**学习记录**

**一、学习内容：**

**1、配置环境**

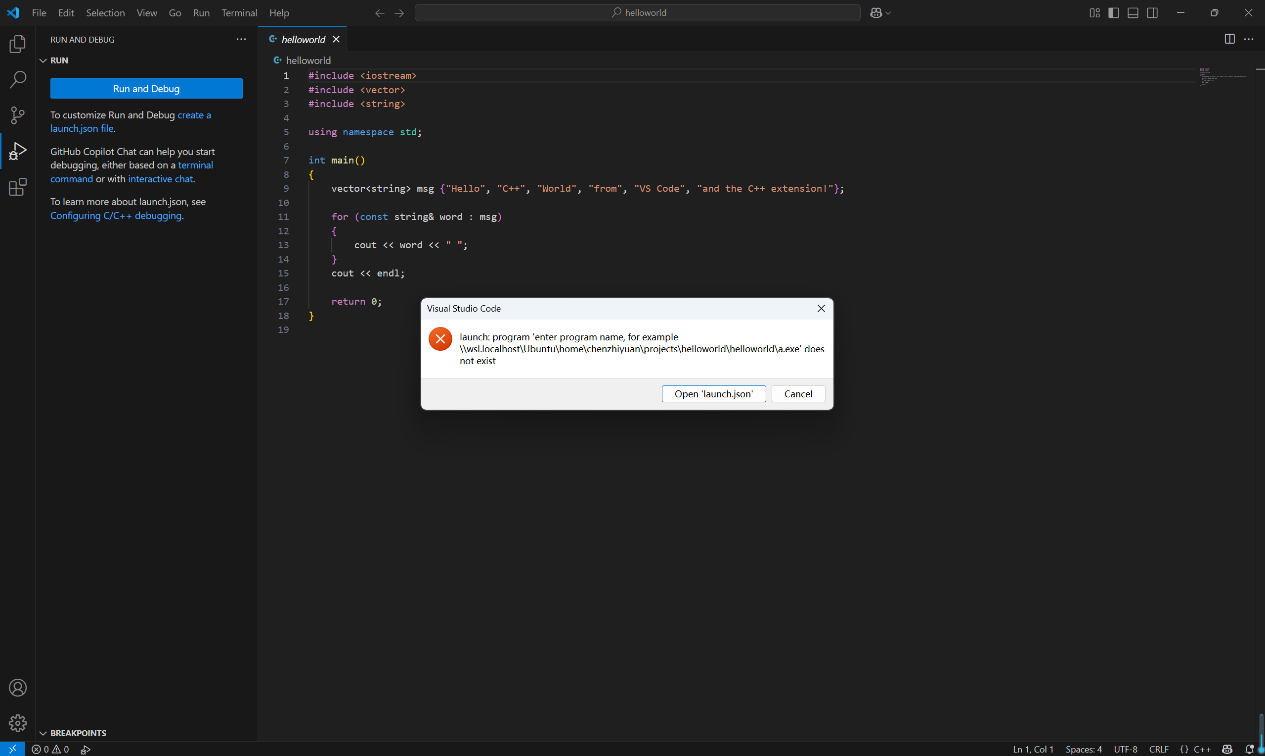
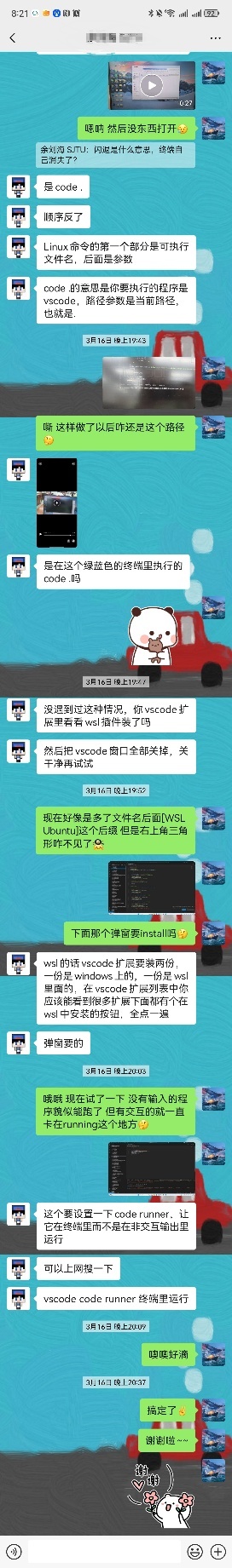
我上学期写C语言代码一直是用的devc++，这是第一次接触VSCode以及在电脑上配置Linux系统的相关环境。我先仔细研读了题目中前序说明给出的参考文件，顺利的安装了WSL、Ubuntu以及VSCode。然而，在尝试在VSCode中运行最简单的hello world程序时却遇到了问题（如图1），系统显示找不到编译器。尝试一晚上未果后，我于第二天向学长请教了这个问题（如图2），并与他进行了交流。在他的指导下，我在VSCode中安装了WSL和Code Runner两个拓展，并学会了区分Linux以及Windows在终端中显示的路径以及如何从Windows文件资源管理器中通过Linux打开文件。

图1 配置环境时遇到的问题界面

图2 相关聊天记录  
 在可以运行helloworld这样无需交互的程序后，我发现在“output”里仍然无法正常运行需要交互的程序，随后我通过查阅资料（图3）实现了让code runner插件在集成终端里面直接运行，解决了这一问题，基本搭建好了VSCode中的开发环境。

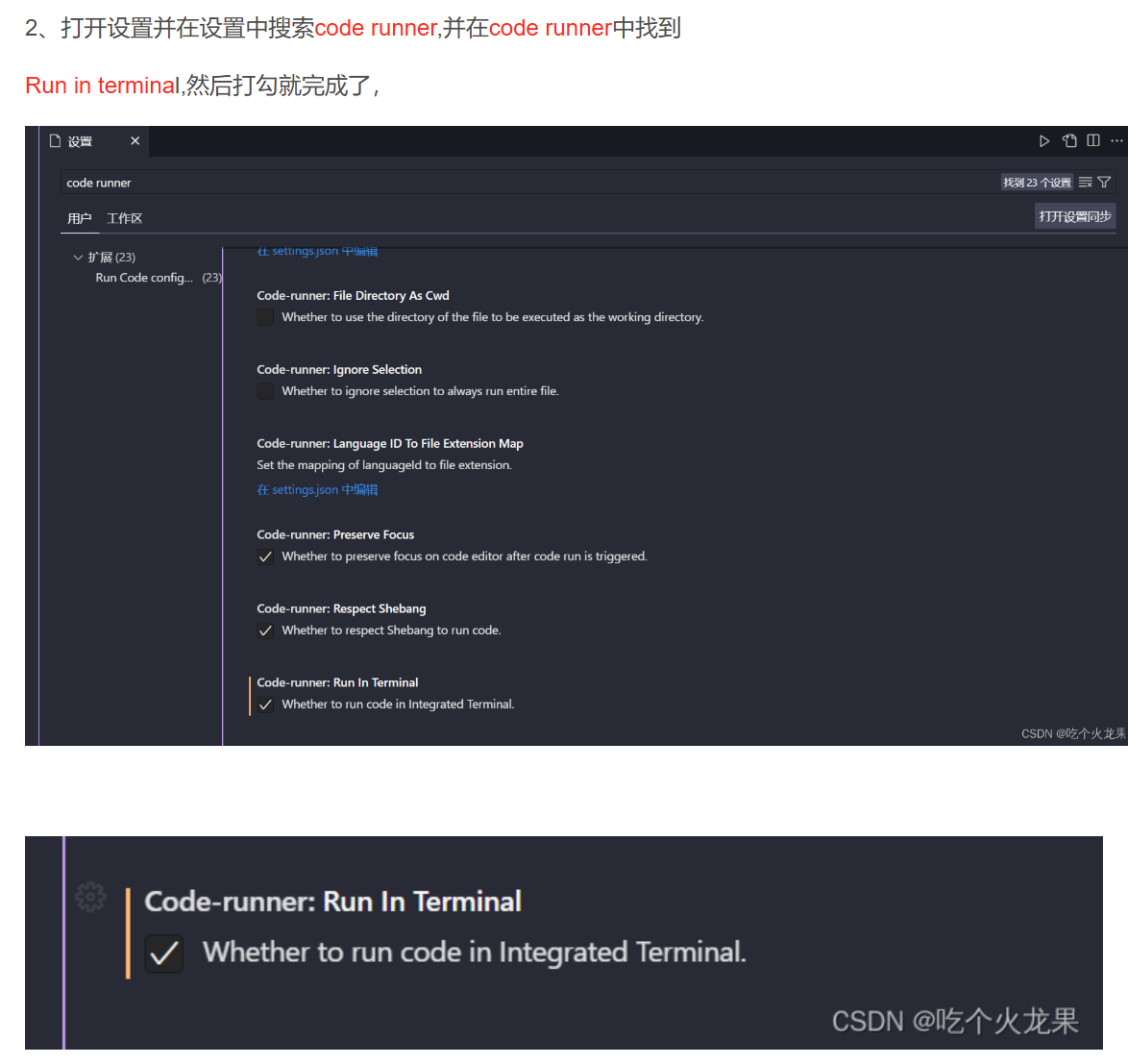
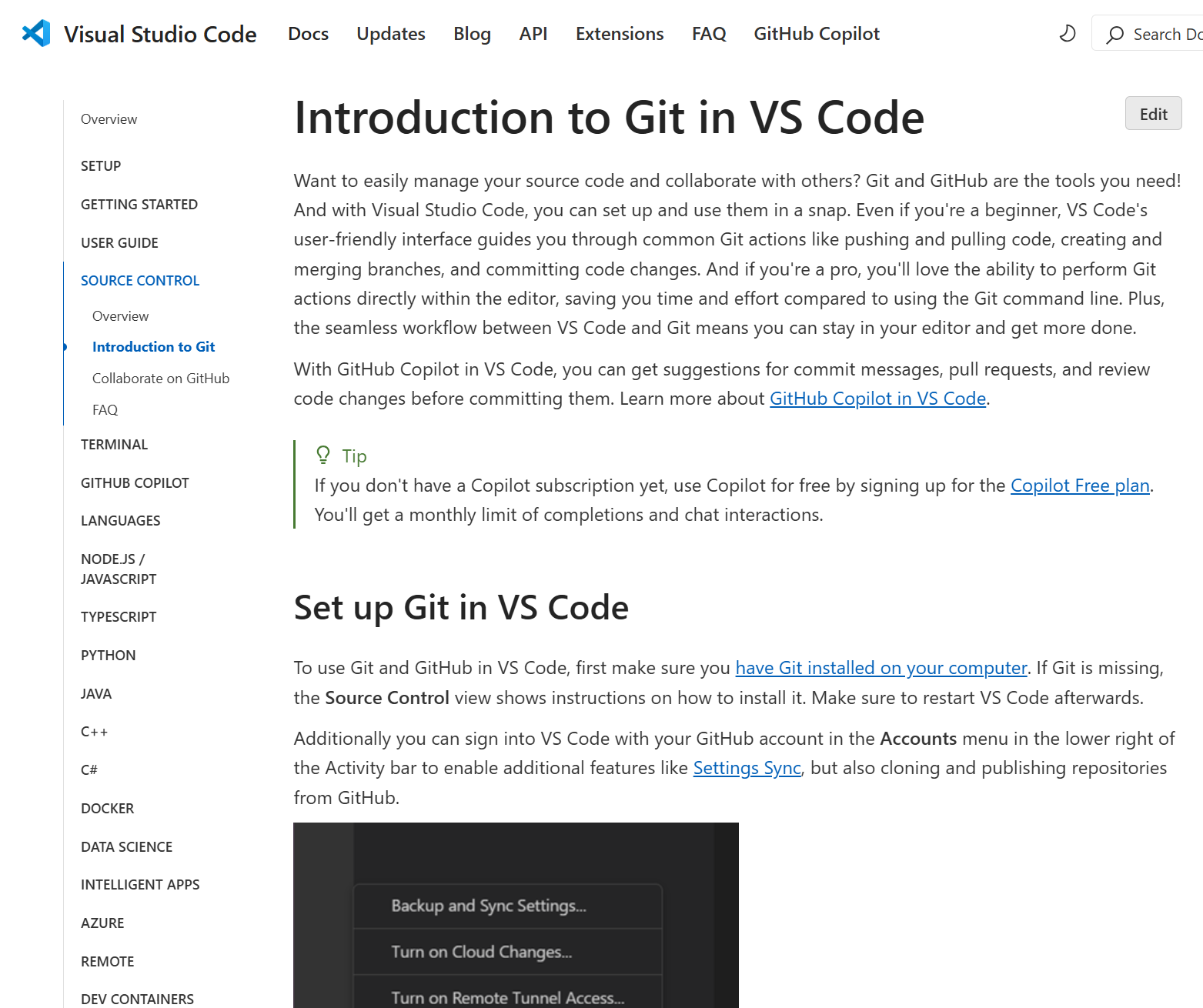


图3 在博客中查阅的相关资料

**2、建立并使用git以及GitHub**

初次接触git代码版本管理器以及GitHub远程仓库，我学习了如何在VSCode中进行相关的配置（如图4），并成功建立了GitHub仓库。同时我也学会了在VSCode中进行相关提交上传操作，实时更新我的代码库。

图4 相关资料

**3、了解make工具以及多文件开发方法**

作为初次使用多文件开发方式的新手，我通过咨询AI助手、查找网上资料等方式初步了解了如何正确使用多文件开发（如如何正确建立.h头文件，如何在其他程序中进行调用等）。同时我也在实操中遇到了一些问题，比如我一开始的文件命名与代码中的声明不相符，这样就会一直产生报错信息（图5）；我也了解了在同一文件夹中不一定只能有一个main.c，但是在程序运行时只能有一个main函数接口；以及make 是一个自动化构建工具，用于根据 Makefile 文件中的规则，自动编译和链接多文件项目。我也尝试了使用在终端中的命令语句进行编译以及通过编写makefile文件进行编译两种方式，在该过程中初步了解了终端编译命令的语法以及makefile文件的一些注意事项，比如缩进要用Tab键等；同时也了解了编译的四个流程，即预处理、编译、汇编与链接。

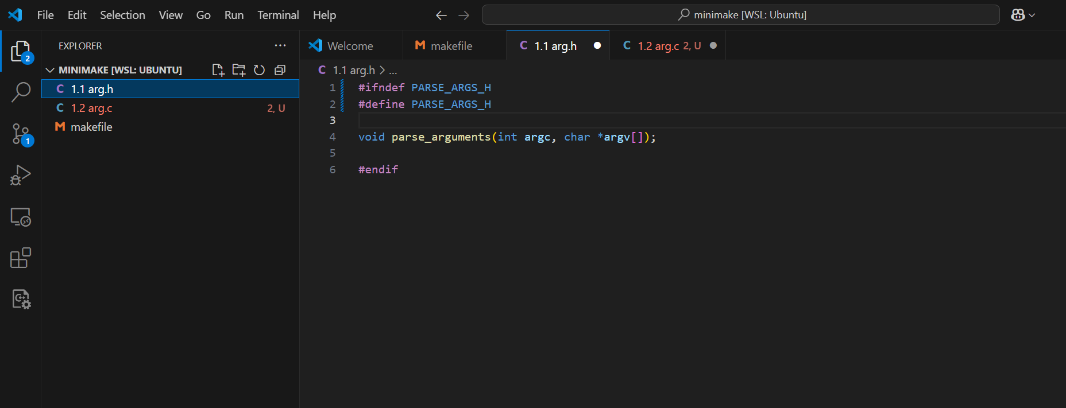


图5 定义头文件时的错误

**4、学习相关代码编写**

这一部分的学习内容主要可以在所提交代码中得以体现。在编程过程中，我尽量将原有的指针、数组、结构体等基础知识运用在此次编程中，但也会遇到一些新的知识。比如在多文件开发环境中，对于变量类型、函数接收端口等设计的语法细节就应加以调整；同时在大型项目中也应该尽量做到将程序的时间、空间复杂度降到最低，故我也在原先自己习惯的基础上添加了内存管理的相关语句；我也学习了如何在程序编写中完善自己的逻辑结构，尽量提升代码的健壮性：比如在读取文件时先判断目标文件是否存在，若不存在则应该做出相应防护以防止后续程序崩溃；相应的我也学习了一些错误处理函数（比如perror函数、fprintf函数写入错误信息流）以及一些关于文件处理的函数语句（比如fopen与fclose、stdio.h中的文件流等）。

在本次练习过程中，我接触到了很多新知识，采用了比往常编写程序更严谨的逻辑，也借此机会完成了很多此前不敢去做的事，对我来说意义非凡。

**二、代码结构：**

**阶段一**

任务1 arg.h arg.c —— 完成识别“—help”以及乱码信息的逻辑架构

main1A.c —— 该程序段主函数

任务2 preprocess.h preprocess.c —— 完成makefile文件的预处理

main1B.c —— 该程序段主函数

任务3 utils.h utils.c —— 对makefile文件进行清理

checker.h checker.c —— 解析makefile文件语句语法的逻辑程序

main1C.c —— 该程序段主函数

**阶段二**

任务一 target.h —— 预先定义存储目标和错误信息所需的结构体

parser.h parser.c —— 解析makefile文件，检测重复目标，分割依赖项

relychecker.h relychecker.c —— 检查目标依赖关系

main2A.c —— 该程序段主函数