EXPERIMENT 10

Aim: Introduction to GCC compiler: Basics of GCC, Compilation of program, Execution of program, Time stamping

Solution:

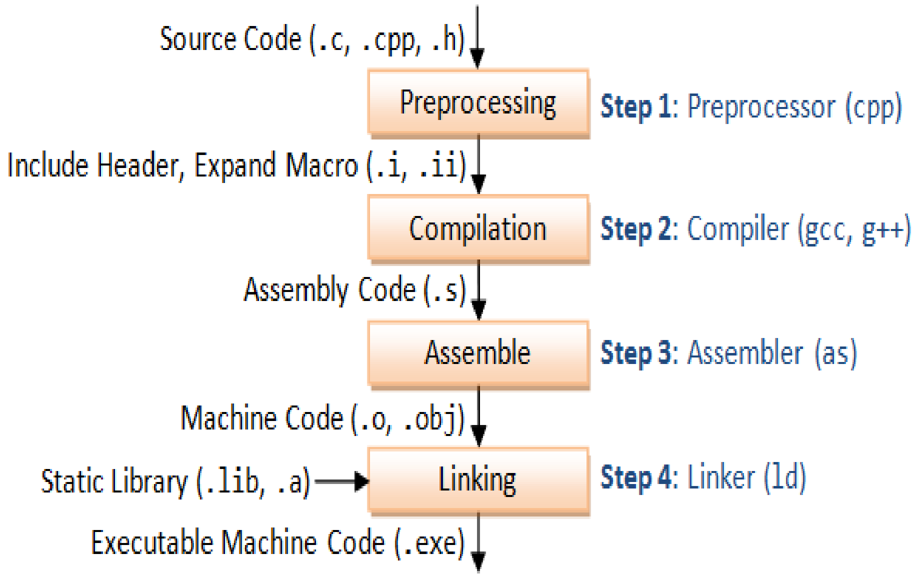
gcc command in Linux with examples

GCC stands for GNU Compiler Collections which is used to compile mainly C and C++ language. It can also be used to compile Objective C and Objective C++. The most important option required while compiling a source code file is the name of the source program, rest every argument is optional like a warning, debugging, linking libraries, object file etc. The different options of *gcc* command allow the user to stop the compilation process at different stages.

Example: This will compile the *source.c* file and give the output file as *a.out* file which is default name of output file given by gcc compiler, which can be executed using **./a.out**

```
gcc source.c
```

Compilation Steps


Source Code (.c, .cpp, .h)
Preprocessing
Step 1: Preprocessor (cpp)
Include Header, Expand Macro (.i, .ii)
Compilation
Step 2: Compiler (gcc, g++)
Assembly Code (.s)
Assemble
Step 3: Assembler (as)
Machine Code (.o, .obj)
Static Library (.lib, .a)
Linking
Step 4: Linker (ld)
Executable Machine Code (.exe)

GCC compiles a C/C++ program into executable in 4 steps as shown in the above diagram. For example, a "`gcc -o hello.exe hello.c`" is carried out as follows:

1. Pre-processing: via the GNU C Preprocessor (`cpp.exe`), which includes the headers (`#include`) and expands the macros (`#define`).

   ```
   > cpp hello.c > hello.i
   ```

   The resultant intermediate file "`hello.i`" contains the expanded source code.
2. Compilation: The compiler compiles the pre-processed source code into assembly code for a specific processor.

   ```
   > gcc -S hello.i
   ```

   The `-S` option specifies to produce assembly code, instead of object code. The resultant assembly file is "`hello.s`".
3. Assembly: The assembler (`as.exe`) converts the assembly code into machine code in the object file "`hello.o`".

   ```
   > as -o hello.o hello.s
   ```
4. Linker: Finally, the linker (`ld.exe`) links the object code with the library code to produce an executable file "`hello.exe`".

   ```
   > ld -o hello.exe hello.o ...libraries...
   ```

Verbose Mode (-v)

You can see the detailed compilation process by enabling `-v` (verbose) option. For example,

```
> gcc -v -o hello.exe hello.c
```

Defining Macro (-D)

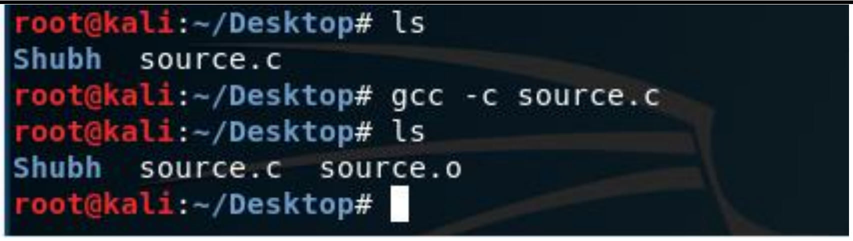
You can use the `-D`*name* option to define a macro, or `-D`*name*=*value* to define a macro with a value. The *value* should be enclosed in double quotes if it contains spaces.

Options in gcc

-Wall: This will check not only for errors but also for all kinds warning like unused variables errors, it is good practice to use this flag while compiling the code.

```
gcc source.c -Wall -o opt
```

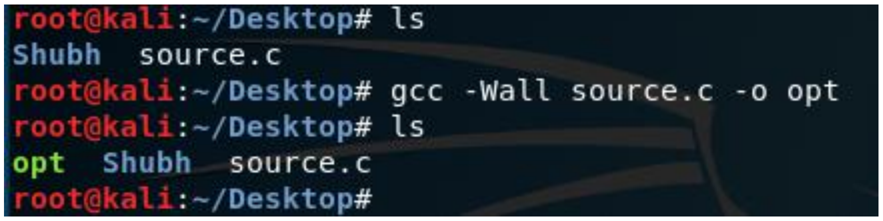
-c : This command compile the program and give the object file as output, which is used to make libraries.



```
root@kali:~/Desktop# ls
Shubh  source.c
root@kali:~/Desktop# gcc -c source.c
root@kali:~/Desktop# ls
Shubh  source.c  source.o
root@kali:~/Desktop# 
```

-o opt: This will compile the source.c file but instead of giving default name hence executed using **./opt**, it will give output file as opt. -*o* is for output file option.

```
gcc source.c -o opt
```



```
root@kali:~/Desktop# ls
Shubh  source.c
root@kali:~/Desktop# gcc -Wall source.c -o opt
root@kali:~/Desktop# ls
opt  Shubh  source.c
root@kali:~/Desktop#
```

Compiling a Program

```
g++ demo2.C -o demo2

## or use the following syntax ##

make demo2
```

To run this program, type:

```
./demo2
```

Creating object files from source files using compile only option -c

$ gcc -Wall -c main.c

Creating executables from object files and linking them together

$ gcc main.o hello_fn.o -o hello

The resulting executable file can now be run:

$ ./hello

Link order of object files.

```
$ gcc main.o hello_fn.o -o hello (correct order)
$ gcc hello_fn.o main.o -o hello (incorrect order)
```

Linking with external libraries

```
#include <math.h>
#include <stdio.h>
void main (void){
double x = sqrt (2.0);
printf ("The square root of 2.0 is %f\n", x);
}
$ gcc -Wall calc.c /usr/lib/libm.a -o calc
$ gcc -Wall calc.c -lm -o calc
$ gcc -Wall -lm calc.c -o calc (incorrect order)
$ gcc -Wall calc.c -lm -o calc (correct order)
```

Compiling with optimization

```
$ gcc -Wall -O0 test.c –lm
```

Timestamp

A timestamp is the current time of an event that is recorded by a computer.Timestamps are employed extensively within computers and over networks for various types of synchronization.

```
$ time ./a.out
  real 0m13.388s
  user 0m13.370s
  sys 0m0.010s
```

Getting Started

The GNU C and C++ compiler are called gcc and g++, respectively.

Compile/Link a Simple C Program - hello.c

Below is the Hello-world C program hello.c:

```
1// hello.c
```

```
#include <stdio.h>

int main() {
   printf("Hello, world!\n");
   return 0;
}
```

To compile the hello.c:

```
> gcc hello.c
  // Compile and link source file hello.c into executable a.exe (Windows) or a (Unixes)
```

The default output executable is called "a.exe" (Windows) or "a.out" (Unixes and Mac OS X).

To run the program:

```
// (Windows) In CMD shell
> a

// (Unixes / Mac OS X) In Bash Shell - include the current path (./)
$ chmod a+x a.out
$ ./a.out
```

Notes for Unixes and Bash Shell:

- In Bash shell, the default PATH does not include the current working directory. Hence, you need to include the current path (./) in the command. (Windows include the current directory in the PATH automatically; whereas Unixes do not - you need to include the current directory explicitly in the PATH.)
- You also need to include the file extension, if any, i.e., "./a.out".
- In Unixes, the output file could be "a.out" or simply "a". Furthermore, you need to assign *executable file-mode* (*x*) to the executable file "a.out", via command "chmod a+x *filename*" (add executable file-mode "+x" to all users "a+x").

To specify the output filename, use -o option:

```
// (Windows) In CMD shell
> gcc -o hello.exe hello.c
  // Compile and link source file hello.c into executable hello.exe
> hello
  // Execute hello.exe under CMD shell

// (Unixes / Mac OS X) In Bash shell
$ gcc -o hello hello.c
$ chmod a+x hello
$ ./hello
```

NOTE for Unixes:

- In Unixes, we typically omit the .exe file extension (meant for Windows only), and simply name the output executable as hello (via command "gcc -o hello hello.c".
- You need to assign executable file mode via command "chmod a+x hello".

Compile/Link a Simple C++ Program - hello.cpp

```
// hello.cpp
#include <iostream>
using namespace std;

int main() {
   cout << "Hello, world!" << endl;
   return 0;
}
```

You need to use g++ to compile C++ program, as follows. We use the -o option to specify the output file name.

```
// (Windows) In CMD shell
> g++ -o hello.exe hello.cpp
  // Compile and link source hello.cpp into executable hello.exe
> hello
  // Execute under CMD shell

// (Unixes / Mac OS X) In Bash shell
$ g++ -o hello hello.cpp
$ chmod a+x hello
$ ./hello
```

More GCC Compiler Options

A few commonly-used GCC compiler options are:

```
$ g++ -Wall -g -o Hello.exe Hello.cpp
```

- -o: specifies the output executable filename.
- -Wall: prints "all" Warning messages.
- -g: generates additional symbolic debuggging information for use with gdb debugger.

Compile and Link Separately

The above command *compile* the source file into object file and link with other object files and system libraries into executable in one step. You may separate compile and link in two steps as follows:

```
// Compile-only with -c option
> g++ -c -Wall -g Hello.cpp
// Link object file(s) into an executable
```

```
> g++ -g -o Hello.exe Hello.o
```

The options are:

- -c: Compile into object file "Hello.o". By default, the object file has the same name as the source file with extension of ".o" (there is no need to specify -o option). No linking with other object files or libraries.
- Linking is performed when the input file are object files ".o" (instead of source file ".cpp" or ".c"). GCC uses a separate linker program (called ld.exe) to perform the linking.

Compile and Link Multiple Source Files

Suppose that your program has two source files: file1.cpp, file2.cpp. You could compile all of them in a single command:

```
> g++ -o myprog.exe file1.cpp file2.cpp
```

However, we usually compile each of the source files separately into object file, and link them together in the later stage. In this case, changes in one file does not require re-compilation of the other files.

```
> g++ -c file1.cpp
> g++ -c file2.cpp
> g++ -o myprog.exe file1.o file2.o
```

Compile into a Shared Library

To compile and link C/C++ program into a shared library (".dll" in Windows, ".so" in Unixes), use -shared option. Read "Java Native Interface" for example.

Viva Questions:

- What do you mean by timestamp?
- Write a command to compile the file file1
- Write a command to execute the file file1
- Create the source file from object files
- Why do you need a compiler?
- How will you change the name of the object file being created at the time of compilation?
- What do you mean by assembling?
- What are linkers?
- What are loaders?

Assignment:

Write a program to add two numbers and compile and execute using gcc compiler.

Learning Outcome:

- **Debugging with gcc compiler**
- **Time-stamping with gcc**
- **Compiling object files with gcc**