Module 2: GNU Tools and Compilation Process

Introduction to GCC and History

The original GNU C Compiler is developed by Richard Stallman in 1984 to create a complete UNIX like operating systems as free software. GCC is referred as "GNU C Compiler" to support many programming languages such as objective C, C++, Java, FORTRAN and Ada. GCC is portable and run in many operating systems. It is available on all UNIX operating systems and ported to windows by Cygwin and MinGW. GCC is also a cross compiler for producing executables for different platforms.

GCC is a key component of "GNU Tool chain" for developing applications as well as operating systems. The GNU Tool chain includes:

- GNU Compiler Collection (GCC): A compiler tool that supports many programming languages like C/C++, Java and etc...
- GNU Make: An automation tool for compiling and building applications.
- GNU Binutils: A tool of binary utility tools which include linker and assembler.
- GNU Debugger (GDB).
- GNU Autotools: A build system including Autoconf, Autoheader, Automake and Libtools.
- GNU Bison.

Types of GCC versions:

- The first version of GCC is released in 1987.
- The second version of GCC is released in 1992 which supports C++.
- The third version is released in 2001 which incorporates Experimental GNU compiler System with improve optimization.
- The fourth version is released in 2005.
- The fifth version is released in 2015.

Installing GCC:

GCC is included by default in all UNIX operating systems and for windows install Cygwin GCC or MinGW GCC.

Cygwin: Cygwin is a UNIX like environment and command line interface for windows. Cygwin is a huge and includes most of the UNIX tools, utilities and Bash shell.

MinGW GCC: Minimalist GNU for Windows is the collection of GNU Compiler Collection and GNU Banalities for use in windows. It also includes the Minimal System, which is basically a Bourne shell.

In Linux Environment:

- gcc –v option → which displays the gcc version using in our computer.

```
bala@ubuntu:~$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/4.6/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu/Linaro 4.6.3-1ubu
ntu5' --with-bugurl=file:///usr/share/doc/gcc-4.6/README.Bugs --enable-languages
=c,c++,fortran,objc,obj-c++ --prefix=/usr --program-suffix=-4.6 --enable-shared
--enable-linker-build-id --with-system-zlib --libexecdir=/usr/lib --without-incl
uded-gettext --enable-threads=posix --with-gxx-include-dir=/usr/include/c++/4.6
--libdir=/usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --enable-li
bstdcxx-debug --enable-libstdcxx-time=yes --enable-gnu-unique-object --enable-pl
ugin --enable-objc-gc --disable-werror --with-arch-32=i686 --with-tune=generic -
-enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --targ
et=x86_64-linux-gnu
Thread model: posix
gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5)
bala@ubuntu:~$
```

In windows Environment:

Cygwin:

- gcc --version

gcc (GCC) 4.6.3

MinGW:

- gcc --version

gcc (GCC) 4.6.3

Help: We can read more information about GCC using help manual and man pages that is using 'help' and 'man' option.

- gcc –help

```
bala@ubuntu:~$ gcc --help
Usage: gcc [options] file...
Options:
  -pass-exit-codes         Exit with highest error code from a phase
  --help                   Display this information
  --target-help            Display target specific command line options
  --help={target|optimizers|warnings|params|[^]{joined|separate|undocumented}}[,...]
                           Display specific types of command line options
  (Use '-v --help' to display command line options of sub-processes)
  --version                Display compiler version information
  -dumpspecs               Display all of the built in spec strings
  -dumpversion             Display the version of the compiler
  -dumpmachine             Display the compiler's target processor
  -print-search-dirs       Display the directories in the compiler's search path
  -print-libgcc-file-name  Display the name of the compiler's companion library
  -print-file-name=<lib>   Display the full path to library <lib>
  -print-prog-name=<prog>  Display the full path to compiler component <prog>
  -print-multi-directory   Display the root directory for versions of libgcc
  -print-multi-lib         Display the mapping between command line options and
                           multiple library search directories
  -print-multi-os-directory Display the relative path to OS libraries
  -print-sysroot           Display the target libraries directory
  -print-sysroot-headers-suffix Display the sysroot suffix used to find headers
  -Wa,<options>            Pass comma-separated <options> on to the assembler
  -Wp,<options>            Pass comma-separated <options> on to the preprocessor
  -Wl,<options>            Pass comma-separated <options> on to the linker
  -Xassembler <arg>        Pass <arg> on to the assembler
  -Xpreprocessor <arg>     Pass <arg> on to the preprocessor
  -Xlinker <arg>           Pass <arg> on to the linker
  -save-temps              Do not delete intermediate files
  -save-temps=<arg>        Do not delete intermediate files
  -no-canonical-prefixes   Do not canonicalize paths when building relative
                           prefixes to other gcc components
  -pipe                    Use pipes rather than intermediate files
  -time                    Time the execution of each subprocess
  -specs=<file>            Override built-in specs with the contents of <file>
  -std=<standard>          Assume that the input sources are for <standard>
  -isysroot <directory>    Use <directory> as the root directory for headers
                           and libraries
  -B <directory>           Add <directory> to the compiler's search paths
  -v                       Display the programs invoked by the compiler
  -###                     Like -v but options quoted and commands not executed
  -E                       Preprocess only; do not compile, assemble or link
  -S                       Compile only; do not assemble or link
  -c                       Compile and assemble, but do not link
  -o <file>                Place the output into <file>
  -x <language>            Specify the language of the following input files
                           Permissible languages include: c c++ assembler none
```

Text Editor

- man gcc

```
GCC(1)                                GNU                                GCC(1)

NAME
       gcc - GNU project C and C++ compiler

SYNOPSIS
       gcc [-c|-S|-E] [-std=standard]
           [-g] [-pg] [-Olevel]
           [-Wwarn...] [-pedantic]
           [-Idir...] [-Ldir...]
           [-Dmacro[=defn]...] [-Umacro]
           [-foption...] [-mmachine-option...]
           [-o outfile] [@file] infile...

       Only the most useful options are listed here; see below for the
       remainder.  g++ accepts mostly the same options as gcc.

DESCRIPTION
       When you invoke GCC, it normally does preprocessing, compilation,
       assembly and linking.  The "overall options" allow you to stop this
       process at an intermediate stage.  For example, the -c option says not
       to run the linker.  Then the output consists of object files output by
       the assembler.

       Other options are passed on to one stage of processing.  Some options
       control the preprocessor and others the compiler itself.  Yet other
       options control the assembler and linker; most of these are not
       documented here, since you rarely need to use any of them.

       Most of the command line options that you can use with GCC are useful
       for C programs; when an option is only useful with another language
       (usually C++), the explanation says so explicitly.  If the description
       for a particular option does not mention a source language, you can use
       that option with all supported languages.

       The gcc program accepts options and file names as operands.  Many
       options have multi-letter names; therefore multiple single-letter
       options may not be grouped: -dv is very different from -d -v.

       You can mix options and other arguments.  For the most part, the order
       you use doesn't matter.  Order does matter when you use several options
       of the same kind; for example, if you specify -L more than once, the
       directories are searched in the order specified.  Also, the placement
       of the -l option is significant.

       Many options have long names starting with -f or with -W---for example,
       -fmove-loop-invariants, -Wformat and so on.  Most of these have both
 Manual page gcc(1) line 1 (press h for help or q to quit)
```

Reading man pages using command line will be difficult, so we can generate a text file using command:

- man gcc | col –b >gcc_help.txt -- it will generate a gcc_help text file in your current directory.

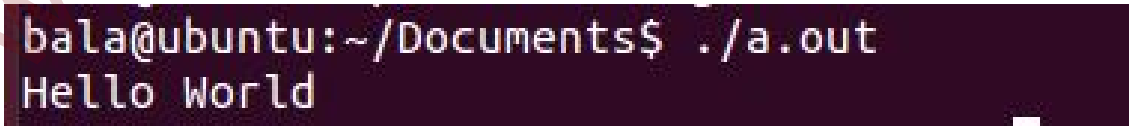
Getting started

Now we will write small code and compile using gcc compilers. Now onwards we are going to use Linux environment only. The GNU C and C++ compilers are gcc and g++ respectively. The below is the simple C code to print “Hello World”.

1. The C program is Hello.c

2. Compile Hello.c using gcc compiler.
   - gcc Hello.c

3. Run the executable generated by above program. The default executable generated in Linux environment is “a.out”.
   - ./a.out → displays the output “Hello World” in terminal.



In bash shell, the default path does not include the current working directory. Hence we need to include the current path using “./” in the command line, when executing the output file in the command line. Instead of depending default name for executable we can assign our own name

while compiling using, '-o' option in the command. '-o' is "output file" which holds the output data.

- gcc Hello.c -o Hello → This will compile the source code "Hello.c" and generate the executable as "Hello". Now we can run the "Hello" executable.

```
bala@ubuntu:~/Documents$ gcc Hello.c -o Hello
bala@ubuntu:~/Documents$ ls
a.out  Hello  Hello.c
```

- "./Hello"

```
bala@ubuntu:~/Documents$ ./Hello
Hello World
bala@ubuntu:~/Documents$
```

Verbose Mode: By using option '-v' we can see the detailed compilation process.

- gcc –v Hello.c –o Hello – It will provide the compilation process on the display, so that we can see what is going on while compiling the source code like which gcc version they using, target system, shared libraries and etc.

GCC Compilation Process:

GCC compiler compiles a program into executable in four steps. GCC provided a provision so that we can compile the source file step by step compilation.

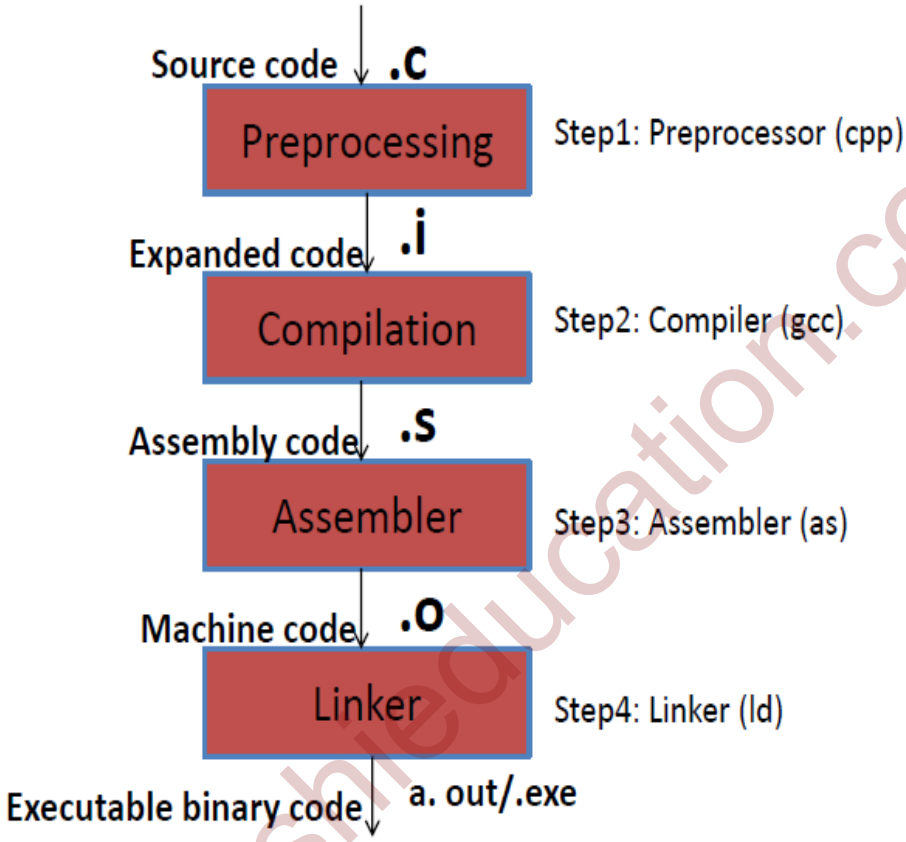


Figure1: Step by step compilation process

In the next section we will take one example program and compile them in each step and see the output file generated at each step. The following is the example source code addition of two numbers. In next section we are going to compile the source code step by step compilation process and we will observe how the source code is converted into executable followed by each step.

//add.c

```c
#include <stdio.h>

int main()
{
        int a=7, b=7, sum=0;
        sum=a+b;
        printf("sum is = %d\n", sum);
        return 0;

}
~
~
~
```

Step by Step Compilation:

1. **Pre-processing:** The first step of the compilation process is the source code is passed through Preprocessor, which includes the headers files (#include) and expands the macros (#define) used in our program. The resultant intermediate file is "add.i" which contains the expanded source code. The preprocessor takes the source code "add.c" and expands this code and gives "add.i" and this "add.i" is passed to compiler for further compilation.
   - gcc –E add.c –o add.i

   -E → invokes the preprocessor.

```
bala@ubuntu:~/Documents$ gcc -E add.c -o add.i
bala@ubuntu:~/Documents$ ls
add.c  add.i  a.out  Hello  Hello.c
bala@ubuntu:~/Documents$
```

   - gcc –E –v add.c –o add.i → verbose mode – Displays the information of gcc compiler, machine architecture and so on.

```
bala@ubuntu:~/Documents$ gcc -E -v add.c -o add.i
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/4.6/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu/Linaro 4.6.3-1ubu
ntu5' --with-bugurl=file:///usr/share/doc/gcc-4.6/README.Bugs --enable-languages
=c,c++,fortran,objc,obj-c++ --prefix=/usr --program-suffix=-4.6 --enable-shared
--enable-linker-build-id --with-system-zlib --libexecdir=/usr/lib --without-incl
uded-gettext --enable-threads=posix --with-gxx-include-dir=/usr/include/c++/4.6
--libdir=/usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --enable-li
bstdcxx-debug --enable-libstdcxx-time=yes --enable-gnu-unique-object --enable-pl
ugin --enable-objc-gc --disable-werror --with-arch-32=i686 --with-tune=generic -
-enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --targ
et=x86_64-linux-gnu
Thread model: posix
gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5)
COLLECT_GCC_OPTIONS='-E' '-v' '-o' 'add.i' '-mtune=generic' '-march=x86-64'
 /usr/lib/gcc/x86_64-linux-gnu/4.6/cc1 -E -quiet -v -imultilib . -imultiarch x86
_64-linux-gnu add.c -o add.i -mtune=generic -march=x86-64 -fstack-protector
ignoring nonexistent directory "/usr/local/include/x86_64-linux-gnu"
ignoring nonexistent directory "/usr/lib/gcc/x86_64-linux-gnu/4.6/../../../../x8
6_64-linux-gnu/include"
#include "..." search starts here:
#include <...> search starts here:
 /usr/lib/gcc/x86_64-linux-gnu/4.6/include
 /usr/local/include
 /usr/lib/gcc/x86_64-linux-gnu/4.6/include-fixed
 /usr/include/x86_64-linux-gnu
 /usr/include
End of search list.
COMPILER_PATH=/usr/lib/gcc/x86_64-linux-gnu/4.6/:/usr/lib/gcc/x86_64-linux-gnu/4
.6/:/usr/lib/gcc/x86_64-linux-gnu/:/usr/lib/gcc/x86_64-linux-gnu/4.6/:/usr/lib/g
cc/x86_64-linux-gnu/
LIBRARY_PATH=/usr/lib/gcc/x86_64-linux-gnu/4.6/:/usr/lib/gcc/x86_64-linux-gnu/4.
6/../../../x86_64-linux-gnu/:/usr/lib/gcc/x86_64-linux-gnu/4.6/../../../../lib/:
/lib/x86_64-linux-gnu/:/lib/../lib/:/usr/lib/x86_64-linux-gnu/:/usr/lib/../lib/:
/usr/lib/gcc/x86_64-linux-gnu/4.6/../../../:/lib/:/usr/lib/
COLLECT_GCC_OPTIONS='-E' '-v' '-o' 'add.i' '-mtune=generic' '-march=x86-64'
bala@ubuntu:~/Documents$
```

2. **Compilation:** The compiler compiles the preprocessed source code into assembly code for specific processor.

   The '-s' option specifies to produce assembly code, instead of object code. The resultant assembly file is "add.s"

   - gcc –S add.i –o add.s

```
bala@ubuntu:~/Documents$ gcc -S add.i -o add.s
bala@ubuntu:~/Documents$ ls
add.c  add.i  add.s  a.out  Hello  Hello.c
bala@ubuntu:~/Documents$
```

   -s → invokes the compiler.

//add.s -- assembly file

```
        .file   "add.c"
        .section        .rodata
.LC0:
        .string "sum is = %d\n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $7, -12(%rbp)
        movl    $7, -8(%rbp)
        movl    $0, -4(%rbp)
        movl    -8(%rbp), %eax
        movl    -12(%rbp), %edx
        addl    %edx, %eax
        movl    %eax, -4(%rbp)
        movl    $.LC0, %eax
        movl    -4(%rbp), %edx
        movl    %edx, %esi
        movq    %rax, %rdi
        movl    $0, %eax
        call    printf
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
        .section        .note.GNU-stack,"",@progbits
~
~
~
~
~
~
```

- gcc –S–v add.c –o add.i → verbose mode

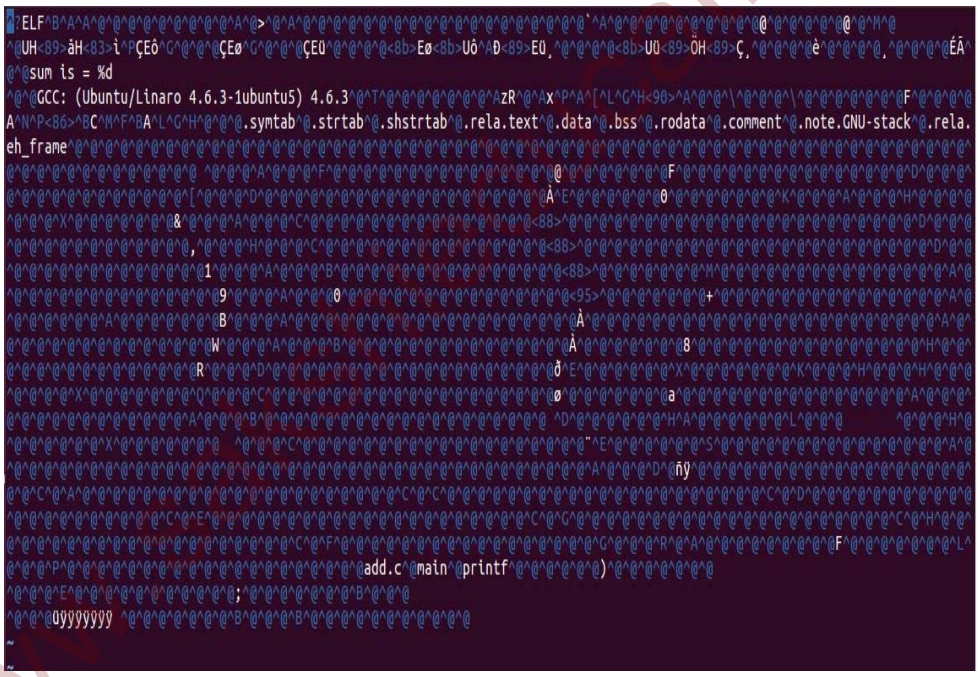
```
bala@ubuntu:~/Documents$ gcc -S -v add.i -o add.s
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/4.6/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu/Linaro 4.6.3-1ubuntu5' --with-bugurl=file:///usr/share/doc/gcc-4
.6/README.Bugs --enable-languages=c,c++,fortran,objc,obj-c++ --prefix=/usr --program-suffix=-4.6 --enable-shared --enable-linke
r-build-id --with-system-zlib --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --with-gxx-include-dir=/u
sr/include/c++/4.6 --libdir=/usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --enable-libstdcxx-debug --enable-libst
dcxx-time=yes --enable-gnu-unique-object --enable-plugin --enable-objc-gc --disable-werror --with-arch-32=i686 --with-tune=gene
ric --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5)
COLLECT_GCC_OPTIONS='-S' '-v' '-o' 'add.s' '-mtune=generic' '-march=x86-64'
 /usr/lib/gcc/x86_64-linux-gnu/4.6/cc1 -fpreprocessed add.i -quiet -dumpbase add.i -mtune=generic -march=x86-64 -auxbase-strip
add.s -version -o add.s -fstack-protector
GNU C (Ubuntu/Linaro 4.6.3-1ubuntu5) version 4.6.3 (x86_64-linux-gnu)
        compiled by GNU C version 4.6.3, GMP version 5.0.2, MPFR version 3.1.0-p3, MPC version 0.9
GGC heuristics: --param ggc-min-expand=97 --param ggc-min-heapsize=126215
GNU C (Ubuntu/Linaro 4.6.3-1ubuntu5) version 4.6.3 (x86_64-linux-gnu)
        compiled by GNU C version 4.6.3, GMP version 5.0.2, MPFR version 3.1.0-p3, MPC version 0.9
GGC heuristics: --param ggc-min-expand=97 --param ggc-min-heapsize=126215
Compiler executable checksum: 75e879ed14f91af504f4150eadeaa0e6
COMPILER_PATH=/usr/lib/gcc/x86_64-linux-gnu/4.6/:/usr/lib/gcc/x86_64-linux-gnu/4.6/:/usr/lib/gcc/x86_64-linux-gnu/:/usr/lib/gcc
/x86_64-linux-gnu/4.6/:/usr/lib/gcc/x86_64-linux-gnu/
LIBRARY_PATH=/usr/lib/gcc/x86_64-linux-gnu/4.6/:/usr/lib/gcc/x86_64-linux-gnu/4.6/../../../x86_64-linux-gnu/:/usr/lib/gcc/x86_6
4-linux-gnu/4.6/../../../../lib/:/lib/x86_64-linux-gnu/:/lib/../lib/:/usr/lib/x86_64-linux-gnu/:/usr/lib/../lib/:/usr/lib/gcc/x
86_64-linux-gnu/4.6/../../../:/lib/:/usr/lib/
COLLECT_GCC_OPTIONS='-S' '-v' '-o' 'add.s' '-mtune=generic' '-march=x86-64'
bala@ubuntu:~/Documents$
```

3. **Assembly:** The assembler converts the assembly code into machine code in the object file "add.o". The object file is a binary file which is understands by machine.
   - gcc –c add.s –o add.o

   -c → invokes the assembler.

```
bala@ubuntu:~/Documents$ gcc -c add.s -o add.o
bala@ubuntu:~/Documents$ ls
add.c  add.i  add.o  add.s  a.out  Hello  Hello.c
bala@ubuntu:~/Documents$
```

//add.o – object file



4. **Linker:** The linker links the object code with library code to produce an executable file "add".
   - gcc add.o –o add
   - gcc –v add.0 add

```
bala@ubuntu:~/Documents$ gcc add.o -o add
bala@ubuntu:~/Documents$ ls
add  add.c  add.i  add.o  add.s  a.out  Hello  Hello.c
bala@ubuntu:~/Documents$
```

- “add” is the output executable file generated by the compiler. By running this executable the output will be displayed on the console.

```
bala@bala-virtual-machine:~/Documents$ ./add
sum is = 14
bala@bala-virtual-machine:~/Documents$
```