This lecture continues the explanation about make file.

Let’s see what make file do in the first homework.

This is a review from the previous lecture: make file can define symbols used for gcc and conditional compilation to select different sections of code. Make file can keep track which dot C files have been changed and their object files need to be generated. Make file can link object files to create an executable file.

To help us understand what make file can do for us, let’s start with a simpler make file.

This is a much simplified make file. In order to show this simple make file, I have removed the alias for G C C so that the warning messages are not included. We will add the warning messages back soon.

The upper left shows the make file.

The lower right shows what happens if we type the make command in bash.

Do you notice that G C C compiles the list of dot C files and creates object files?

The list of C files are exactly the same as the list of files specified by S R C S in make file.

The last line of the output matches the action for creating the executable file.

How did this happen?

In a make file, there should be at least one target. What is a target? It tells make file the purpose. A target is on the left side of a colon.

This target depends on the list of object files, specified by O B J S.

.

O B J S is defined right above.

O B J S is created by replacing dot C in S R C S. by dot O.

.

Thus, O B J S is a list of object files: add dot O, div dot O, mul dot O, main dot O, solution dot O, and sub dot O.

.

In order to create the object files, make file uses G C C minus C to compile dot C files.

Let’s review what we have learned so far:

The object files are created replacing dot C in S R C S by dot O in order to form the list of object files.

To create the object files, make file tells G C C to compile the C files.

After the object files are available, make file uses G C C again. This time there is no minus C and an executable file is created. The name of the executable file is specified after minus O.

.

Please notice that this is the lower case O, not upper case.

The upper case O means optimization for G C C and will not be used in this class.

This finishes the entire process of building the executable file.

Let’s see how make file can help us keep track of changes.

If we type make again, bash says the executable file M A I N is update to date. This occurs because we have not changed any dot C file. It is unnecessary to rebuild the executable file.

Next, we use the echo command to append an empty line at the end of add dot C and type make again.

Make detects that add dot C has been changed and compiles it again. Then, all the object files are put together to create the executable file.

Please notice that only add dot O. is regenerated because only add dot C has been changed.

The next two echo commands append empty lines in mul dot C and sub dot C. Then, we type the make command.

Since two dot C files have been changed, their object files are generated. Finally, all object files are linked together to generate the executable file.

Are you ready to see a more complex make file?

Let’s try.

We add minus S T D equals C 99 to use the C standard announced in 1999. We also add minus G in order to enable debugging.

In bash, remove all object files. If we type make now, the changes are added for every G C C command.

As you can see, the change in one place is used everywhere. You do not have to type the command by hand over and over again.

Next, we add warnings into G C C by defining a symbol. The symbol is used by adding a dollar sign in front.

When we type make, all G C C commands have the warning turned on. Again, we need to make only one change in the make file and this change is effective for every G C C command.

Our next change defines the symbol for conditional compilation. In this case, we define a symbol called test main. The C program’s main function is enclosed by I. F. D. E. F. . and E. N. D. I. F. .

If we type make, the executable will be built.

Let’s remove the test flags, remove the object files, and type make.

The executable cannot be built because the main function cannot be found. The main function is the starting point of every C program. If the symbol test main is not defined, the function enclosed by I. F. D. E. F. and E. N. D. I. F. is discarded and thus undefined.

Without the main function, the executable file cannot be generated.