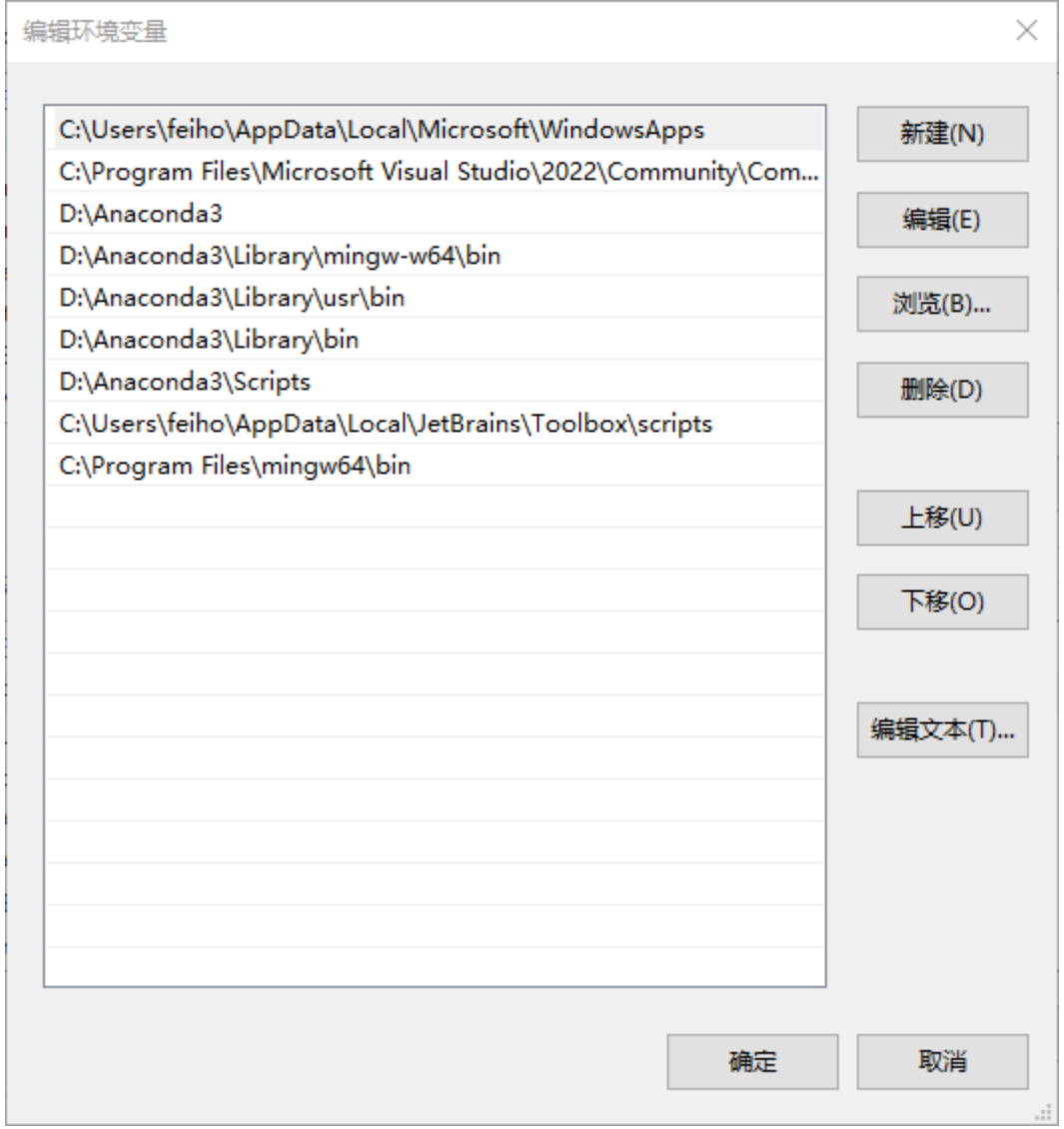
2.1 编译器安装位置



2.2 添加环境变量

为了让程序能访问到这些编译程序，需要把它们所在的目录**（C:\Program File\mingw64\bin）添加到环境变量Path**中。

环境变量是 Windows 系统中用来指定运行环境的一些参数，它包含了关于系统及当前登录用户的环境信息字符串。当用户运行某些程序时，系统除了会在当前文件夹中寻找某些文件外，还会到环境参数的默认路径中去查找程序运行时所需要的系统文件。



2.3 CMD验证

cmd命令提示符 → gcc --version(中间有空格)

1、下载安装

2.3 C/C++ 扩展

【C/C++】： 这个肯定是编译C/C++程序必不可少的

【C/C++ Snippets】： 就是C/C++的重用代码块

【C/C++ Advanced Lint】： C/C++静态检测

【Code Runner】： 这个也是必不可少的，代码运行

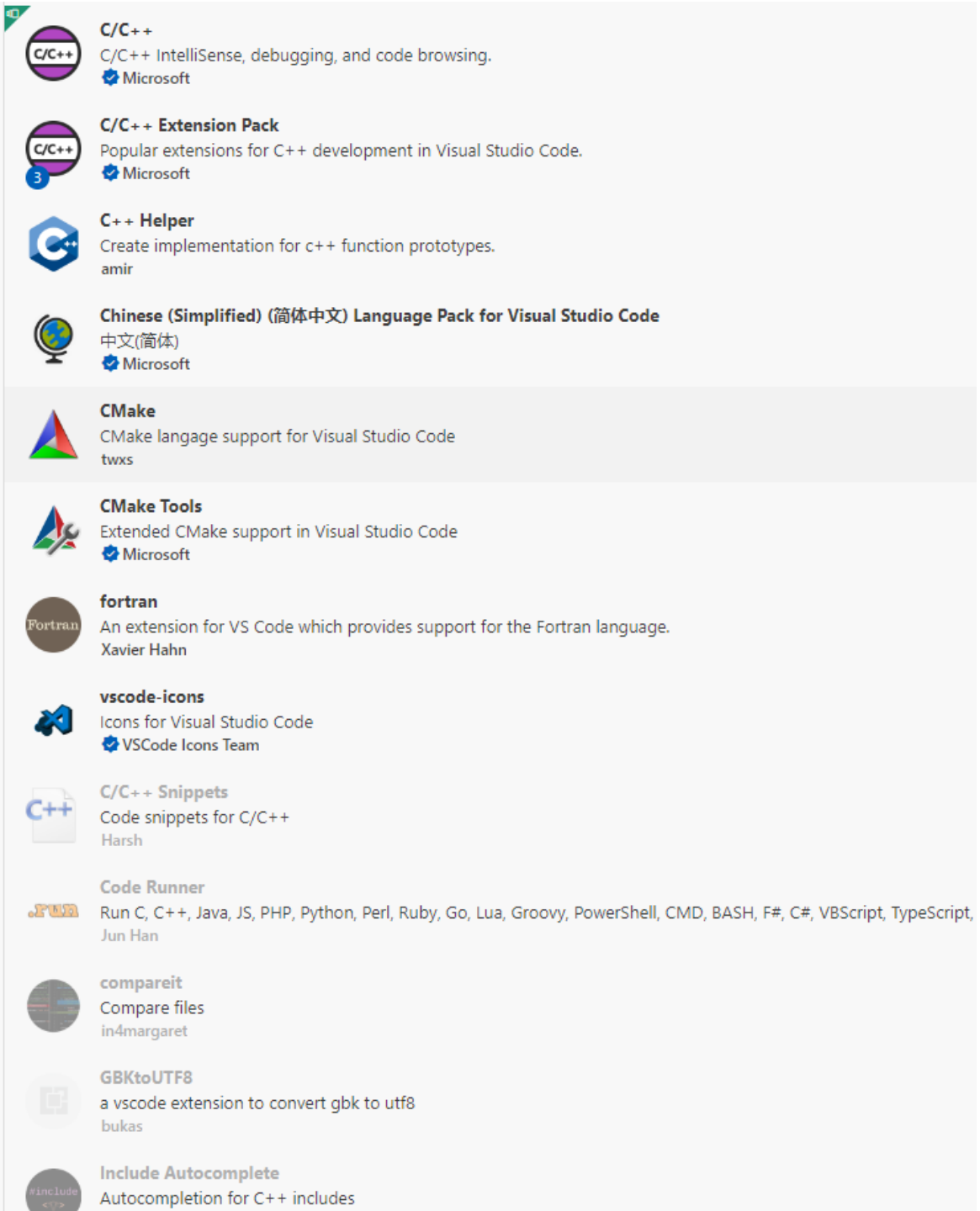
【Include AutoComplete】： 自动头文件包含

【GBKtoUTF8】： GBK编码转换为UTF8编码

【Chinese(Simplified)】： 简体中文环境，可要可不要

【vscode-icons】： VSCode 图标插件

【compareit】： 比较插件，可以用于比较两个文件的差异



2、文件结构

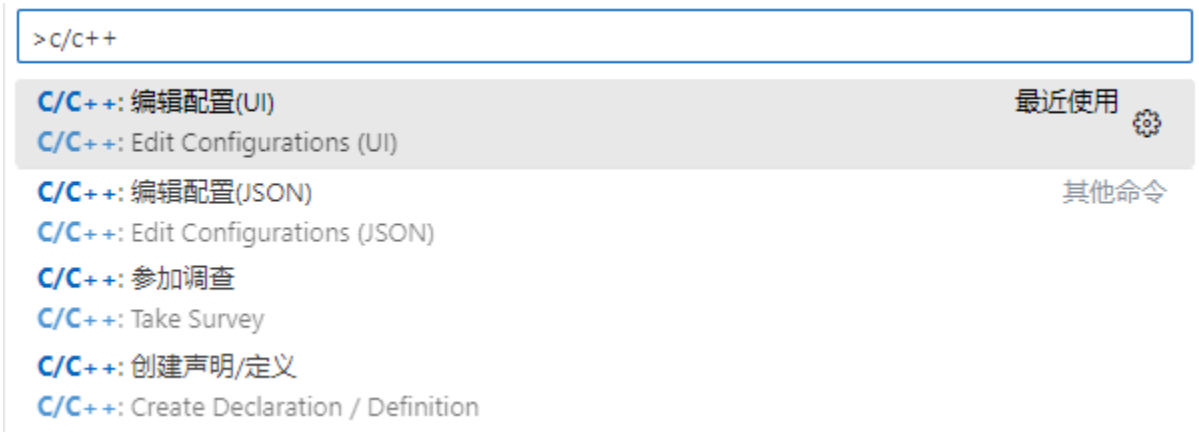
文件结构就是你组织文件夹、文件，决定他们怎样嵌套、怎样从属的方法。

3、配置文件

编译通过.vscode文件夹下的json配置文件实现的，其中一个是tasks.json，我们的编译和运行就是我们想要vscode执行的任务，为此我们要在**tasks.json**里写两个task：**Build和Run**(这里为什么不是Compile呢？是因为从源码到可执行的过程中不仅是编译(Compile)，还有预编译、链接等过程，用构建(Build)来表述更合适)。
除了编译和运行，我们还需要进行**调试(Debug)**，这个就不是通过task来实现的了，而是通过**launch.json**文件来实现。

3.1 配置编译器

按快捷键**Ctrl+Shift+P调出命令面板**，输入**C/C++，选择“Edit Configurations(UI)”**进入配置。
编译器路径：D:/mingw-w64/mingw64/bin/g++.exe



3、配置文件

3.1 配置编译器 c_cpp_properties.json

IntelliSense 配置

使用此编辑器编辑在基础 c_cpp_properties.json 文件中定义的 IntelliSense 设置。在此编辑器中所做的更改仅适用于所选的配置。要一次编辑多个配置，请转到 c_cpp_properties.json。

配置名称

用于标识某个配置的友好名称。Linux、Mac 和 Win32 是特殊标识符，用于将在这些平台上自动选择的配置。

选择要编辑的配置集。

Win32 ⌄ 添加配置

编译器路径

用于生成项目来启用更准确的 IntelliSense 的编译器的完整路径，例如 /usr/bin/gcc。扩展将查询编译器以确定要用于 IntelliSense 的系统是否包含路径和默认定义。

指定编译器路径或从下拉列表中选择检测到的编译器路径。

C:/Program Files/mingw64/bin/g++.exe ⌄

编译器参数

用于修改所使用的包含或定义的编译器参数，例如 -nostdinc++、-m32 等。采用其他空格分隔参数的参数应在数组中作为为单独的参数输入，例如，对于 --sysroot <arg> 使用 "--sysroot", "<arg>"。

每行一个参数。

IntelliSense 模式

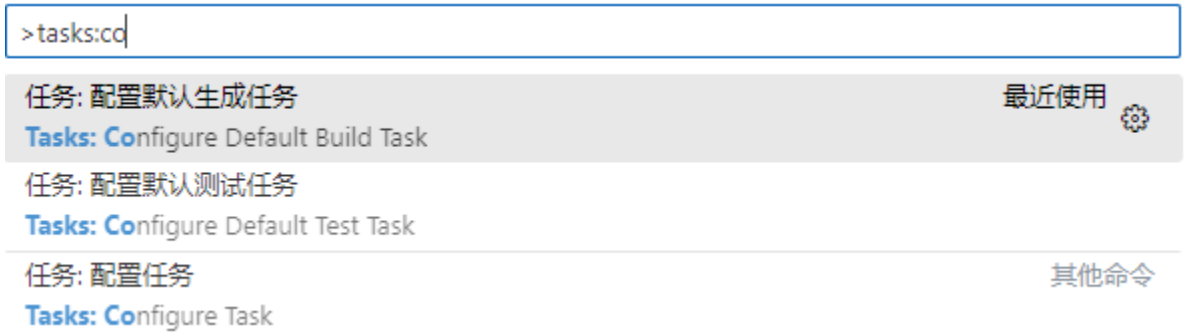
要使用的 IntelliSense 模式，该模式映射到 MSVC、gcc 或 Clang 的平台和体系结构变体。如果未设置或设置为 ${default}，则扩展将选择该平台的默认值。Windows 默认为 windows-msvc-x64，Linux 默认为 linux-gcc-x64，macOS 默认为 macos-clang-x64。选择特定 IntelliSense 模式以替代 ${default} 模式。仅指定 <compiler>-<architecture> 变体(例如 gcc-x64)的 IntelliSense 模式是旧模式，它们会根据主机平台自动转换为 <platform>-<compiler>-<architecture> 变体。

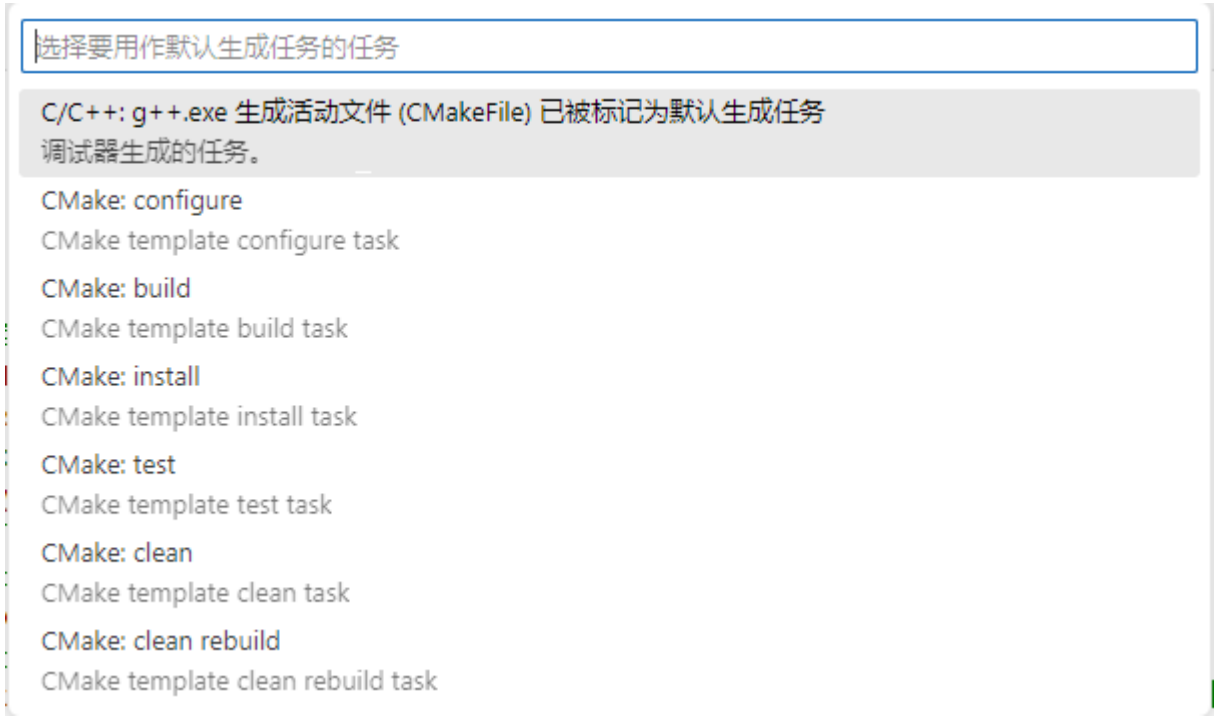
gcc-x64 (legacy) ⌄

3、配置文件

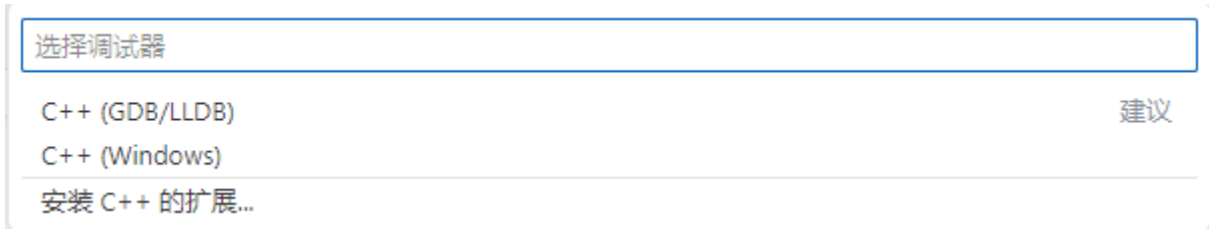
3.2 配置构建任务 生成tasks.json

告诉VS Code如何构建（编译）程序





3.3 配置调试设置 生成launch.json

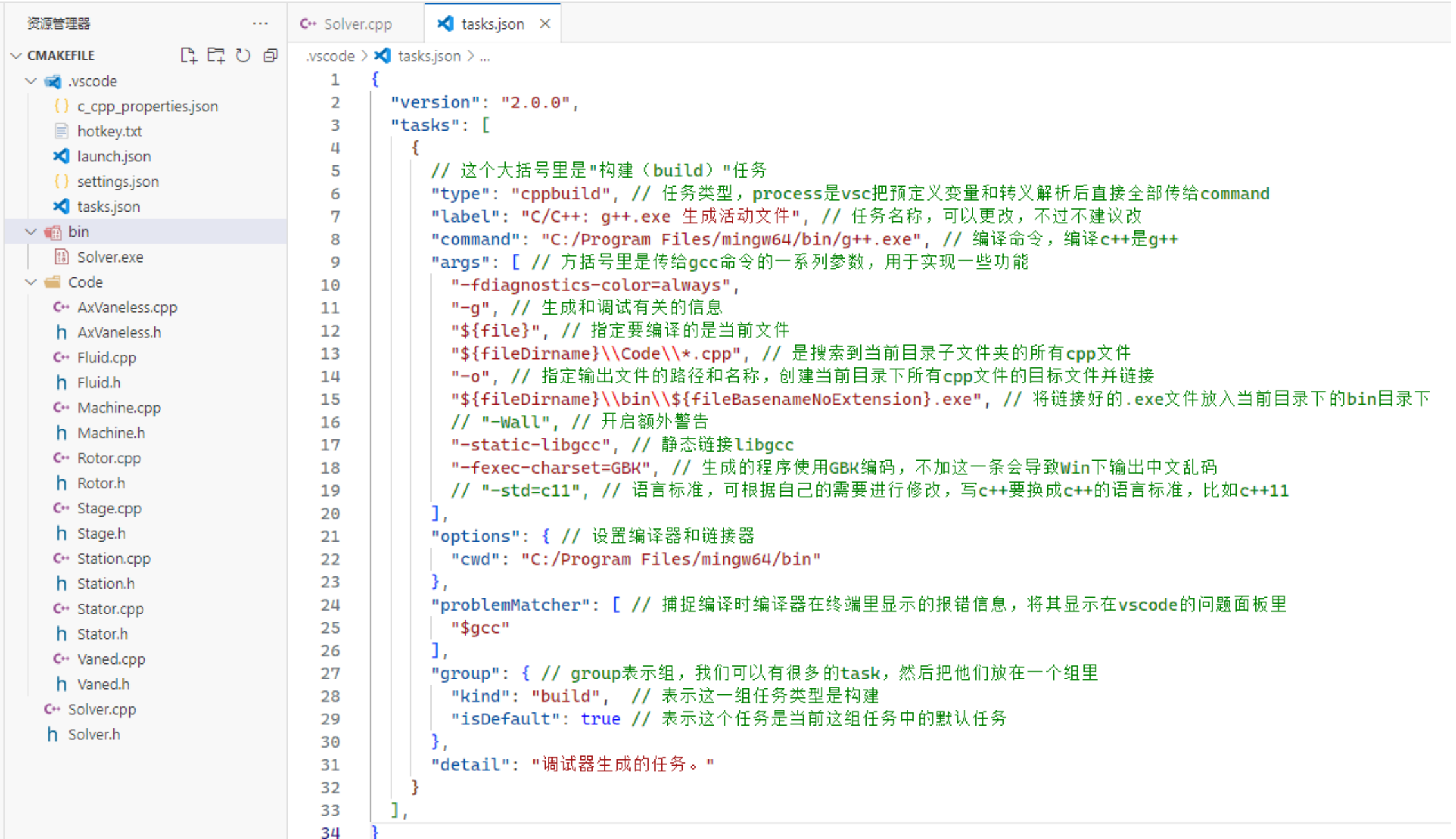



```
PS E:\VSCodeProject\CMakeFile> cd bin
PS E:\VSCodeProject\CMakeFile\bin> .\Solver.exe
Design axial Machine(Compressor or Turbine) Start!
Input 0: axial Turbine, Input 1: axial Compressor.
please choose your axial machine type:
1
MachineType is Axial Compressor!
Design axial Machine(Compressor or Turbine) End!
```

```
E:\VSCodeProject\CMakeFile>cd bin

E:\VSCodeProject\CMakeFile\bin>Solver.exe
Design axial Machine(Compressor or Turbine) Start!
Input 0: axial Turbine, Input 1: axial Compressor.
please choose your axial machine type:
0
MachineType is Axial Turbine!
Design axial Machine(Compressor or Turbine) End!
```

3、配置文件


资源管理器
CMAKEFILE
.vscode
c_cpp_properties.json
hotkey.txt
launch.json
settings.json
tasks.json
bin
Solver.exe
Code
AxVaneless.cpp
AxVaneless.h
Fluid.cpp
Fluid.h
Machine.cpp
Machine.h
Rotor.cpp
Rotor.h
Stage.cpp
Stage.h
Station.cpp
Station.h
Stator.cpp
Stator.h
Vaned.cpp
Vaned.h
Solver.cpp
Solver.h
Solver.cpp
tasks.json
.vscode > tasks.json > ...
{
"version": "2.0.0",
"tasks": [
{
// 这个大括号里是"构建（build）"任务
"type": "cppbuild", // 任务类型，process是vsc把预定义变量和转义解析后直接全部传给command
"label": "C/C++: g++.exe 生成活动文件", // 任务名称，可以更改，不过不建议改
"command": "C:/Program Files/mingw64/bin/g++.exe", // 编译命令，编译c++是g++
"args": [// 方括号里是传给gcc命令的一系列参数，用于实现一些功能
"-fdiagnostics-color=always",
"-g", // 生成和调试有关的信息
"${file}", // 指定要编译的是当前文件
"${fileDirname}\\Code*.cpp", // 是搜索到当前目录子文件夹的所有cpp文件
"-o", // 指定输出文件的路径和名称，创建当前目录下所有cpp文件的目标文件并链接
"${fileDirname}\\bin\\${fileBasenameNoExtension}.exe", // 将链接好的.exe文件放入当前目录下的bin目录下
// "-Wall", // 开启额外警告
"-static-libgcc", // 静态链接libgcc
"-fexec-charset=GBK", // 生成的程序使用GBK编码，不加这一条会导致Win下输出中文乱码
// "-std=c11", // 语言标准，可根据自己的需要进行修改，写c++要换成c++的语言标准，比如c++11
],
"options": { // 设置编译器和链接器
"cwd": "C:/Program Files/mingw64/bin"
},
"problemMatcher": [// 捕捉编译时编译器在终端里显示的报错信息，将其显示在vscode的问题面板里
"$gcc"
],
"group": { // group表示组，我们可以有很多的task，然后把他们放在一个组里
"kind": "build", // 表示这一组任务类型是构建
"isDefault": true // 表示这个任务是当前这组任务中的默认任务
},
"detail": "调试器生成的任务。"
}
],
}

4、CMake

4.0.1.概述: 在windows上使用CMake编译C/C++程序时，首先需要下载CMake,安装gcc/g++编译环境，然后使用VSCode 以及配置下CMakelist.txt

4.0.2.Cmake工具:你或许听过好几种 Make 工具，例如 GNU Make 、QT 的 qmake 、微软的 MS nmake、BSD Make（pmake）、Makepp。这些 Make 工具遵循着不同的规范和标准，所执行的 Makefile 格式也千差万别。这样就带来了一个严峻的问题：如果软件想跨平台，必须要保证能够在不同平台编译。而如果使用上面的 Make 工具，就得为每一种标准写一次 Makefile ，这将是一件让人抓狂的工作。CMake就是针对上面问题所设计的工具：它首先允许开发者编写一种平台无关的 CMakeList.txt 文件来定制整个编译流程，然后再根据目标用户的平台进一步生成所需的本地化 Makefile 和工程文件，如 Unix 的 Makefile 或 Windows 的 Visual Studio 工程。显然，CMake 是一个比上述几种 make 更高级的编译配置工具。一些使用 CMake 作为项目架构系统的知名开源项目有 VTK、ITK、KDE、OpenCV、OSG 等。

4.0.3.Cmake流程:编写 CMake 配置文件 CMakeLists.txt → 执行命令 CMake PATH 或者 ccmake PATH 生成 Makefile(ccmake 和 CMake 的 区别在于前者提供了一个交互式的界面)，其中， PATH 是 CMakeLists.txt 所在的目录 → 使用 make 命令进行编译

4.0.4.Cmake准备: 安装VSCode插件 → 安装MinGW → 安装CMake

4、CMake

4.1.安装CMake

下载链接：https://cmake.org/download/ **验证：cmd命令提示符 → cmake --version(中间有空格)**
尽量选择Latest Release版本，比较稳定。下载后缀为.msi的安装文件，然后直接安装。

Latest Release (3.27.1)

The release was packaged with CPack which is included as part of the release. The .sh files are self extracting gziped tar files. To install a .sh file, run it with /bin/sh and follow the directions. The OS-machine.tar.gz files are gziped tar files of the install tree. The OS-machine.tar.Z files are compressed tar files of the install tree. The tar file distributions can be untared in any directory. They are prefixed by the version of CMake. For example, the linux-x86_64 tar file is all under the directory cmake-linux-x86_64. This prefix can be removed as long as the share, bin, man and doc directories are moved relative to each other. To build the source distributions, unpack them with zip or tar and follow the instructions in README.rst at the top of the source tree. See also the CMake 3.27 Release Notes.

Source distributions:

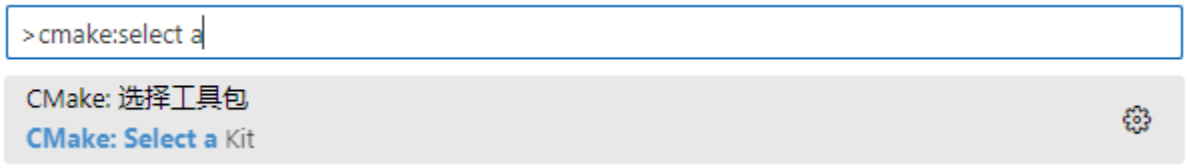
Platform	Files
Unix/Linux Source (has \n line feeds)	cmake-3.27.1.tar.gz
Windows Source (has \r\n line feeds)	cmake-3.27.1.zip

Binary distributions:

Platform	Files
Windows x64 Installer:	cmake-3.27.1-windows-x86_64.msi
Windows x64 ZIP	cmake-3.27.1-windows-x86_64.zip
Windows i386 Installer:	cmake-3.27.1-windows-i386.msi
Windows i386 ZIP	cmake-3.27.1-windows-i386.zip
Windows ARM64 Installer:	cmake-3.27.1-windows-arm64.msi
Windows ARM64 ZIP	cmake-3.27.1-windows-arm64.zip
macOS 10.13 or later	cmake-3.27.1-macos-universal.dmg
	cmake-3.27.1-macos-universal.tar.gz

4、CMake

4.2.VSCode配置CMake



4.3.VSCode编写CMake

建立目录（Build/CMakeLists.txt/Code）

cd Build

进入目录，执行cmake

cmake -G "MinGW Makefiles" ..

执行make，生成可执行文件

mingw32-make

```
E:\VSCodeProject\CMakeFile>cd Build

E:\VSCodeProject\CMakeFile\Build>cmake -G "MinGW Makefiles" ..
CMake Deprecation Warning at CMakeLists.txt:4 (cmake_minimum_required):
  Compatibility with CMake < 3.5 will be removed from a future version of
  CMake.

  Update the VERSION argument <min> value or use a ...<max> suffix to tell
  CMake that the project does not need compatibility with older versions.


-- The C compiler identification is GNU 8.1.0
-- The CXX compiler identification is GNU 8.1.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files/mingw64/bin/gcc.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files/mingw64/bin/c++.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (1.0s)
-- Generating done (0.0s)
-- Build files have been written to: E:/VSCodeProject/CMakeFile/Build

E:\VSCodeProject\CMakeFile\Build>mingw32-make
[ 50%] Building CXX object CMakeFiles/main.dir/main.cpp.obj
[100%] Linking CXX executable main.exe
[100%] Built target main
```

4、CMake

4.4. CMakeLists.txt

.vscode
bin
Build
Include
lib
src
CmakeLists.txt