This lecture discusses the details of the first homework.

Let’s start from the main dot c file.

At the top of the file, several files are included. These are called the header files. The top four, S T D I O dot H, S T D L I B dot H, S T R I N G dot H, and S T D B O O L dot H are header files provided by the C language. These header files define the information needed for the program.

The first header file declares the standard input and output functions, such as printing and reading from files. The second header file defines exit failure and exit success. The third header file declares the functions for strings, such as comparing two strings. The fourth defines the logic values, true and false.

In typical Linux settings, these header files should be stored in the directory slash U S R slash I N C L U D E.

.

At line twelve, this file includes another header file. This file is defined by our own program, not the C language. Please notice that this header file’s name is enclosed by quotation marks.

Before we talk about the rest of main dot c, let’s read header dot H first.

The top of header dot h, at line 5 and line 6, has I F N D E F and D E F I N E.

.

What do they mean? Why do we need them?

Header files are included, not compiled or linked. In a complex program, it is possible that the same header file is included by several dot C files. As a result, the same information may be seen by G C C multiple times. When this occurs, G C C may get confused by seeing the same information again and again. The top I F N D E F and D E F I N E. ensure that the header file is included only once. The fifth line must be matched by E N D I F at the very bottom of the header file.

This header file declares four functions: add op, mul op, sub op, and div op. They represent the functions for addition, multiplication, subtraction, and division.

These are the declarations of the functions. The purpose of function declaration is to inform G C C that these functions will be provided at the link time, not compile time. When a C file is compiled, G C C assumes the declared functions exist somewhere and these functions will be found later.

This header file declares four functions, Each function takes two arguments; both are long integers. Each function also returns one long integer.

Let’s return to the main function.

The main function is enclosed by I F D E F test main. This allows us to turn on. or off. this main function by defining. or not .defining test \_. main.

The main function needs four arguments: two are integers and one indicates the operation. The operation can be addition, subtraction, multiplication, or division.

The function checks whether A R G C is 4. If it is not 4, the program returns exit failure and stops.

If A R G C is four, the program continues. Vee A Ell A and. Vee A Ell B . are long integers by converting the strings A R G Vee one and two using the S T R T O L function explained earlier.

If A R G Vee three is upper case letter. A, the add op function is called. Since this is a valid operation, valid is set to true.

The program continues checking whether A R G Vee is S, M, or D.

.

If A R G Vee three is none of the four allowed operations, the program prints an error message and returns exit failure.

If A R G Vee three is one of the four allowed operations, the program returns exit success.

Next, let’s see add dot C.

.

This is a relatively simple file. At the top, it includes the header file we saw earlier. The function is enclosed by I F D E F., test \_.. add and E N D I F, .

If test \_.. add is defined, the function is turned on. If test \_. . add is not defined, this function is discarded.

Why is this useful? Let’s look at what solution dot C has.

It has something similar. However, there is a crucial difference. Instead of using I F D E F, it uses

I F N. D E F.

.

By adding N, this function is used only if test \_. add is not defined.

Thus, test \_. add decides which add op to use. If test \_. add is defined, the add op function in add dot C is used. If test \_. add is not defined, the add op function in solution dot C is used.

This is how all programming assignments will be structured and graded.

Each function you write should be enclosed by a pair of I F D E F and E N D I F.

.

If the test symbol is defined, your function is used. If the test symbol is not defined, a reference solution is used.

By using a reference solution, it is possible to grade other parts of your programs even if this part is incorrect.

How will the symbol be defined or not defined?

This is controlled by make file.

In make file, the test flags define what functions will be tested.

Now, let’s run the program and see what we get.

If the arguments are 4, 9, and. A, the output is 13.

If the arguments are 4, 9, and. S, the output is minus5.

If the arguments are 4, 9, and. M, the output is 36.

If the arguments are 4, 9, and. T, the output is “unknown operation T”.

.

If we remove minus D TEST \_. ADD from the test flag, remove all object files, and run make again. This is what we get.

If the operation is add, the reference solution is used.

For subtraction and multiplication, the reference solution is not used.

Make file can be used for testing.

If we scroll down make file, we can find a section about testsub. It depends on main. This means that if the executable file called main does not already exist, make will create the executable file. This was specified earlier in the make file.

If the executable file already exists, then test sub will execute the program with two test cases. The first test case has arguments 4 ,5, and S. The program’s output is stored in a file called sub 1 dot out.

The greater than sign here is called redirection. It redirects the outputs from the computer screen to a file. This is necessary because if the output is shown on the screen, a person needs to sit in front of the screen and read it. If the output is saved in a file, then there is no need for a person to sit in front of the screen. The program’s output is saved in this file called sub 1 dot out.

.

The next line uses the diff command to compare two files. The first file is the newly created file that stores the program’s output. The second file is the known correct answer. If these two files have the same content, the diff command says nothing. If these two files are different, the diff command points out which lines are different.

.

The make file has the second test case using 7, 26, and. S as the arguments. The program’s output is compared with another file, called sub 2 dot correct, that stores the correct answer.

.

How to run these tests? In Bash, type make followed by test sub. This will run the commands in lines 46 to 49. The dependence of main will ensure that the executable file is built before running the tests.

.

.

It is possible to run even more tests using make file. This example shows that test all is composed of test add, test sub, test mul, and test div.

Test add tests the addition function by using A. as the last argument.

Test sub tests the subtraction function by using S. as the last argument.

In all tests, the program’s outputs are saved to files. The files are compared with known correct answers by using the diff command.

To run all tests, type make test all in bash. Make will then call test add, test sub, test mul, and test div.

Next, let’s see the bottom of this make file. If you type make clean, the machine generated files will be removed.

It is important that you do not put star dot C here. If you put star dot C here, your program will be erased.

Sometimes, you want to remove all machine generated files to clean up your directory, this command can become helpful.

.

Here is a summary of what make file can do:

.

Inside a make file, we can define symbols. These symbols can be used to turn on G C C warnings or for conditional compilations.

A make file can compile dot C files and generate object files. A make file can link object files to create an executable file.

A make file can track which dot C files have been changed and rebuilt the object files when necessary. If any object file is rebuilt, the executable file is also rebuilt.

A make file can run tests and compare the outputs with known correct answers.

As you can see, a make file can do many things for you. If you use make file properly, you can save significant amounts of time because many of these tasks are automated.