


# C CPP ON UNIX

Lesson 1: Compiling C and CPP Programming on Unix

January 19, 2018

Proprietary and Confidential

< 1 >



Capgemini  
ENJOY THE TECHNOLOGY EXPERIENCE

## Lesson Objectives

➤ **In this lesson, you will learn about:**

- Compiling C Programs using gcc compiler
- Using static and dynamic link library
- Compiling C++ Programs using g++ compiler



January 19, 2016 Proprietary and Confidential < 2 >



### Lesson Objectives:

This lesson introduces you to program development in Unix/Linux Platform

Lesson 1: Compiling C and CPP programs using gcc and g++ compiler


1.1: Compiling C Programs Using gcc Compiler

## Introduction to gcc Compiler

- gcc compiler is founded by Richard Stallman
- gcc stands for GNU Compiler Collection
  - As it provides support for languages like C, C++ (g++), Java (gcj), gnu smalltalk (gst) etc
- Usage:

`gcc [options] inputfilename  
Eg: gcc -o outputfile inputfile`
- When you invoke gcc, it normally does preprocessing, compilation, assembly and linking

January 19, 2016 Proprietary and Confidential 3



### GCC Compiler: Introduction

The original author of the GNU C Compiler (GCC) is Richard Stallman, founder of the GNU Project. The GNU Project was started in 1984 to create a complete Unix-like operating system as free software, in order to promote freedom and cooperation among computer users and programmers. Every Unix-like operating system needs a C compiler, and as there were no free compilers in existence at that time, the GNU Project had to develop one from scratch.

The first release of GCC was made in 1987. This was a significant breakthrough, being the first portable ANSI C optimizing compiler released as free software. Since that time GCC has become one of the most important tools in the development of free software.

A major revision of the compiler came with the 2.0 series in 1992, which added the ability to compile C++. In 1997, an experimental branch of the compiler (EGCS) was created, to improve optimization and C++ support. Following this work, EGCS was adopted as the new main-line of GCC development, and these features became widely available in the 3.0 release of GCC in 2001. Over time GCC has been extended to support many additional languages, including Fortran, AD A, Java and Objective-C. The acronym GCC is now used to refer to the “GNU Compiler Collection”.


1.1: Compiling C Programs Using gcc Compiler

## Features of gcc

➤ **gcc compiler:**

- Is a portable compiler running on many platforms
- Is a cross compiler, producing executable files for a different system from the one used by gcc
- Supports multiple programming languages
- Has a modular design, allowing support for new languages and architectures
- Is a free software, which:
  - Is distributed under GNU General Public License(GNU GPL)
  - Allows support for new languages and architectures to be added

January 19, 2016 Proprietary and Confidential > 4 <

 Capgemini  
ENVIRONNEMENT TECHNOLOGIQUE

### GCC Compiler: Features

First of all, gcc is a portable compiler — it runs on most platforms available today, and can produce output for many types of processors.

gcc is not only a native compiler — it can also cross-compile any program, producing executable files for a different system from the one used by gcc itself. This allows software to be compiled for embedded systems which are not capable of running a compiler. gcc is written in C with a strong focus on portability, and can compile itself, so it can be adapted to new systems easily.

gcc has multiple language frontends, for parsing different languages. Programs in each language can be compiled, or cross-compiled, for any architecture. For example, an ADA program can be compiled for a microcontroller, or a C program for a supercomputer.

gcc has a modular design, allowing support for new languages and architectures to be added. Adding a new language front-end to gcc enables the use of that language on any architecture, provided that the necessary run-time facilities (such as libraries) are available. Similarly, adding support for a new architecture makes it available to all languages.

Finally, and most importantly, gcc is free software, distributed under the GNU General Public License (GNU GPL). This means you have the freedom to use and to modify gcc, as with all GNU software. If you need support for a new type of CPU, a new language, or a new feature you can add it yourself, or hire someone to enhance gcc for you. You can hire someone to fix a bug if it is important for your work. Furthermore, you have the freedom to share any enhancements you make to gcc. As a result of this freedom you can also make use of enhancements to gcc developed by others. The many features offered by gcc today show how this freedom to cooperate works to benefit you, and everyone else who uses gcc.

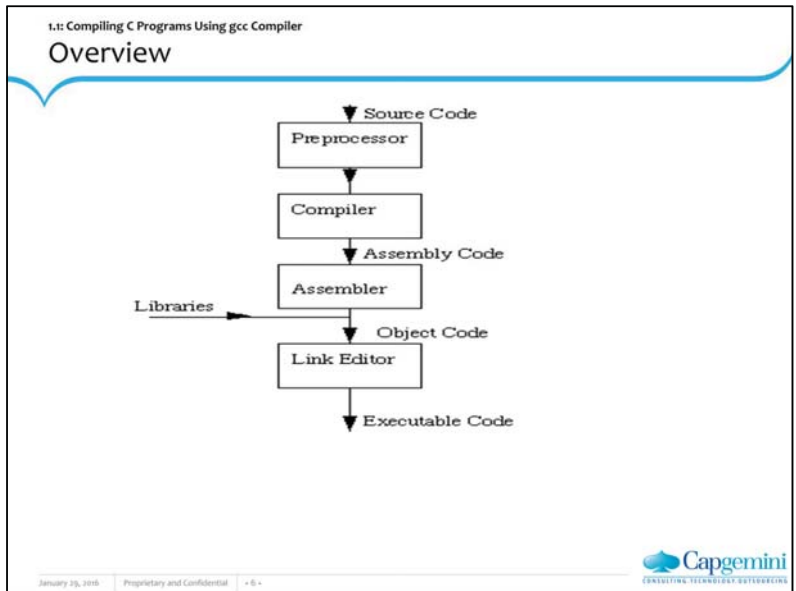
1.1: Compiling C Programs Using gcc Compiler

## Definition

- **Compilation refers to the process of converting a program written in high level programming language such as C or C++, into machine code**
- **Programs can be compiled from a single source file or from multiple source files, and may use system libraries and header files**
- **This machine code is then stored in a file known as an executable file which is executed by the CPU**

January 19, 2018 Proprietary and Confidential 15

 **Capgemini**  
DIGITAL | TECHNOLOGY | BUSINESS



### Stages of Compilation:

There are four stages in the process of Compilation of a C program

#### The Preprocessor:

Preprocessing is the first step in the C program compilation stage. This feature is unique to C compilers. The Preprocessor accepts source code as input and is responsible for removing comments. All preprocessor directives or commands begin with a #. The Preprocessor accepts source code as input and is responsible for removing comments interpreting special preprocessor directives denoted by #.

For example:

```
#include - To include contents of a file
#include <math.h> - standard library maths file.
#include <stdio.h> - standard library I/O file
#define - defines a symbolic name or constant.
```

Preprocessing causes the contents of the indicated header file(s) (which is in glibc) to be copied into the current file, effectively replacing the directive line with the contents of that file.

#### C Compiler:

The C compiler translates source to assembly code. The source code is received from the preprocessor.

#### Assembler:

The assembler creates object code. On a UNIX system you may see files with a .o suffix (.OBJ on MSDOS) to indicate object code files.

#### Link Editor:

If a source file references library functions or functions defined in other source files the link editor combines these functions (with main()) to create an executable file. External Variable references resolved here als

1.1: Compiling C Programs Using gcc Compiler

Example

➤ Step1: Create hello.c using vi editor

```
#include <stdio.h>
int main (void)
{ printf ("Hello, world!\n"); return 0; }
```


➤ Step 2: Compile the file 'hello.c' with gcc

```
$ gcc -Wall hello.c -o hello
```

➤ Step 3: To run the program execute hello

```
$ ./hello
Hello, world!
```

January 19, 2016 Proprietary and Confidential < 7 >

Capgemini  
EXCELLENCE IN TECHNOLOGY SERVICES

Compiling the program:

```
$ gcc -Wall hello.c -o hello
```

This compiles the source code 'hello.c' to machine code and stores it in an executable file 'hello'. The output file for the machine code is specified using the -o option. This option is usually given as the last argument on the command line. If it is omitted, the output is written to a default file called 'a.out'.

Note: If a file with the same name as the executable file already exists in the current directory it will be overwritten.

The option -Wall turns on the most commonly-used compiler warnings—it is recommended that you always use this option! GCC will not produce any warnings unless they are enabled. Compiler warnings are an essential aid in detecting problems when programming in C and C++.

In this case, the compiler does not produce any warnings with the -Wall option, since the program is completely valid. Source code which does not produce any warnings is said to compile cleanly.

Run the program:

Type the path name of the executable like this:

```
$ ./hello
```

This loads the executable file into memory and causes the CPU to begin executing the instructions contained within it. The path ./ refers to the current directory, so ./hello loads and runs the executable file 'hello' located in the current directory.

1.1: Compiling C Programs Using gcc Compiler

Example (Contd...)

➤ Write a Program:

```
#include <stdio.h>

int main (void) {
    int a=4;
    printf ("Square of %d is %f/n",a,a*a);
    return 0; }
```


➤ Compile the program:

```
$gcc -Wall eg.c
eg.c: In function `main':
eg.c:7: warning: double format, different type arg (arg 3)
$ ./a.out
Square of 4 is 0.000000
```

January 19, 2016

Proprietary and Confidential

18

Capgemini  
INNOVATIVE TECHNOLOGY SOLUTIONS

#### Using Wall Option:

This indicates that a format string has been used incorrectly in the file 'eg.c' at line 7. The messages produced by gcc always have the form file:line-number:message. The compiler distinguishes between error messages, which prevent successful compilation, and warning messages which indicate possible problems (but do not stop the program from compiling).

In this case, the correct format specifier should be '%d' for an integer argument. The allowed format specifiers for printf can be found in any general book on C, such as the GNU C Library Reference Manual.

Without the warning option -Wall the program appears to compile cleanly, but produces incorrect results:

```
$ gcc eg.c
$ ./a.out
```

Square of 4 is 0.000000(incorrect output)

The incorrect format specifier causes the output to be corrupted, because the function printf is passed an integer instead of a floating-point number. Integers and floating-point numbers are stored in different formats in memory, and generally occupy different numbers of bytes, leading to a spurious result. The actual output shown above may differ, depending on the specific platform and environment.

Clearly, it is very dangerous to develop a program without checking for compiler warnings. If there are any functions which are not used correctly they can cause the program to crash or produce incorrect results. Turning on the compiler warning option -Wall will catch many of the commonest errors which occur in C programming.




1.1: Compiling C Programs Using gcc Compiler

## Analysis

- **Preprocessing stage**  
gcc -E prog.c>prog.i
- **Compilation stage**  
gcc -S prog.i>prog.s
- **Assembling stage**  
gcc -c prog.s
  - This produces prog.o file
- **Linking stage:**  
gcc -o prog prog.o

January 25, 2018    Proprietary and Confidential    - 9 -



Stages of Compilation analyzed:

There are four stages in the C compilation model:

1. Preprocess
2. Compile
3. Assemble and
4. Linking.

We will track the following simple C program through the stages. The program source is contained in the file prog.c.

```
#define MESSAGE "hello world\n"
int main(void)
{
    printf(MESSAGE);
    return 0;
}
```

The Preprocessor:

The preprocessor is a program that processes the source text of a C program before the compiler. It has three major functions:

1. File inclusion
2. Macro replacement and
3. Conditional inclusion.

C CPP ON UNIX

|

Compiling C and CPP Programming on Unix

Cont..

File inclusion is the insertion of the text of a file into the current file, macro replacement is the replacement of one string by another, and conditional inclusion is the selective inclusion and exclusion of portions of source text on the basis of a computed condition. Our simple program prog.c just contains the single preprocessor directive

```
#define MESSAGE "hello world\n"
```

which causes MESSAGE to be replaced by "hello world\n" in the code. To preprocess prog.c and redirect the result to the file prog.i use the command

```
gcc -E prog.c > prog.i
```

Compile:

In this stage, it checks for the syntax errors and generates the assembly code. To compile the preprocessed code use the command

```
gcc -S prog.i
```

Assemble:

In this stage the assembled code is converted to the binary format, which is understood by the system. To convert to binary code use the command

```
gcc -c prog.s
```

This will produce an object file named prog.o.

Linking:

In this stage the object files are linked with the library functions.

The linking may be static or dynamic.

Static Linking: In this type of linking the entire definition of library functions will be copied into the executable image. This may require more disk space and memory than dynamic linking, but can be more portable (does not require the presence of the library on the system where it is run).

To perform static compilation use the following command:

```
gcc -static -o prog prog.c
```

Dynamic Linking:

Dynamic linking is accomplished by placing the name of a sharable library in the executable image. Actual linking with the library routines does not occur until the image is run, when both executable and library are placed in memory. An advantage of dynamic linking is that multiple programs can share a single copy of the library.

```
gcc -o prog prog.o
```

All four stages can be done in one go with the command

```
gcc -o prog prog.c
```

which produces an executable in the file prog.

1.1: Compiling C Programs Using gcc Compiler


## Need for Multiple Source Files

- Whole program needs to be compiled whenever any small change is made to the program
- Recompile of large program is very time consuming
- In case of independent source files, only the files changed need to be recompiled

January 29, 2016

Proprietary and Confidential

+ 11 +

  
CENTRAL TECHNOLOGY R&D

### Need for Multiple Source Files:

If a program is stored in a single file, then any change to an individual function requires the whole program to be recompiled to produce a new executable. The recompilation of large source files can be very time-consuming.

When programs are stored in independent source files, only the files that have changed need to be recompiled after the source code has been modified. In this approach, the source files are compiled separately and then linked together -a two stage process. In the first stage, a file is compiled without creating an executable. The result is referred to as an object file, and has the extension '.o' when using gcc.

In the second stage, the object files are merged together by a separate program called the linker. The linker combines all the object files to create a single executable.

An object file contains machine code where any references to the memory addresses of functions (or variables) in other files are left undefined. This allows source files to be compiled without direct reference to each other. The linker fills in these missing addresses when it produces the executable.


1.1: Compiling C Programs Using gcc Compiler

## Multiple Source Files

- A program can be split into multiple files for:
  - Ease of maintenance and understanding, incase of large programs
- In such cases the object files of the individuals can be linked together to form an executable.

```
$ gcc -Wall main.c hello_fn.c -o newhello
```

January 19, 2016 Proprietary and Confidential 12

 Capgemini  
EXCELLENCE TECHNOLOGY INNOVATIONS

Compiling Multiple Source Files:

In the following example we will split up the program 'Hello World' into three files: 'main.c', 'hello\_fn.c' and the header file 'hello.h'. Here is the main program 'main.c':

```
#include "hello.h"
int main (void)
{ hello ("world");
  return 0; }
```

The original call to the printf system function in the previous program 'hello.c' has been replaced by a call to a new external function hello, which we will define in a separate file 'hello\_fn.c'.

Cont...

Continued from previous notes page:

The main program also includes the header file 'hello.h', which will contain the declaration of the function hello. The declaration is used to ensure that the types of the arguments and return value match up correctly between the function call and the function definition. We no longer need to include the system header file 'stdio.h' in 'main.c' to declare the function printf, since the file 'main.c' does not call printf directly. The declaration in 'hello.h' is a single line specifying the prototype of the function hello:

```
void hello (const char * name);
```

The definition of the function hello itself is contained in the file

```
'hello_fn.c':  
#include <stdio.h>  
#include "hello.h"  
void hello (const char * name)  
{ printf ("Hello, %s!\n", name); }
```

This function prints the message "Hello, name!" using its argument as the value of name.

To compile these source files with gcc, use the following command:

```
$ gcc -Wall main.c hello_fn.c -o newhello
```

In this case, we use the -o option to specify a different output file for the executable, 'newhello'. Note that the header file 'hello.h' is not specified in the list of files on the command line. The directive #include "hello.h" in the source files instructs the compiler to include it automatically at the appropriate points.

To run the program, type the path name of the executable:

```
$ ./newhello  
Hello, world!
```

1.1: Compiling C Programs Using gcc Compiler

## Creating Object Files from Source Files


- The `gcc -c` option is used to produce an object file

```
$ gcc -Wall -c main.c
```

- This produces an object file 'main.o' containing the machine code for the main function
- The corresponding command for compiling the hello function in the source file 'hello\_fn.c' is:

```
$ gcc -Wall -c hello_fn.c
```

January 19, 2016 Proprietary and Confidential 14



### Creating Object Files from Source Files:

The command-line option `-c` is used to compile a source file to an object file. For example, the following command will compile the source file 'main.c' to an object file:

```
$ gcc -Wall -c main.c
```

This produces an object file 'main.o' containing the machine code for the main function. It contains a reference to the external function `hello`, but the corresponding memory address is left undefined in the object file at this stage (it will be filled in later by linking). The corresponding command for compiling the `hello` function in the source file

```
'hello_fn.c' is:  
$ gcc -Wall -c hello_fn.c
```

This produces the object file 'hello\_fn.o'.

Note: There is no need to use the option `-o` to specify the name of the output file in this case. When compiling with `-c` the compiler automatically creates an object file whose name is the same as the source file, but with '.o' instead of the original extension.


1.1: Compiling C Programs Using gcc Compiler

## Creating Executable Files from Object Files

➤ An executable file is created using gcc to link the object files together

```
$ gcc main.o hello_fn.o -o hello
```

January 19, 2016 Proprietary and Confidential 15

 Capgemini  
INNOVATIVE TECHNOLOGY SOLUTIONS

### Creating Executable Files from Object Files:

The final step in creating an executable file is to use gcc to link the object files together and fill in the missing addresses of external functions. To link object files together, they are simply listed on the command line:

```
$ gcc main.o hello_fn.o -o hello
```

This is one of the few occasions where there is no need to use the `-Wall` warning option, since the individual source files have already been successfully compiled to object code. Once the source files have been compiled, linking is an unambiguous process which either succeeds or fails (it fails only if there are references which cannot be resolved).

The resulting executable file can now be run:

```
$ ./hello  
Hello, world!
```

It produces the same output as the version of the program using a single source file in the previous section



## 1.1: Compiling C Programs Using gcc Compiler

## Important gcc Options


gcc option	Usage
<b>-c file</b>	Direct gcc to compile the source files into an object files without going through the linking stage.
<b>-o file</b>	Specifies that gcc's executable output should be named <i>file</i> .
<b>-g file</b>	Directs the compiler to include extra debugging information in its output.
<b>-I dir</b>	Adds the directory <i>dir</i> to the list of directories searched for #include files. There is no space between the "-I" and the directory name.
<b>-l mylib</b> (lower case L)	Search the library named <i>mylib</i> for unresolved symbols (functions, global variables) when linking. The actual name of the file will be <i>libmylib.a</i> .
<b>-Wall</b>	Show all compiler warnings. This flag is useful for finding problems that are not necessarily compilation errors. It tells the compiler to print warnings issued during compile which are normally hidden.

1.2: Using static and dynamic link library

## Introduction

- **Library file is a unit of data(such as C/C++ routine, classes or variables), which can be shared by many programs.**
- **There are two types of libraries**
  - Static Library
  - Dynamic Library

January 19, 2016 Proprietary and Confidential 17



C functions/C++ classes and methods which can be shared by more than one application are separated out of the application's source code, compiled and bundled into a library. The C standard libraries and C++ STL are examples of shared components which can be linked with your code.

Whenever there are some C / C++ routines (methods), classes or variables which can be shared by many programs are put in the separate file such files are called as library files. These are called as user defined library files. There are two types of libraries –

1. Static Library

2. Dynamic Library

These libraries are named with the prefix "lib". When linking, the command line reference to the library will not contain the library (lib) prefix or suffix.

`gcc src-file.c -lm -lpthread`


The libraries referenced in the above example for inclusion during linking are the math library and the thread library. They are found in `/usr/lib/libm.a` and `/usr/lib/libpthread.a`.

1.2: Using static and dynamic link library

## Introduction

- **If the library files get linked with object files and standalone executable at compile time then it is called as static link library**
- **Advantages:**
  - It avoids dependency problem :
  - Result in a performance improvement
  - Static linking can allow the application to be contained in a single executable file, simplifying distribution and installation

January 19, 2016 Proprietary and Confidential 18



### Static Library

If the library files get linked with object files and standalone executable at compile time then it is called as static link library.

#### Advantages:

1. It avoids dependency problem :

The application can be certain that all its libraries are present and that they are the correct version.
2. Some times it can result in a performance improvement.


Since in static link library every application has there own copy of library file, it need not have to wait for getting access to library function like it is in dynamic link library. The program control need not have to jump to some other file (location to execute library function, which reduces the time of execution.
3. Static linking can allow the application to be contained in a single executable file, simplifying distribution and installation.

1.2: Using static and dynamic link library

## Creating Static Link Library

- **Create Source Code files**
  - For Example, welcome.c, success.c and main.c
- **Create object file for the source code**
  - gcc -c welcome.c success.c
- **Creates an archive library, which consists of the named object files using ar command**
  - \$ ar -cr libmessage.a welcome.o success.o
    - cr - "create and replace". If the library does not exist, it is first created. If the library already exists, any original files in it with the same names are replaced by the new files

January 19, 2016    Proprietary and Confidential    < 19 >



```
message.h
void welcome (const char * name);
void success (void);
```

```
welcome.c
#include <stdio.h>
#include "message.h"
void welcome (const char * name)
{
    printf ("Hello, %s!\n", name);
}
```


```
Success.c
#include <stdio.h>
#include "message.h"
void success (void)
{
    printf ("Congratulations on your best performance!\n");
}
```

1.2: Using static and dynamic link library

## Creating Static Link Library (Contd...)

- **Generate output file using static linking**
  - `$ gcc -L -o main.o main.c -lmessage`
- **Run the program**
  - `./main.o`

January 19, 2016 Proprietary and Confidential 20



main.c

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

```
int main (void)
{
    welcome ("John");
    success ();
    return 0;
}
```

Compile the files as follows:

step1: Create object file welcome.o and success.o

```
$ gcc -c welcome.c success.c
```

Step2 : Creates an archive library, libmessage.a, which consists of the named object files.

```
$ ar -cr libmessage.a welcome.o success.o
```

The option cr stands for "create and replace". If the library does not exist, it is first created. If the library already exists, any original files in it with the same names are replaced by the new files specified on the command line. The first argument 'libhello.a' is the name of the library. The remaining arguments are the names of the object files to be copied into the library.

step3: Generate output file main.o using static linking. Use the -l option to link your program with libmessage.a

The link editor incorporates in your executable only the object files in this archive that contain a function you have called in your program.

Options used in command

- L – specifies the directory name where to search libmessage.a file
- o specifies name of executable file
- l to link output file with libmessage.a file

```
$ gcc -L. -o main.o main.c -lmessage
```

step4: Run the program

```
$ ./main.o
```

Hello, John

Congratulations on your best performance

Since this the library files welcome.c and success.c is included in executable file main.o (static linking) even if you remove those files still you can execute the main.o

1.2: Using static and dynamic link library


## Disadvantages

### ➤ Disadvantages:


- The size of executable file becomes greater as compared to dynamic link library because the library code is stored within the executable rather than in separate files.
- A change in the library source requires building executable source also

# Demo

➤ Demo: Creating static library.



January 19, 2016 | Proprietary and Confidential | 23

  
INNOVATIVE TECHNOLOGY SOLUTIONS



1.2: Using static and dynamic link library

## Introduction

- If the library files get linked with shared object files and standalone executable at run time then it is called as dynamic link library.
- Advantages:
  - This allows the library files to be shared between many applications leading to space savings
  - It allows the library to be updated to fix bugs and security flaws without updating the applications that use the library

January 19, 2016 Proprietary and Confidential 24



1.2: Using static and dynamic link library

## Shared Libraries

- Shared libraries are libraries that are loaded by programs at runtime with extension '.so', which stands for shared object
- Linker makes a note that the calling function is a part of the shared object, so it adds startup code instruction of the required shared library, instead of extracting executable code from the shared object
- The shared libraries supports position independence by using -fPIC while compiling the program

January 19, 2016 Proprietary and Confidential 25




1.2: Using static and dynamic link library

## Creating Dynamic link library

- **Create Source Code files**
  - For Example, welcome.c, success.c and main.c
- **Create Library**
  - `gcc -Wall -fPIC -c welcome.c success.c`  
Link object files into a shared library named as "libmessage.so"
  - `gcc -shared -o libmessage.so welcome.o success.o`
- **Create executable file main.o by setting the environment variable to search for the library file while linking**

January 19, 2016    Proprietary and Confidential    26



Compiler options:

- Wall: include warnings. See man page for warnings specified.
- fPIC: Compiler directive to output position independent code, a characteristic required by shared libraries.
- shared: Produce a shared object which can then be linked with other objects to form an executable.
- Wl: Pass options to linker.
- o: Output of operation. In the above example the name of the output will be "libmessage.so"

Compile the files as follows

Step 1 : Create library

```
gcc -Wall -fPIC -c welcome.c success.c
```

```
gcc -shared -o libmessage.so welcome.o success.o
```

Step 2: Create executable file main.o

When you use the -l option, you must point the dynamic linker to the directories of the dynamically linked libraries that are to be linked with your program at execution by setting LD\_RUN\_PATH

To set LD\_RUN\_PATH, list the absolute pathnames of the directories you want searched in the order you want them searched.

```
LD_RUN_PATH=/home/user11 export LD_RUN_PATH
```

1.2: Using static and dynamic link library


## Creating Dynamic link library (Contd...)

- Environment variable - LD\_RUN\_PATH directs the dynamic linker to search the library file in the specified absolute path
  - LD\_RUN\_PATH=/home/user11 export LD\_RUN\_PATH
- gcc -o main.o main.c -L/home/user11 -lmessage

➤ **Run the Program.**

- ./main.o

January 19, 2016 Proprietary and Confidential 22



```
gcc -o main.o main.c -L/home/user11 welcome.c success.c -lmessage
```

directs the dynamic linker to search for libmessage.so in /home/user11 when you execute your program.

The dynamic linker searches the standard place by default, after the directories you have assigned to LD\_RUN\_PATH. The standard place for libraries is /usr/lib. Any executable versions of libraries supplied by the compilation system kept in /usr/lib.

Step 3:Run the program.

```
./main.o
```

Hello, John!

Congratulations on your best performance!

1.2: Using static and dynamic link library

## Different ways to locate library file

- **At the compile time, gcc employs -L and -l options to tell where a library locates and which library to be linked, these options are passed to ld**
- **There are two ways to determine a library path**
  - Compile time
    - Using Environment variable LD\_RUN\_PATH
    - Using command line option -rpath or -R
  - Execution time
    - Using Environmental variable LD\_LIBRARY\_PATH

January 19, 2016 Proprietary and Confidential 28



1.2: Using static and dynamic link library

## Different ways to locate library file (Contd...)

### ➤ Using command line option `--rpath` or `-R`:

- Setting the environmental variables like `LD_LIBRARY_PATH` or `LD_RUN_PATH` can override and modify the system-wide defaults.
- Use linker option `--rpath` in `ld` command or `"-Wl,rpath,"` in `gcc` command to specify the current directory as a search directory
  - For example:
- `$gcc -o main.o main.c -Wl,-rpath,/home/user11 -L/home/user11 -lmessage`

January 19, 2016 Proprietary and Confidential 28



### Using command line option `--rpath` or `-R`:

Setting the environmental variables like `LD_LIBRARY_PATH` or `LD_RUN_PATH` can override and modify the system-wide defaults.

Use linker option `--rpath` in `ld` command or `"-Wl,rpath,"` in `gcc` command to specify the current directory as a search directory For example

```
$gcc -o main.o main.c -Wl,-rpath,/home/user11 -L/home/user11 -lmessage
```

1.2: Using static and dynamic link library

## Different ways to locate library file (Contd...)

### ➤ LD\_LIBRARY\_PATH

- Once the library file is moved to another location, You can temporarily substitute a different library for particular execution of your program without performing compilation again

- For example,

- \$ LD\_LIBRARY\_PATH=/home/user11/dynamic export LD\_LIBRARY\_PATH
  - \$. /main.o

### ➤ ldd command – Used to check which shared object is in use.

- \$ ldd main.o



January 19, 2016 Proprietary and Confidential 30

### LD\_LIBRARY\_PATH

The environment variable LD\_LIBRARY\_PATH lets you do the same thing at run time.

Suppose the library file is moved from /home/user11 to /home/user11/dynamic, You can temporarily substitute a different library for particular execution of your program without performing compilation again as follows:

```
$ LD_LIBRARY_PATH=/home/user11/dynamic export LD_LIBRARY_PATH
$. /main.o
```

Now when you execute your program, the dynamic linker searches for libmessage.so first in /home/user11 and, if not found, then it will search in /home/user11/dynamic. The directory assigned to LD\_RUN\_PATH is searched before the directory assigned to LD\_LIBRARY\_PATH.

ldd command – Used to check which shared object is in use.

```
$ ldd main.o
    /lib/libcwait.so (0x00c67000)
    libmessage.so => /home/user11/libmessage.so (0x00111000)
    libc.so.6 => /lib/tls/libc.so.6 (0x00366000)
    /lib/ld-linux.so.2 (0x0034d000)
```

## Demo

- Demo: Creating dynamic link library.





1.3: Compiling C++ Programs Using g++ Compiler

## Introduction to g++ Compiler

- **The GNU g++ compiler can be used to control both the compilation phase and the linking phase of creating executable files for C++ programs**
- **Steps to obtain the final executable program**
  - **Compiler stage**
    - The .cpp file is converted to Assembly language (.s files)
  - **Assembler stage**
    - Assembly language code is converted into object code (.o file)
  - **Linker stage**
    - Object files linked to produce an executable file

January 19, 2016 Proprietary and Confidential 32



## 1.3: Compiling C++ Programs Using g++ Compiler

## Process

## ➤ Step 1: Create a test.cpp program

```
#include <iostream>
using namespace std;
int main()
{ cout << "Testing 1, 2, 3 \n";
  return 0; }
```

## ➤ Step 2: Compile your program with g++ compiler

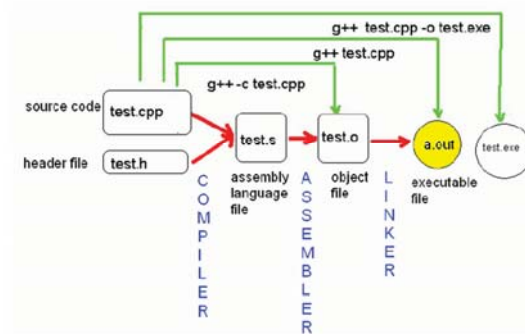
```
$g++ test.cpp
```

## ➤ Step 3: If there are no errors, an executable file is generated with the standard name a.out

```
$/a.out
$Testing 1, 2, 3
```

## 1.3: Compiling C++ Programs Using g++ Compiler

## Overview



## 1.3: Compiling C++ Programs Using g++ Compiler

## Important g++ Compiler Options

g++ compiler options	Usage
<b>-C</b>	This flag forces the compiler to produce object files from source files without linking
<b>-o</b>	This flag allows renaming the executable file with another name than a.out.
<b>-Wall</b>	This flag forces the compiler to display all the warning messages.

All the options of g++ compiler behaves in the same manner as that of the gcc compiler.

## Summary

- **In this lesson, we learnt about:**
- Various stages of compilation
  - gcc compiler and its various options
  - g++ compiler and its various options



## Review Question

- **Question 1: The gcc -c option is used:**
  - To produce an executable file
  - To produce an object file
  - To produce warnings
- **Question 2: Which of the following options is true regarding the gcc compiler -Wall option?**
  - Used to display warnings
  - Used to display message to all users



## Review Question

- **Question 3: The compiler used to compile C++ Programs is:**
  - gcc
  - g++
  - Both Option1 and Option2 can be used
- **Question 4: Dynamic linking makes executable files smaller and saves disk space**
  - True/False

