**实验五 Linux开发工具的使用**

班级： 姓名： 学号： 上机时间：

任课教师： 实验教师： 实验成绩：

**一、实验目的**

1．掌握C语言编译的基本方法。

2．理解makefile文件，学会make工具的使用。

**二、实验注意事项**

实验室内的实验环境与系统是共用设施，请不要在系统内做对系统或对其他用户不安全的事情。

要求每个同学登录后系统后，要在自己的家目录下创建一个属于自己的子目录（以自己(拼音)名字或学号）。以后所有工作都要在自己的目录内进行。建议以后的实验都在同台计算机上做，这样可以保持连续性。

用户要按通常实验要认真书写实验报告。

**三、实验内容及步骤**

**1. 用gcc带不同参数编译下列hello.c程序。**

#include <stdio.h>

int main()

{

printf(”Hello World!\n”);

return 0;

}

（1）只作预处理，生成hello.i，相应命令为： gcc -E hello.c -o hello.i

（2）只进行编译，不做汇编，生成汇编代码，命令为： gcc -S hello.c -o hello.s

（3）只进行汇编，不做连接，生成目标代码，命令为： gcc -c hello.c -o hello.o

（4）以默认方式生成可执行程序a.out，命令为： gcc hello.c

（5）生成可执行程序hello，命令为： gcc hello.c -o hello

**2.假设我们有一个程序由5个文件组成，源代码如下所示，根据以下步骤，熟悉makefile编程。**

/\*main.c\*/

#include "mytool1.h"

#include "mytool2.h"

int main()

{

mytool1\_print("hello mytool1!");

mytool2\_print("hello mytool2!");

return 0;

}

/\*mytool1.c\*/

#include "mytool1.h"

#include <stdio.h>

void mytool1\_print(char \*print\_str)

{

printf("This is mytool1 print : %s ",print\_str);

}

/\*mytool1.h\*/

#ifndef \_MYTOOL\_1\_H

#define \_MYTOOL\_1\_H

void mytool1\_print(char \*print\_str);

#endif

/\*mytool2.c\*/

#include "mytool2.h"

#include <stdio.h>

void mytool2\_print(char \*print\_str)

{

printf("This is mytool2 print : %s ",print\_str);

}

/\*mytool2.h\*/

#ifndef \_MYTOOL\_2\_H

#define \_MYTOOL\_2\_H

void mytool2\_print(char \*print\_str);

#endif

根据实验要求，写出第一个Makefile如下：

main:main.o mytool1.o mytool2.o

gcc -o main main.o mytool1.o mytool2.o

main.o:main.c mytool1.h mytool2.h

gcc -c main.c

mytool1.o:mytool1.c mytool1.h

gcc -c mytool1.c

mytool2.o:mytool2.c mytool2.h

gcc -c mytool2.c

clean:

rm -f \*.o

在shell提示符下输入make，执行显示：

gcc -c main.c

gcc -c mytool1.c

gcc -c mytool2.c

gcc -o main main.o mytool1.o mytool2.o

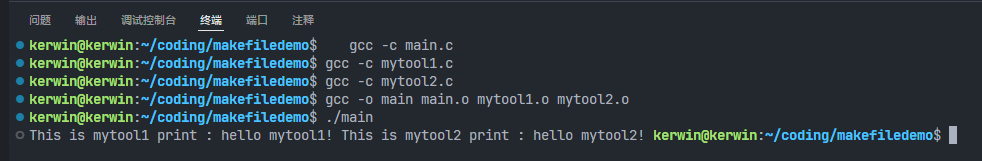
执行结果如下：

[armlinux@lqm makefile-easy]$ ./main

This is mytool1 print : hello mytool1!

This is mytool2 print : hello mytool2!

这只是最为初级的Makefile，现在来对这个Makefile进行改进。



**改进一：使用变量**

一般在书写Makefile时，各部分变量引用的格式如下：

1) make变量（Makefile中定义的或者是make的环境变量）的引用使用“$(VAR)”格式，无论“VAR”是单字符变量名还是多字符变量名。

2) 出现在规则命令行中shell变量（一般为执行命令过程中的临时变量，它不属于Makefile变量，而是一个shell变量）引用使用shell的“$tmp”格式。

3) 对出现在命令行中的make变量同样使用“$(CMDVAR)” 格式来引用。

OBJ=main.o mytool1.o mytool2.o

main:$(OBJ)

gcc -o main $(OBJ)

main.o:main.c mytool1.h mytool2.h

gcc -c main.c

mytool1.o:mytool1.c mytool1.h

gcc -c mytool1.c

mytool2.o:mytool2.c mytool2.h

gcc -c mytool2.c

clean:

rm -f main $(OBJ)

**改进二：使用自动推导**

让make自动推导，只要make看到一个.o文件，它就会自动的把对应的.c文件加到依赖文件中，并且gcc －c .c也会被推导出来，所以Makefile就简化了。

CC = gcc

OBJ = main.o mytool1.o mytool2.o

main: $(OBJ)

$(CC) -o main $(OBJ)

main.o: mytool1.h mytool2.h

mytool1.o: mytool1.h

mytool2.o: mytool2.h

.PHONY: clean

clean:

rm -f main $(OBJ)

**改进三：自动变量（$^  $<  $@）的应用**

Makefile 有三个非常有用的变量，分别是$@、$^、$<。代表的意义分别是:

$@--目标文件，

$^--所有的依赖文件，

$<--第一个依赖文件。

CC = gcc

OBJ = main.o mytool1.o mytool2.o

main: $(OBJ)

    $(CC) -o $@ $^

main.o: main.c mytool1.h mytool2.h

    $(CC) -c $<

mytool1.o: mytool1.c mytool1.h

    $(CC) -c $<

mytool2.o: mytool2.c mytool2.h

    $(CC) -c $<

.PHONY: clean

clean:

    rm -f main $(OBJ)

这些是最为初级的知识，现在至少可以减少编译时的工作量。细节方面的东西还需要在以后的学习中不断的总结，不断的深化理解。可以参考GNU Make手册，那里讲解的比较全面。

**3. 按下面要求，编制makefile程序。**

本示例共包含float类型加法、加法头函数、int类型加法、main主函数、float类型减法、减法头函数、int类型减法

**主函数**

1. #include "add.h"
2. #include "sub.h"
3. #include <stdio.h>
4. **int** main()
5. {
6. **int** x, y;
7. **float** a, b;
8. x=5;
9. y=2;
10. a=5.5;
11. b=2.2;
12. printf("%d + %d = %d/n", x, y, add\_int(x, y));
13. printf("%3.1f + %3.1f = %3.1f/n", a, b, add\_float(a, b));
14. printf("%d - %d = %d/n", x, y, sub\_int(x, y));
15. printf("%3.1f - %3.1f = %3.1f/n", a, b, sub\_float(a, b));
16. return 0;
17. }

**加法头函数**

1. /\* add.h \*/
2. #ifndef \_ADD\_H\_
3. #define \_ADD\_H\_
4. extern **int** add\_int(**int** x, **int** y);
5. extern **float** add\_float(**float** x, **float** y);
6. #endif

**int类型加法**

1. /\* add\_int.c \*/
2. **int** add\_int(**int** x, **int** y)
3. {
4. return x+y;
5. }

**float类型加法**

1. /\* add\_float.c \*/
2. **float** add\_float(**float** x, **float** y)
3. {
4. return x+y;
5. }

**减法头函数**

1. /\* sub.h \*/
2. #ifndef \_SUB\_H\_
3. #define \_SUB\_H\_
4. extern **int** sub\_int(**int** x, **int** y);
5. extern **float** sub\_float(**float** x, **float** y);
6. #endif

**int类型减法**

1. /\* sub\_int.c \*/
2. **int** sub\_int(**int** x, **int** y)
3. {
4. return x-y;
5. }

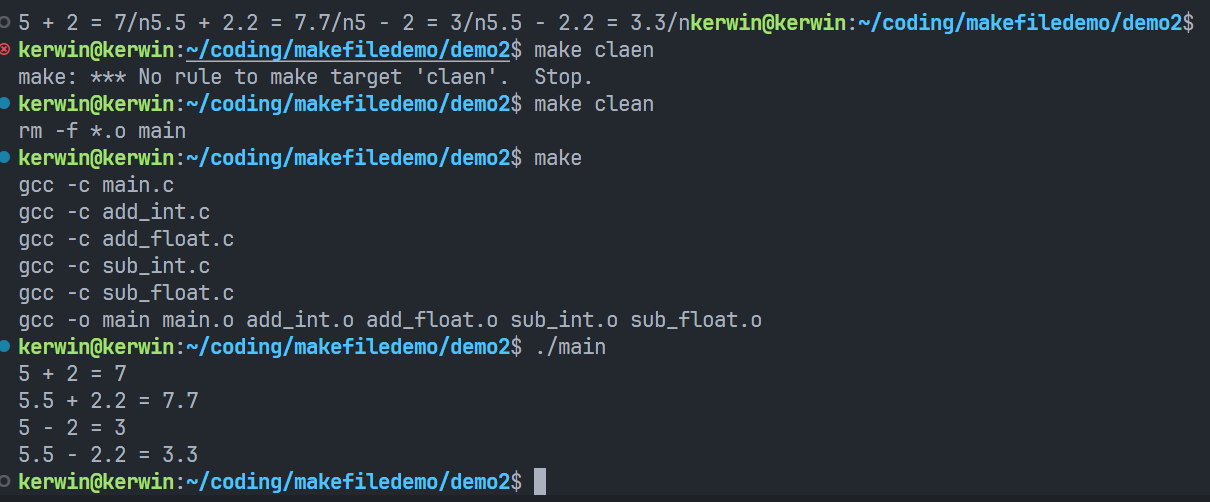
**float类型减法**

1. /\* sub\_float.c \*/
2. **float** sub\_float(**float** x, **float** y)
3. {
4. return x-y;
5. }

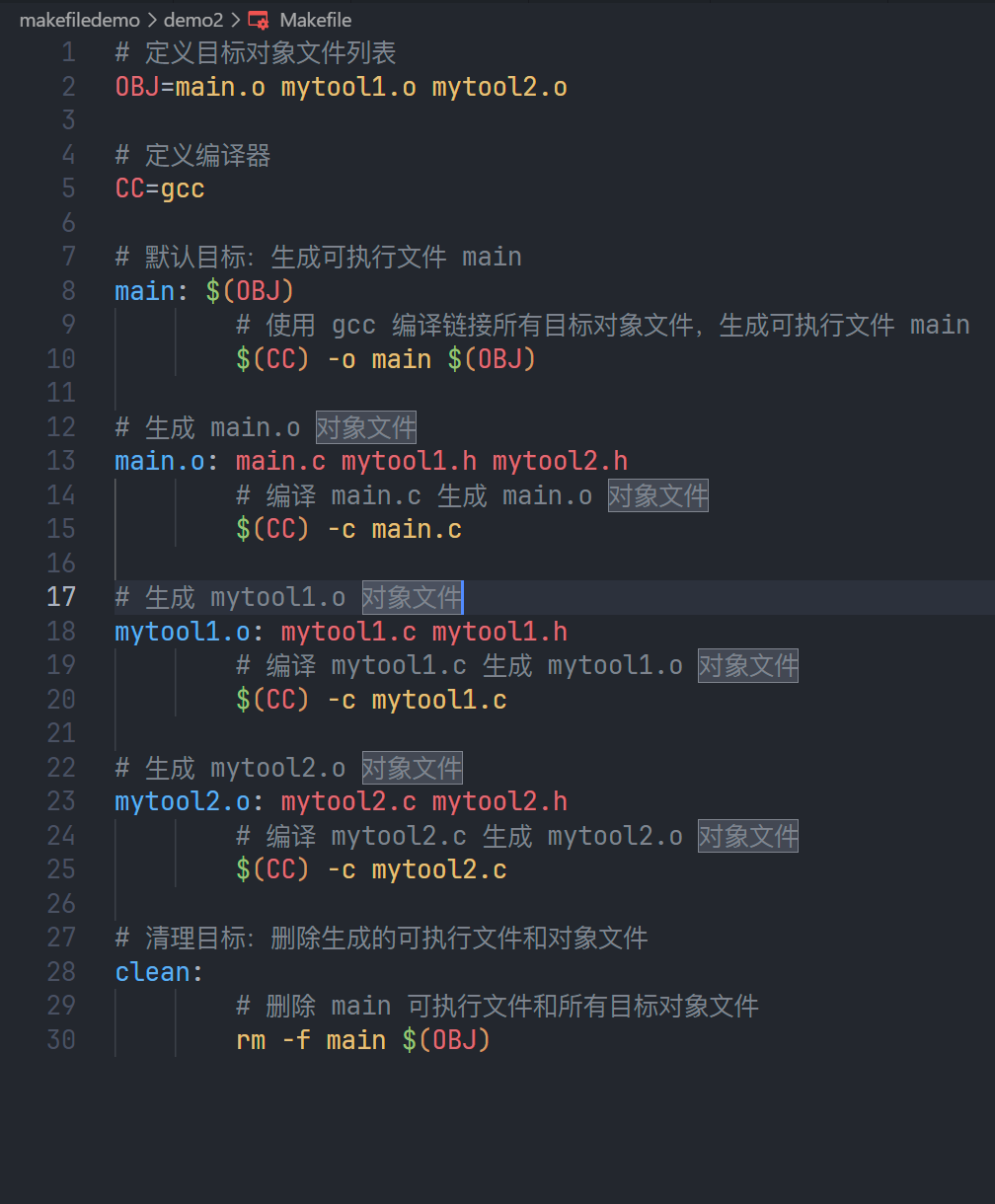
以上文件都准备好后,不使用变量编写makefile文件：

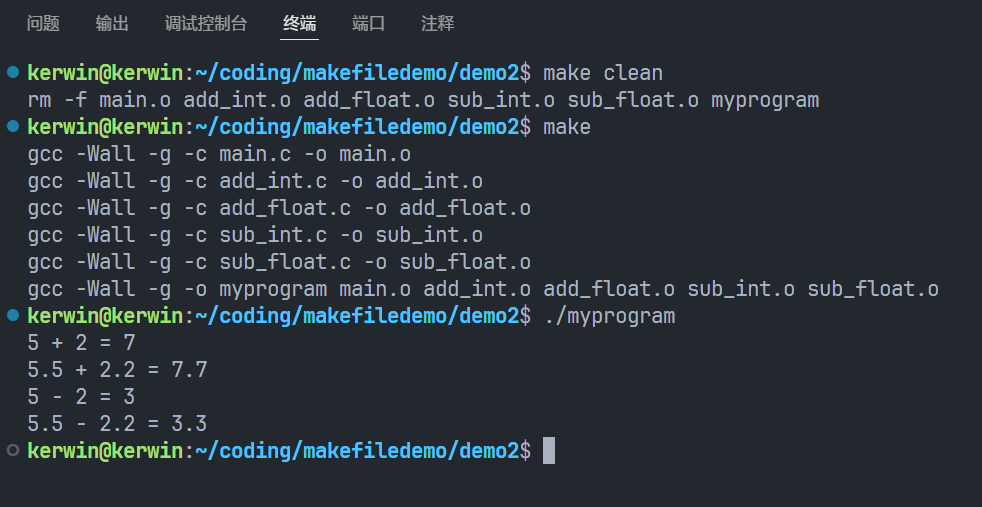
**文件名为Makefile**





使用变量编写makefile文件：





使用自动推导编写makefile文件：

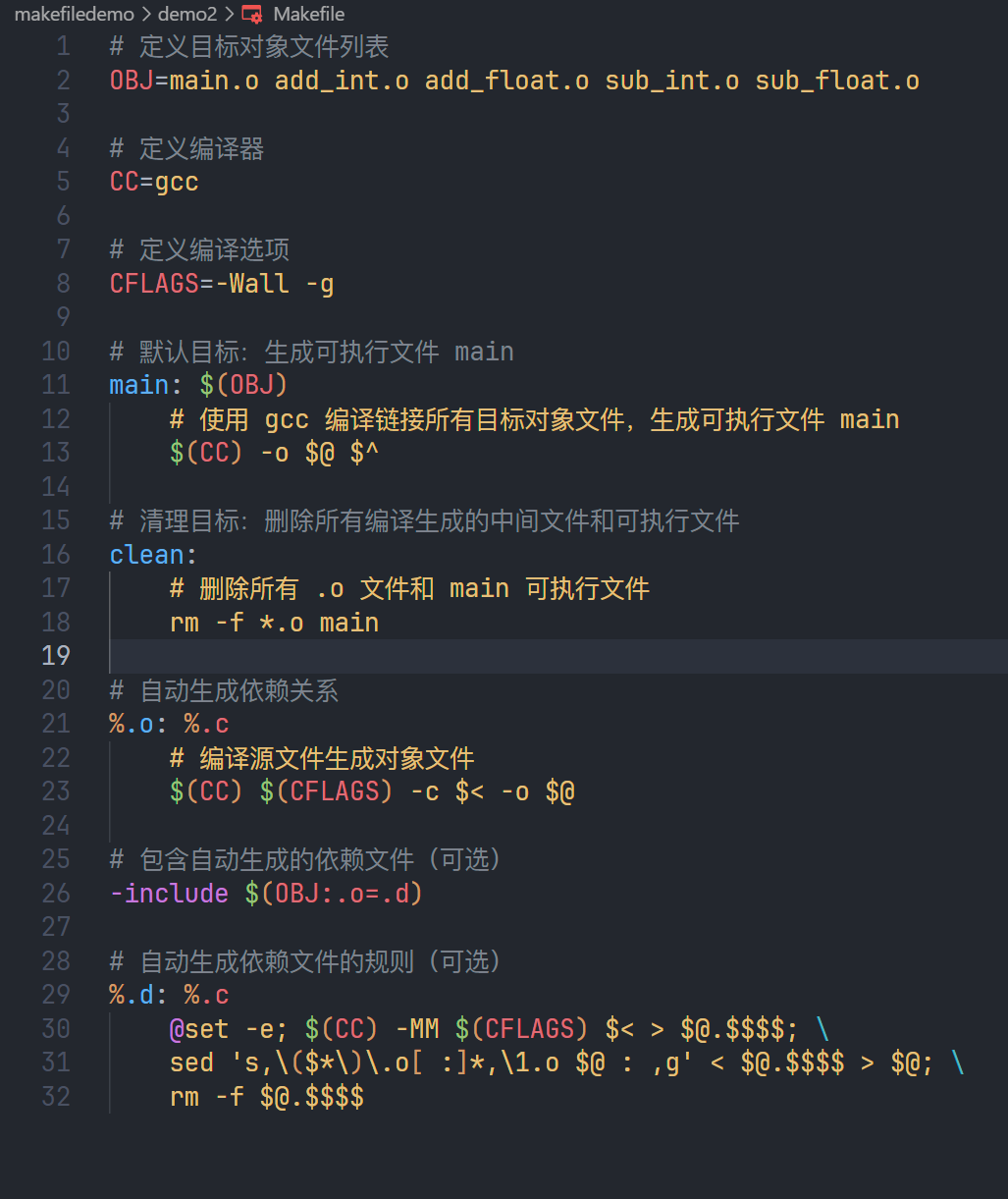
文本

描述已自动生成

文本

描述已自动生成

使用自动变量编写makefile文件：



文本

描述已自动生成

**四、实验报告要求**

1．以书面形式记录下你操作的每一步过程，包括编译及调试过程。

2. 回答所提出的问题。

3．总结上机调试过程中所遇到的问题和解决方法及感想。

本次实验中，我掌握了C语言编译的基本方法，学会了使用 gcc 编译器进行单个文件和多个文件的编译。通过编写和使用 Makefile 文件，我理解了如何自动化编译过程，减少了重复的编译命令，提高了开发效率。在实验过程中，我遇到了一些常见的问题，如文件路径错误和依赖关系不明确等，通过查阅文档和调试，这些问题都得到了解决。此外，我还学习了如何生成和使用依赖文件，这有助于管理复杂的项目依赖关系。通过这次实验，我对C语言的编译流程和Makefile的使用有了更深入的理解，为今后的项目开发打下了坚实的基础。