Compilation

Contact : H K Sim

Email: simhkeng@yahoo.com

Compilation Process

Pre-processor

Pre-processor runs before the complier and add or manage the lines of code that will be sent for actual compilation.

Line of code that with the # (hash mark) is belongs to the pre-processor , not compiler. The common instructions (called directives) are :

#include

#define

#ifdef … #endif this Supports conditional compilation.

Compilation

The compiler take in the xxx.c file and turn it into a object file with a .o extension. When compiling with a gcc, this file is usually deleted once the compilation is done successfully. Option extension can be use to compile into object file e.g. xxx.o

Email : simhkeng@yahoo.com

Compilation Process

Linker

The linker combines the .o files and the specified libraries into a single executable file. In linux ,by default, the library file /usr/lib/libc.so is also included.

e.g If a program includes #inlcude<math.h>, the maths library (/usr/lib/libm.so) will be linked to provide the final executable program.

Email : simhkeng@yahoo.com

Make Utility

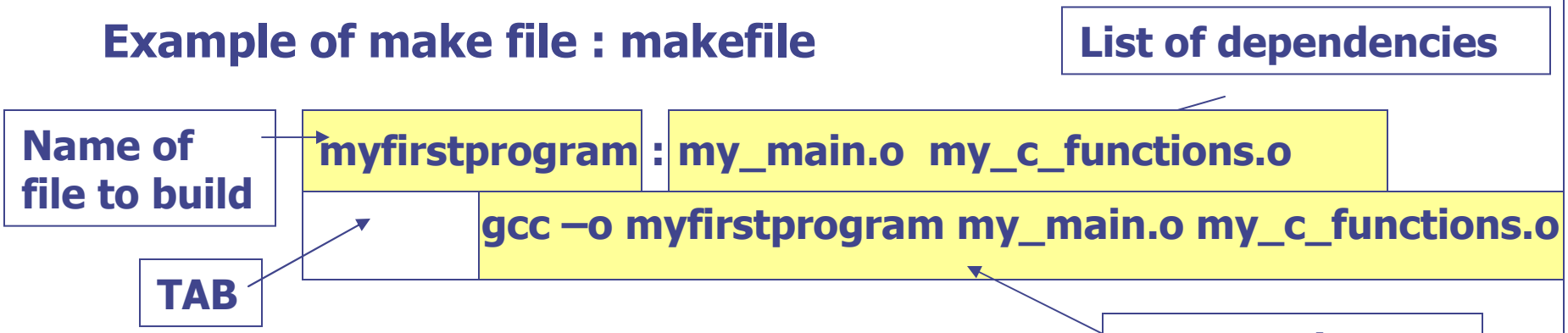
- Use in large development project involves many C code files and header files.
- The make file content must be created in the file name called “makefile” or “Makefile”.
- To run the make utilities, just type (@ command prompt) :

  …#] make.
- Make utility will look for the “makefile” or “Makefile” and execute it
- Makefile is very sensitive to the format sequence , carriage return and Tab. Tab cannot be replaced by Space.

Email : simhkeng@yahoo.com

Make Utility

Example of make file : makefile




```
my_main : my_main.c
        gcc –c my_main.c
my_c_function : myfunction.c
        gcc –c my_function.c
.PHONY : clean
 clean : rm *.o
```

For command that longer than one line, the next line can be continued using a "\"

Email : simhkeng@yahoo.com

Compilation extension

Common gcc compiler's interpretation of extension

-o filename - output to a file call "filename" If not specified, default is a.out

-c - compile without linking

-Idirname - specific directory that gcc will search for include file

-Ldirname - specific directory that gcc will search for library file

-static - link the static library

-lmylib - link the mylib library

-g - include standard debugging information

-O - optimize the compiled code

-W - suppress all warning message

-Wall - display all the warning message that gcc can provide

-v - show the commands use in each step

Email : simhkeng@yahoo.com

Preprocessor Extension

Compile only preprocessing code

Compile only preprocessing code and save it in the file named tempfile

gcc **–E** my_main.c **–o tempfile**

-E – means compile only preprocessor code

-o output file

-o tempfile means output file to a file called “tempfile”

Email : simhkeng@yahoo.com

Multi-file compile and link Example

main.c

```
#include<stdio.h>
#include "myheader.h"

int main(void)
{
int i = 10, k = 50;
int m, result;

m = add (i,k);
result = multiply( m, gInt);

printf("Final Result =
%d\n",result);

return 0;
}
```

myheader.h

```
int add ( int x, int y);
int multiply ( int x, int y);
int gInt = 100;  // declare global
integer
```

func1.c

```
#include<stdio.h>
int add( int x, int y)
{
printf("Print from func1.c->(x+y) =
%d\n",(x+y));
return (x+y);
}
```

func2.c

```
#include<stdio.h>
int multiply(int x, int y)
{
printf("Print from func2.c->(x*y) =
%d\n",(x*y));
return (x*y);
}
```

Compile , link and create a executable file called multi_files

Email : simhkeng@yahoo.com

Multi-file compile and link Example

Compile and "link" by hand

```
gcc –c main.c –o main.o
gcc –c func1.c –o func1.o
gcc –c func2.c –o func2.o
gcc main.o func1.o func2.o –o multi_files
```

Using makefile

```
multi_files: main.o func1.o func2.o
                    gcc -o multi_files   main.o    func1.o    func2.o
main.o :main.c
          gcc -c main.c
func1.o :func1.c
          gcc -c func1.c
func2.o :func2.c
          gcc -c func2.c

.PHONY:          clean
clean :
          rm -f *.o
```


Email : simhkeng@yahoo.com