南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

Lab 3 Common Commands

于仕琪，王大兴，廖琪梅

Commands in Linux

Linux is a family of open-source Unix operating systems based on the Linux Kernel.
There are some popular distributions
such as Ubuntu, Fedora, Debian, openSUSE, and Red Hat.

A Linux command is a program or utility that runs on the Command Line Interface – a console that interacts with the system via texts and processes.

Linux command's general syntax looks like:

CommandName [option(s)] [parameter(s)]

- **CommandName** is the rule that you want to perform.
- **Option** or **flag** modifies a command's operation. To invoke it, use hyphens (-) or double hyphens (--).
- **Parameter** or **argument** specifies any necessary information for the command.

Common commands in Linux

Linux directory and file commands:

Command	Meaning
pwd	**P**rint the name of current/**w**orking **d**irectory.
cd <directory name>	**C**hange the current **d**irectory.
ls	**List** of content of a directory.
mkdir <directory name>	**Mak**e a new directory under any directory.
rmdir <directory name>	**Rem**ove directories without files.
cat <file name>	Display content of the file.
rm <file name>	**Rem**ove a file.
cp <source> <dest>	**Cop**y a file or files to another
mv<source><dest>	**Mov**e a file or files to another directory

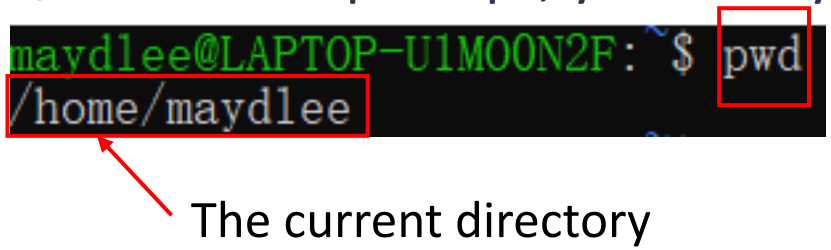
https://www.javatpoint.com/linux-commands

pwd command

Use the **pwd** command to display the current working directory you are in.

Start Ubuntu, you will see:

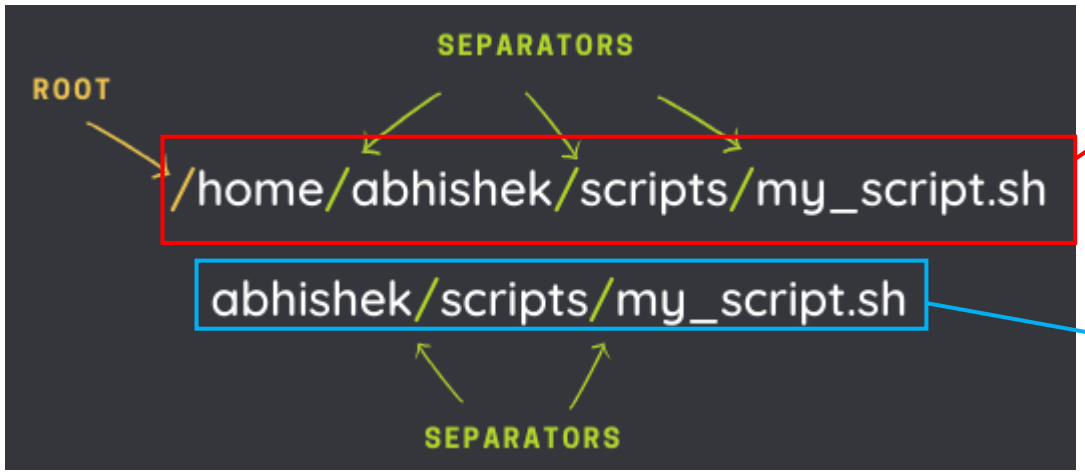
$ or **#** is the prompt, you can type command now.



Absolute path and relative path

A **path** is how you refer to files and directories. It gives the location of a file or directory in the Linux directory structure. It is composed of a **name** and **slash** syntax.

If the path starts with slash "/", the first slash denotes root. The rest of the slashes in the path are just separators.



The **absolute path** always starts from the root directory (/).

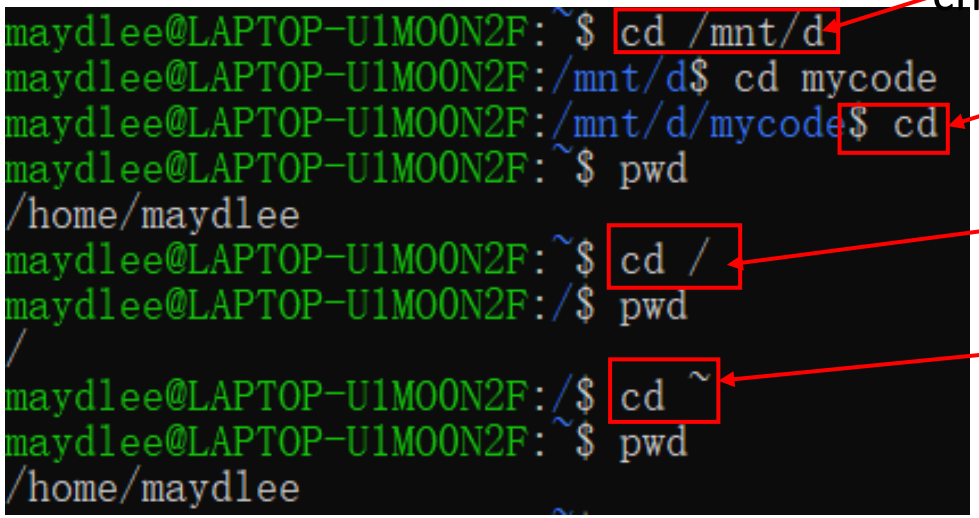
A **relative path** starts from the current directory.

Two special relative paths:

- . (single dot) denotes the current directory in the path.
- .. (two dots) denotes the parent directory, i.e., one level above.

cd command

To navigate through the Linux files and directories, use the **cd** command.



change the directory to the root directory of d drive

change the directory to the home directory

change the directory to the root directory

change the directory to the home directory

Here are some shortcuts to help you navigate:

- **cd ~[username]** goes to another user's home directory.
- **cd ..** moves one directory up.
- **cd -** moves to your previous directory.
- **cd** without an option will take you to the home folder.

ls command

The **ls** command lists files and directories within a system. Running it without a flag or parameter will show the current working directory's content.

```
maydlee@LAPTOP-U1MOON2F:/mnt/d$ cd CMake
maydlee@LAPTOP-U1MOON2F:/mnt/d/CMake$ ls
CMakeCache.txt  CMakeLists.txt  Demo2  HelloWorld.cpp  cmake_install.cmake
CMakeFiles      Demo1           Demo3  Makefile        hello.exe
maydlee@LAPTOP-U1MOON2F:/mnt/d/CMake$ ls Demo1
CMakeCache.txt  CMakeFiles  CMakeLists.txt  Makefile  cmake_install.cmake  hello.exe  main.cpp
maydlee@LAPTOP-U1MOON2F:/mnt/d/CMake$ ls -l
total 188
-rwxrwxrwx 1 maydlee maydlee  14456 Oct 25  2020 CMakeCache.txt
drwxrwxrwx 1 maydlee maydlee   4096 Jun  4  2021 CMakeFiles
-rwxrwxrwx 1 maydlee maydlee     99 Oct 25  2020 CMakeLists.txt
drwxrwxrwx 1 maydlee maydlee   4096 Feb 22  2021 Demo1
drwxrwxrwx 1 maydlee maydlee   4096 Feb 22  2021 Demo2
drwxrwxrwx 1 maydlee maydlee   4096 Jun  4  2021 Demo3
-rwxrwxrwx 1 maydlee maydlee    114 Oct 30  2020 HelloWorld.cpp
-rwxrwxrwx 1 maydlee maydlee   4783 Oct 25  2020 Makefile
-rwxrwxrwx 1 maydlee maydlee   1341 Oct 25  2020 cmake_install.cmake
-rwxrwxrwx 1 maydlee maydlee 160805 Oct 30  2020 hello.exe
```

List subdirectory and files in the current directory

List subdirectory and files in the Demo1 directory

List detail information of subdirectory and files in the current directory

Here are some options you can use with the **ls** command:

- **ls -R** lists all the files in the subdirectories.
- **ls -a** shows hidden files in addition to the visible ones.
- **ls -l (or ll)** shows detail information of subdirectory and files

mkdir command

Use the **mkdir** command to create one or multiple directories at once.

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d$ cd examples
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples$ ls
CMakeLists.txt  a.exe  a.exe.stackdump  main.cpp  matoperation.cpp  matoperation.hpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples$ mkdir demo1 demo2
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples$ ls
CMakeLists.txt  a.exe  a.exe.stackdump  demo1  demo2  main.cpp  matoperation.cpp  matoperation.hpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples$ mkdir demo1/exercise_demo
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples$ ls
CMakeLists.txt  a.exe  a.exe.stackdump  demo1  demo2  main.cpp  matoperation.cpp  matoperation.hpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples$ cd demo1
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples/demo1$ ls
exercise_demo
```

Create two subdirectories in the current directory

Create a subdirectory inside the demo1 directory

rmdir command

Use the **rmdir** command to permanently delete an empty directory.

```
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ mkdir demo1 demo2
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ ls
CMakeLists.txt  a.exe  a.exe.stackdump  demo1  demo2  main.cpp  matoperation.cpp  matoperation.hpp
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ mkdir demo1/exercise_demo
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ ls
CMakeLists.txt  a.exe  a.exe.stackdump  demo1  demo2  main.cpp  matoperation.cpp  matoperation.hpp
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ cd demo1
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples/demo1$ ls
exercise_demo
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples/demo1$ cd ..
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ rmdir demo1
rmdir: failed to remove 'demo1': Directory not empty
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ rmdir demo1/exercise_demo
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ rmdir demo1
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ ls
CMakeLists.txt  a.exe  a.exe.stackdump  demo2  main.cpp  matoperation.cpp  matoperation.hpp
```

Delete demo1 in the current directory

First delete the directory in demo1, then delete demo1

rm command

The **rm** command is used to delete files within a directory. Make sure that the user performing this command has write permissions.

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples$ cd demo2
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples/demo2$ ls
CMakeLists.txt  a.out  hello  hello.c  hello.o  main.cpp  welcome.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples/demo2$ rm a.out hello
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples/demo2$ ls
CMakeLists.txt  hello.c  hello.o  main.cpp  welcome.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples/demo2$ rm -i hello.o
rm: remove regular file 'hello.o'? y
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples/demo2$ ls
CMakeLists.txt  hello.c  main.cpp  welcome.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples$ rm demo2/*.*
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples$ cd demo2
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples/demo2$ ls
maydlee@LAPTOP-U1MO0N2F:/mnt/d/examples/demo2$
```

Delete two files without confirmation

Delete a file with confirmation

Delete all the files in demo2

Here are some acceptable options you can add:

- **-i** prompts system confirmation before deleting a file.
- **-f** allows the system to remove without a confirmation.
- **-r** deletes files and directories recursively.

cp command and mv command

The **cp** command is used to copy a file or directory.

```
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ ls
CMakeLists.txt  a.exe  a.exe.stackdump  demo2  main.cpp  matoperation.cpp  matoperation.hpp
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ cp main.cpp demo2
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ ls demo2
main.cpp
```

Copy a file into demo2

The **mv** command is used to move a file or a directory form one location to another location.

Move a file into demo2

```
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ mv CMakeLists.txt demo2
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ ls demo2
CMakeLists.txt  main.cpp
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ ls
a.exe  a.exe.stackdump  demo2  main.cpp  matoperation.cpp  matoperation.hpp
```

The CMakeLists.txt is not in the examples directory because it is moved into demo2.

Use mv command to rename a file

```
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ mv main.cpp test_main.cpp
maydlee@LAPTOP-U1MOON2F:/mnt/d/examples$ ls
a.exe  a.exe.stackdump  demo2  matoperation.cpp  matoperation.hpp  test_main.cpp
```

cat command

Concatenate, or **cat**, is one of the most frequently used Linux commands. It lists, combines, and writes file content to the standard output. To run the cat command, type **cat** followed by the file name and its extension.

```
maydlee@LAPTOP-U1MOON2F:/mnt/d/CMake$ cat HelloWorld.cpp
#include <iostream>
using namespace std;

int main()
{
        cout << "Hello World!" << endl;

        return 0;
}
```

Here are other ways to use the cat command:

- **cat > filename.txt** creates a new file.
- **cat filename1.txt filename2.txt > filename3.txt**

merges **filename1.txt** and **filename2.txt** and stores the output in **filename3.txt**.

- **tac filename.txt** displays content in reverse order.

Some other commands

- cat/tail/head
- less/more
- nano/vim
- file
- where
- echo

Shortcut keys

- **Up** and **down** arrow keys can list the commands you typed.
- **Tab** key can complete the command. For a long command, you can type first few letters and press Tab key to complete the command or list alternate commands.

```
maydlee@LAPTOP-U1MOON2F:/mnt/d/CMake$ mkd
mkdir       mkdir.exe   mkdosfs
```

Type the first few letters of a command, and then press Tab key. If there is completion, press Tab key again, it will list the alternate commands.

clear is a standard Unix computer operating system command that is used to clear the terminal screen.

```
maydlee@LAPTOP-U1MOON2F:/mnt/d/CMake$ clear
```

gcc & g++

gcc and **g++** are GNU C or C++ compilers respectively, which issued for preprocessing, compilation, assembly and linking of source code to generate an executable file.

Type command **gcc** or **g++ --help**, you can get the common options of the gcc or g++. **g++** accepts mostly the same options as **gcc**.

```
maydlee@LAPTOP-U1MOON2F: $ 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={common|optimizers|params|target|warnings|[^]{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.
  -std=<standard>          Assume that the input sources are for <standard>.
  --sysroot=<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>.
  -pie                     Create a dynamically linked position independent
                           executable.
  -shared                  Create a shared library.
```

gcc & g++

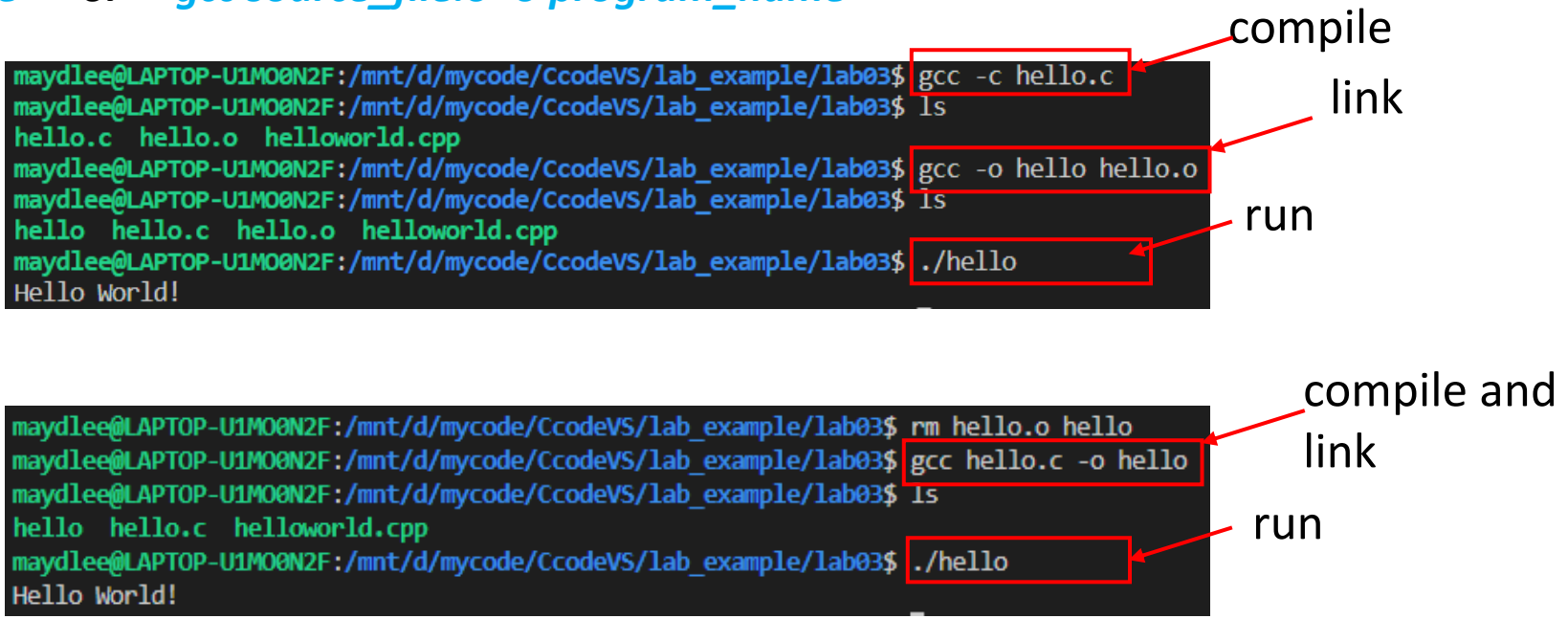
-c Compile or assemble the source files, but do not link. The ultimate output is in the form of an object file for each source file. The object file name for a source file is made by replacing the suffix **.c** with **.o**.

-o *<file>* Place output in file *file*. This applies regardless to whatever sort of output is being produced, whether it be an executable file, an object file, an assembler file or preprocessed C code. If **-o** is not specified, the default is to put an executable file in *a.out*.

gcc source_file.c -o program_name ***or*** ***gcc source_file.o -o program_name***

```
#include <stdio.h>
int main()
{
    printf("Hello World!\n");

    return 0;
}
```



gcc & g++

With one step to generate an executable target file:

gcc file_name or ***g++ file_name***

This command is used to compile and create an executable file *a.out* (default target name).

```
#include <iostream>
using namespace std;
int main()
{
	cout << "Hello World!!!" << endl;

	return 0;
}
```

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ gcc hello.c
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ ls
a.out  hello  hello.c  helloworld.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ ./a.out
Hello World!
```

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ rm a.out hello
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ g++ helloworld.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ ls
a.out  hello.c  helloworld.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ ./a.out
Hello World!!!
```

gcc & g++

compile multiple files

You can compile the files one by one and then link them to an executable file.

Another choice is using one step to list all the .c(or .cpp) files after gcc(or g++) command and create an executable file named a.out.

```
//area.h
#define PI 3.1415

double compute_area(double r);
```

```
//area.c
#include "area.h"

double compute_area(double r)
{
   return PI * r * r;
}
```

```
//main.c
#include <stdio.h>
#include "area.h"

int main()
{
   double r, area;

   printf("Please input a radius:");
   scanf("%lf", &r);

   area = compute_area(r);

   printf("The area of %lf is %.4lf\n", r, area);

   return 0;
}
```

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ gcc -c area.c
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ gcc -c main.c
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ ls
area.c  area.h  area.o  hello.c  helloworld.cpp  main.c  main.o
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ gcc -o main main.o area.o
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ ls
area.c  area.h  area.o  hello.c  helloworld.cpp  main  main.c  main.o
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ ./main
Please input a raduis:4.8
The area of 4.800000 is 72.3802
```

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ gcc area.c main.c
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab_example/lab03$ ./a.out
Please input a raduis:2.3
The area of 2.300000 is 16.62
```

Exercises

1. Demonstrate two Linux commends which are introduced previously and randomly selected by the SA for creating a directory, changing to another directory, listing the subdirectories and files in a certain directory, removing files, copying and moving files.

Exercises

```cpp
// factorial.cpp

#include "functions.h"
int factorial(int n)
{

    if (n == 1)
        return 1;
    else
        return n * factorial(n - 1);

}
```

```cpp
// main.cpp

#include <iostream>
#include "functions.h"
using namespace std;

int main()
{
    print_hello();

    cout << "This is main:" << endl;
    cout << "The factorial of 5 is: " << factorial(5) << endl;

    return 0;

}
```

```cpp
// printhello.cpp

#include <iostream>
#include "functions.h"
using namespace std;

void print_hello()
{
    cout << "Hello World!" << endl;
}
```

```cpp
// functions.h
void print_hello();
int factorial(int n);
```

2. Compile the 3 source files one by one using "g++ -c", then link the generated object files together to generate an execute file.

Demonstrate to a SA to pass the test.

Exercises

3. Run the following source code and explain the result.

You need to explain the reason to a SA to pass the test.

```
#include <iostream>
using namespace std;

int main()
{
   for(size_t n = 2; n >= 0; n--)
      cout << "n = " << n << "  ";

   return 0;

}
```

Exercises

4. Run the following source code and explain the result.

You need to explain the reason to a SA to pass the test.

```
#include <iostream>
using namespace std;

int main()
{
   int n = 5;
   int sum;
   while(n >0){
      sum += n;
      cout << "n = " << n << "  ";
      cout << "sum = " << sum << "  ";
   }

   return 0;

}
```

Exercises

5. Run the following source code and explain the result.

You need to explain the reason to a SA to pass the test.

```
#include <iostream>
using namespace std;

int main()
{
   int n,fa;

   do{
      fa *= n;
      n++;
   }while(n <= 10);

   cout << "fa = " << fa << endl;

   return 0;
}
```