Compiling Files independently and linking them together

In a real world situation, several developers work on a product creating and modifying functions and files they created or someone created for them. The normal compilation and linking we have so far being doing with the command gcc should be changed so developer can just compile , not link to the main function. When everyone completed their developing their function, we link them together to generate the main product or program.

Let us take a calculator example program having several functions such as add, subtract, and multiply. The add function

// add.c : function add

#include <stdio.h>

int add ( int x, int y )

{

printf ( “ Inside the function add \n”) ;

return x + y ;

}

and we will code this a file add.c .

As you can see, there is no main function within this file. We can compile this piece of code using the command line :

**gcc -c add.c -o add**

the command will look for compilation errors and generates an object file without linking to any other file. The –o option specifies the output file and –c option as per the man page

-c Compile or assemble the source files, but do not link. The linking stage simply is not done. The ultimate output is in the form of an object file for each source file.

We will add similar piece of code for subtract in a file subtract.c

// subtract.c : function subtract

#include <stdio.h>

int subtract ( int x, int y )

{

printf ( “ Inside the function subtract \n”) ;

return x - y ;

}

**gcc –c subtract.c -o sub**

and similarly for multiply

// multiply.c : function multiply

#include <stdio.h>

int multiply ( int x, int y )

{

printf ( “ Inside the function multiply \n”) ;

return x \* y ;

}

**gcc –c multiply.c –o multiply.o**

We have three object files with no main function. Let us create the main function

// main.c : main function

#include <stdio.h>

int main ( )

{

// let us make this very simple

int value = add ( 5, 4 );

printf ( “ 5 + 4 = %d \n”, value );

value = subtract ( 5, 4 );

printf ( “ 5 - 4 = %d \n”, value );

value = multiply ( 5, 4 );

printf ( “ 5 x 4 = %d \n”, value );

}

**gcc main.c –o calculator**

But you will get compilation errors because the compiler is trying to figure the definition of add, multiply and subtract. Because it couldn’t find, it will end with linking errors.

How do we link those functions ? There are two ways

gcc main.c add.c subtract.c multiply.c -o calculator

in this all source codes will be compiled together taking long time

or

**gcc main.c add.o subtract.o multiply.o –o calculator**

main.c is compiled and linked with already compiled code of add.o sub.o and multiply.o .

Now the compiler find the add function being defined in the object file add.o, subtract function defined in subtract.o and multiply function in multiply.o , generating the final product calculator.

So, whenever you develop a function(s) in a separate file(s), just make sure it compiles, and generate the object file. Then, you will link all of them together.

Advantage with this process of developing programs is multiple programmers will be able to develop programs simultaneously without waiting for the other to finish.

But the problem with this approach is the all source code should be compiled from the scratch and/or linking with old files when programmers have new code available.

The solution to both method is makefiles.

Makefile monitor for changes in the source code. If the timestamp of the source code of a function in a file is newer than the timestamp of the executable, it will compile only the source code of that function, it will not compile the entire source code saving considerable amount of time.

For instance, say our executable is compiled on Tuesday 5PM and the source code of add.c was modified at 6PM, makefiles compile only the add.c code and link with the rest of the code. Can you imagine the time it saves when you are amongst 10000 developers. You change one file and the entire process will take only your file, not files created by 10000 developers. What a nice piece of creativity ?

Makefiles

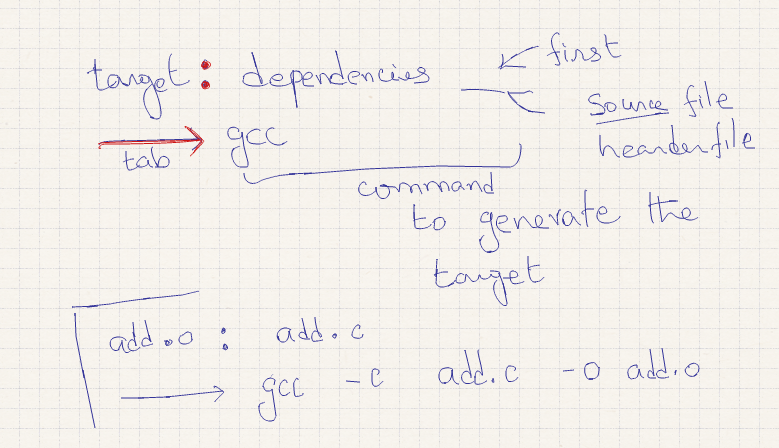
Makefile contains pair of two lines : first line specifies object file (or target) to be created followed by <colon> and followed by the dependencies the target depends. In the next line starting with a tab, then specify the commands to compile to generate a target. For instance,

add.o : add.c

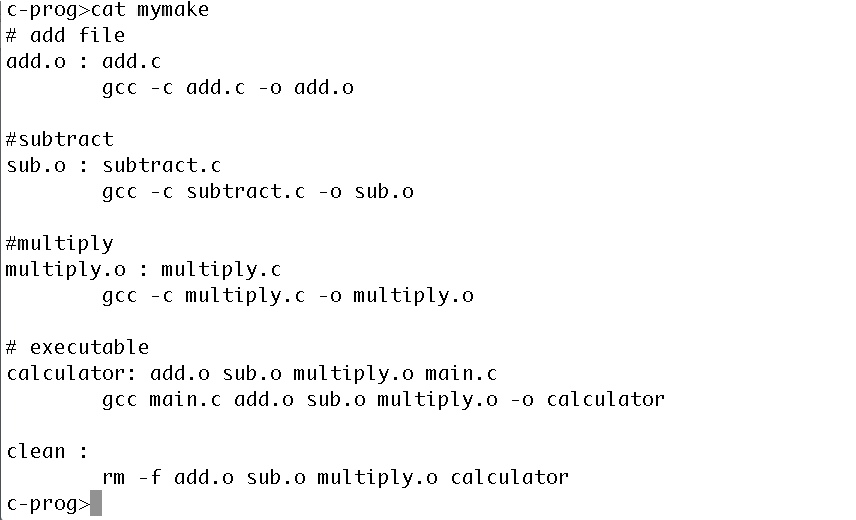
<tab>gcc –c add.c

In the above lines, add.o is the target we are tryig to generate, followed by <colon> followed by dependency. In the second line, we enter tab first followed by the command.

Here is the screen shot we did in the video



The entire makefile is



To clean and remove all object file, we run the command

make –f mymake clean

the target is clean, it run the shell command rm –f , f stands for forcefully remove without prompting or any errors.

to generate the executable calculator (the target here) , we run the command

make –f mymake calculator

to generate only an object file say add (the target here) , we run the command

make –f mymake add