南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

Lab 3, Makefile

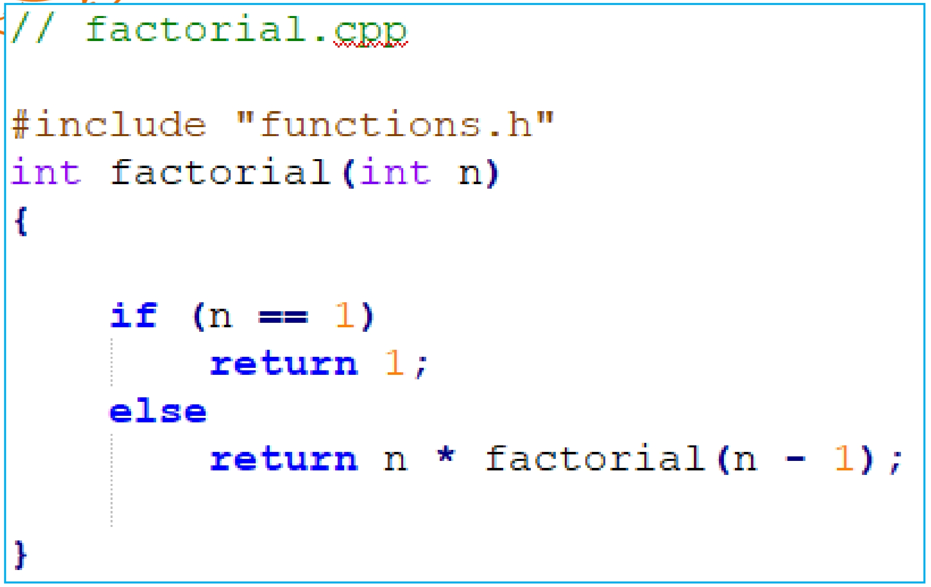
于仕琪，王大兴，廖琪梅

Makefile

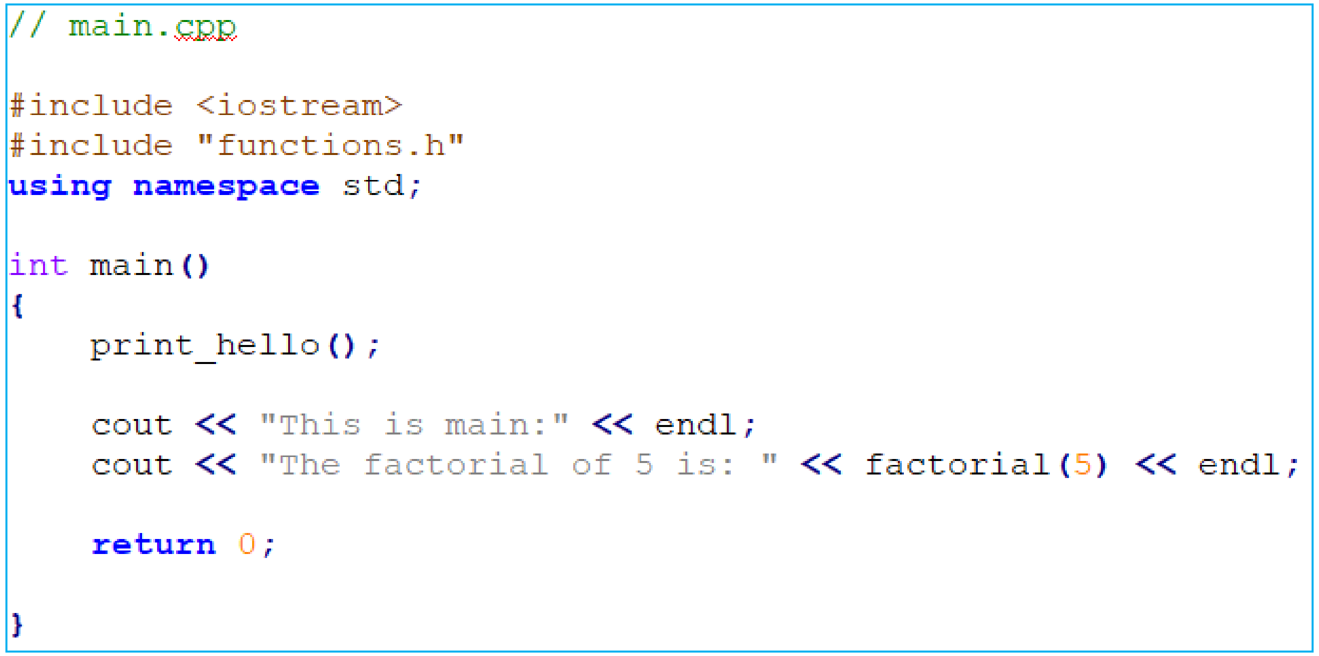
What is a Makefile?

Makefile is a tool to simplify or to organize for compilation. **Makefile is a set of commands with variable names and targets .** You can compile your project(program) or only compile the update files in the project by using Makefile.

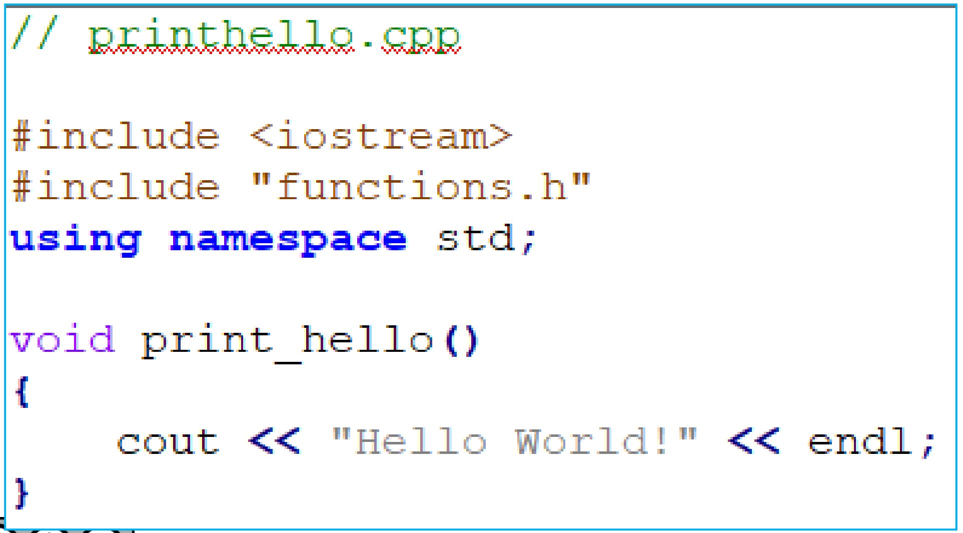
Suppose we have four source files as follows:



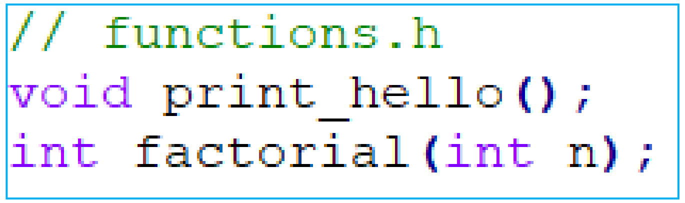
```cpp
// factorial.cpp

#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);
```

Normally, you can compile these files by the following command:

```
$ g++ -o hello main.cpp printhello.cpp factorial.cpp
```

How about if there are hundreds of files need to compile? Do you think it is comfortable to write g++ or gcc compilation command by mentioning all these hundreds file names? Now you can choose makefile.

The name of makefile must be either **makefile** or **Makefile** without extension. You can write makefile in any text editor. A rule of makefile including three elements: **targets**, **prerequisites** and **commands**. There are many rules in the makefile.

A makefile consists of a set of rules. A rule including three elements: **target**, **prerequisites** and **commands**.

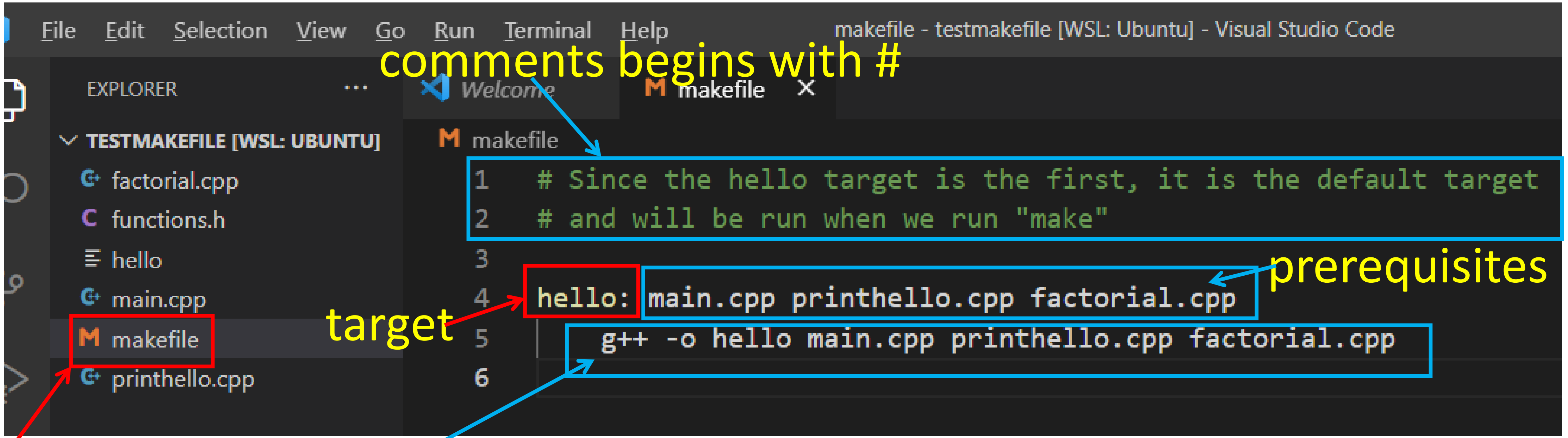
```
targets :  prerequisites

<TAB> command
```

- The **target** is an object file, which is generated by a program.

Typically, there is only one per rule.

- The **prerequisites** are file names, separated by spaces, as input to create the target.
- The **commands** are a series of steps that make carries out.

These need to start with a **tab character**, not spaces.


File Edit Selection View Go Run Terminal Help
makefile - testmakefile [WSL: Ubuntu] - Visual Studio Code
comments begins with #
EXPLORER
Welcome
makefile
TESTMAKEFILE [WSL: UBUNTU]
factorial.cpp
functions.h
hello
main.cpp
makefile
printhello.cpp
target
prerequisites

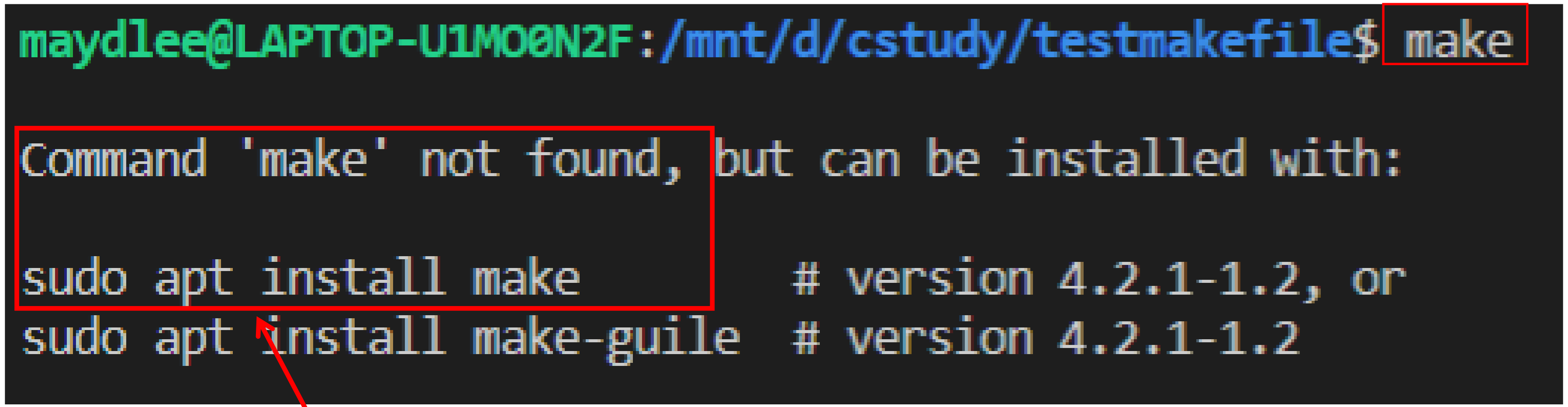
```
# Since the hello target is the first, it is the default target
# and will be run when we run "make"

hello: main.cpp printhello.cpp factorial.cpp
	g++ -o hello main.cpp printhello.cpp factorial.cpp

```

Put the makefile together with your programs.

commands

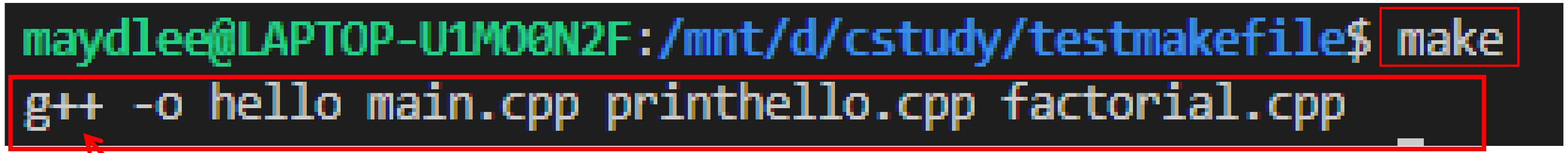
g++ is compiler name, -o is linker flag and hello is binary file name.

Type the command make in VScode



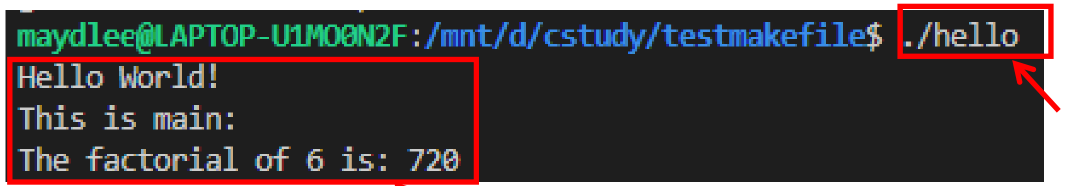
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ make

Command 'make' not found, but can be installed with:

sudo apt install make        # version 4.2.1-1.2, or
sudo apt install make-guile  # version 4.2.1-1.2
```

If you don’t install make in VScode, install it first according to the instruction.



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ make
g++ -o hello main.cpp printhello.cpp factorial.cpp
```

Run the commands in the makefile automatically.



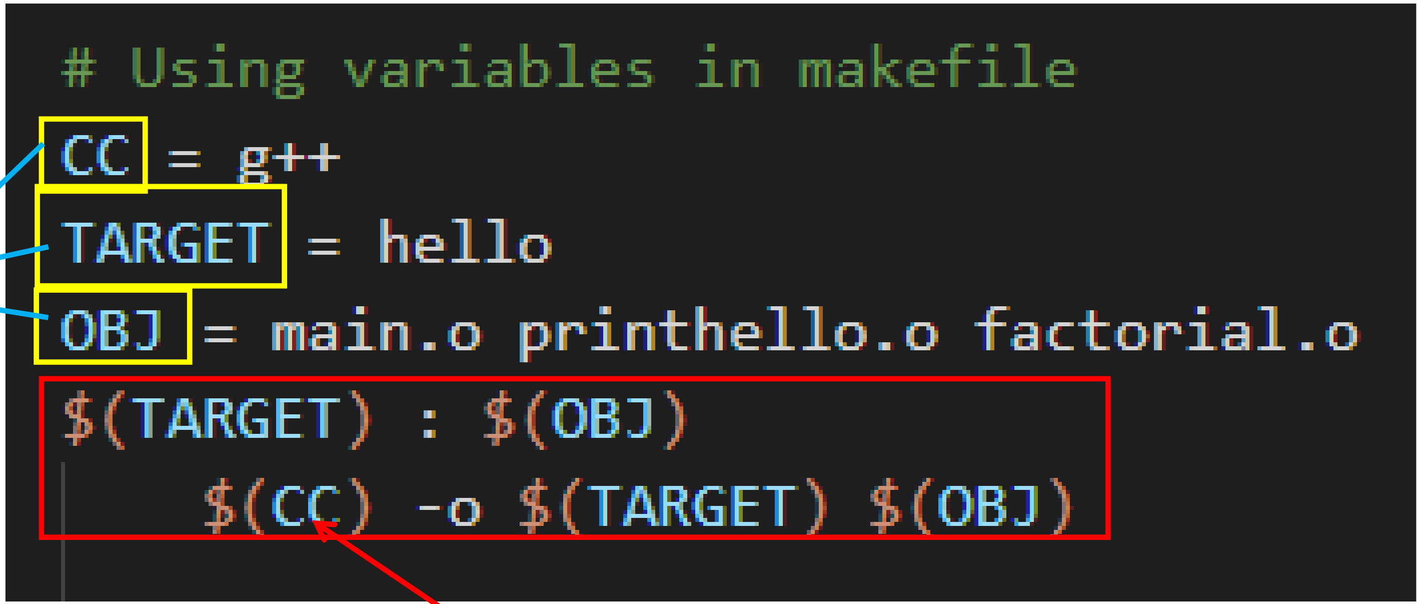
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ ./hello
Hello World!
This is main:
The factorial of 6 is: 720
```

Run your program

output

Defining Macros/Variables in the makefile

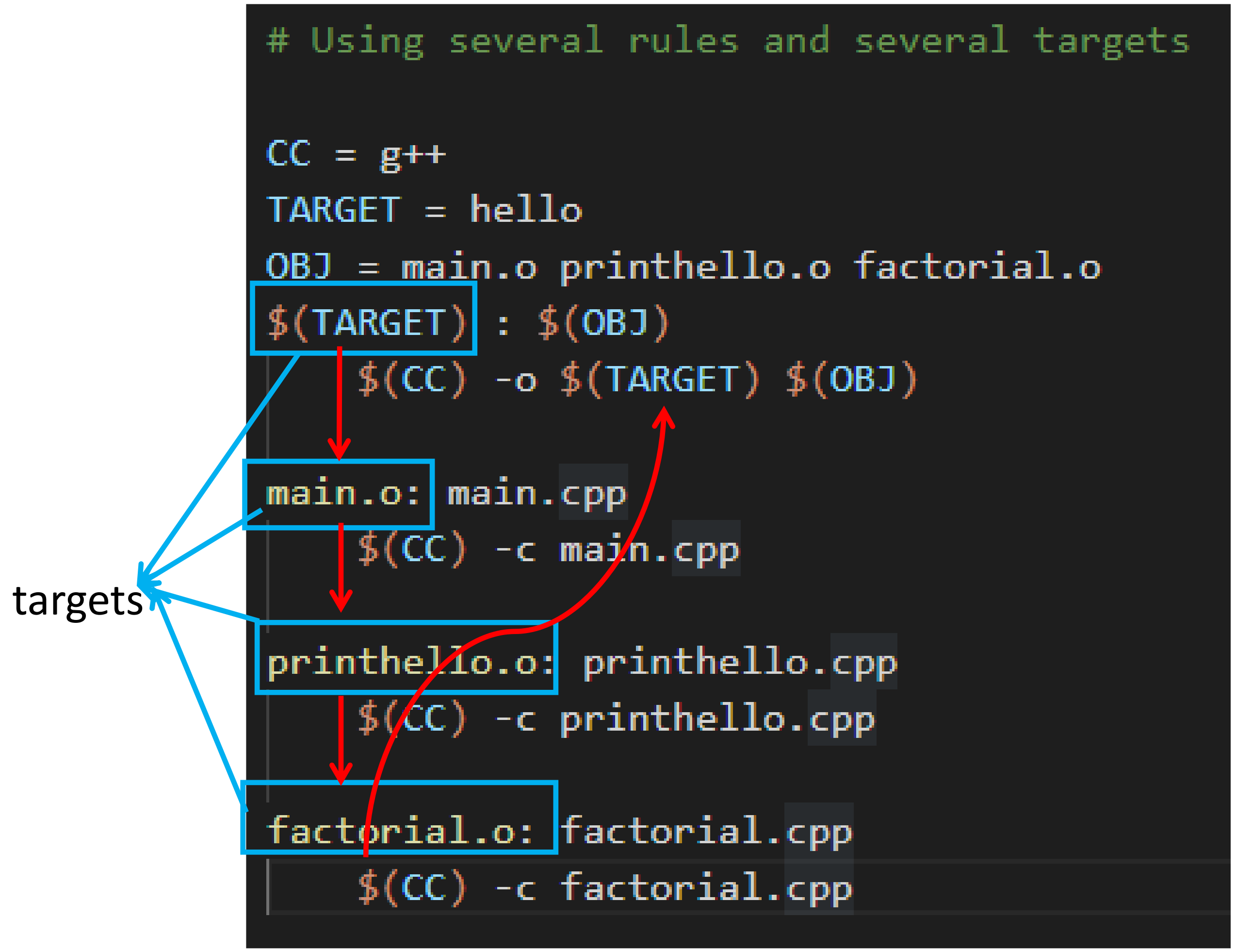
To improve the efficiency of the makefile, we use variables.



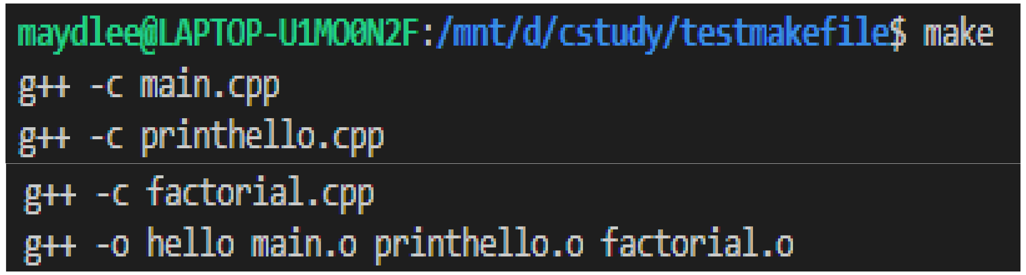
```
# Using variables in makefile
CC = g++
TARGET = hello
OBJ = main.o printhello.o factorial.o
$(TARGET) : $(OBJ)
	$(CC) -o $(TARGET) $(OBJ)
```

Write target, prerequisite and commands by variables using ‘$()’

If only one source file is modified, we need not compile all the files. So, let's modify the makefile.

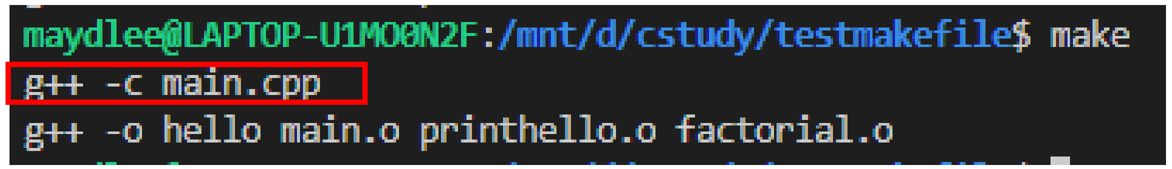


```
# Using several rules and several targets

CC = g++
TARGET = hello
OBJ = main.o printhello.o factorial.o
$(TARGET) : $(OBJ)
	$(CC) -o $(TARGET) $(OBJ)

main.o: main.cpp
	$(CC) -c main.cpp

printhello.o: printhello.cpp
	$(CC) -c printhello.cpp

factorial.o: factorial.cpp
	$(CC) -c factorial.cpp
```



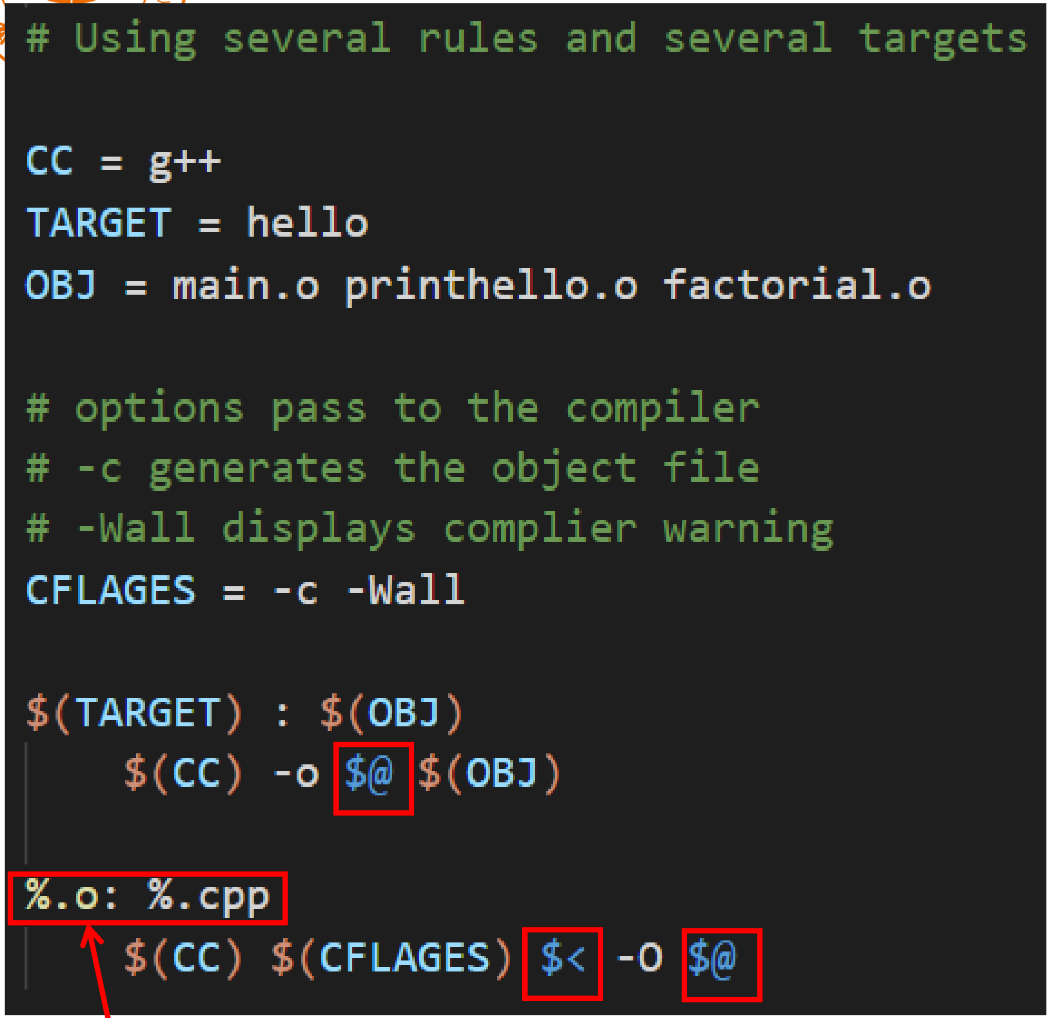
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ make
g++ -c main.cpp
g++ -c printhello.cpp
g++ -c factorial.cpp
g++ -o hello main.o printhello.o factorial.o
```

If main.cpp is modified, it is compiled by make.



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ make
g++ -c main.cpp
g++ -o hello main.o printhello.o factorial.o
```

All the .cpp files are compiled to the .o files, so we can modify the makefile like this:



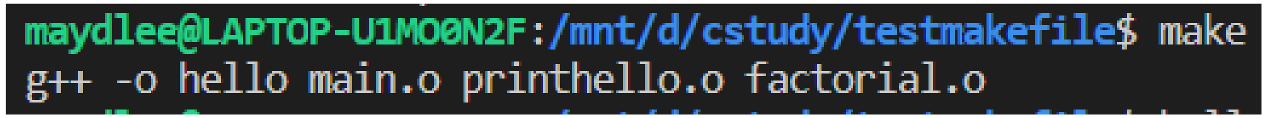
```
# Using several rules and several targets

CC = g++
TARGET = hello
OBJ = main.o printhello.o factorial.o

# options pass to the compiler
# -c generates the object file
# -Wall displays complier warning
CFLAGES = -c -Wall

$(TARGET) : $(OBJ)
	$(CC) -o $@ $(OBJ)

%.o: %.cpp
	$(CC) $(CFLAGES) $< -o $@
```

This is a model rule, which indicates that all the .o objects depend on the .cpp files

$@: Object Files

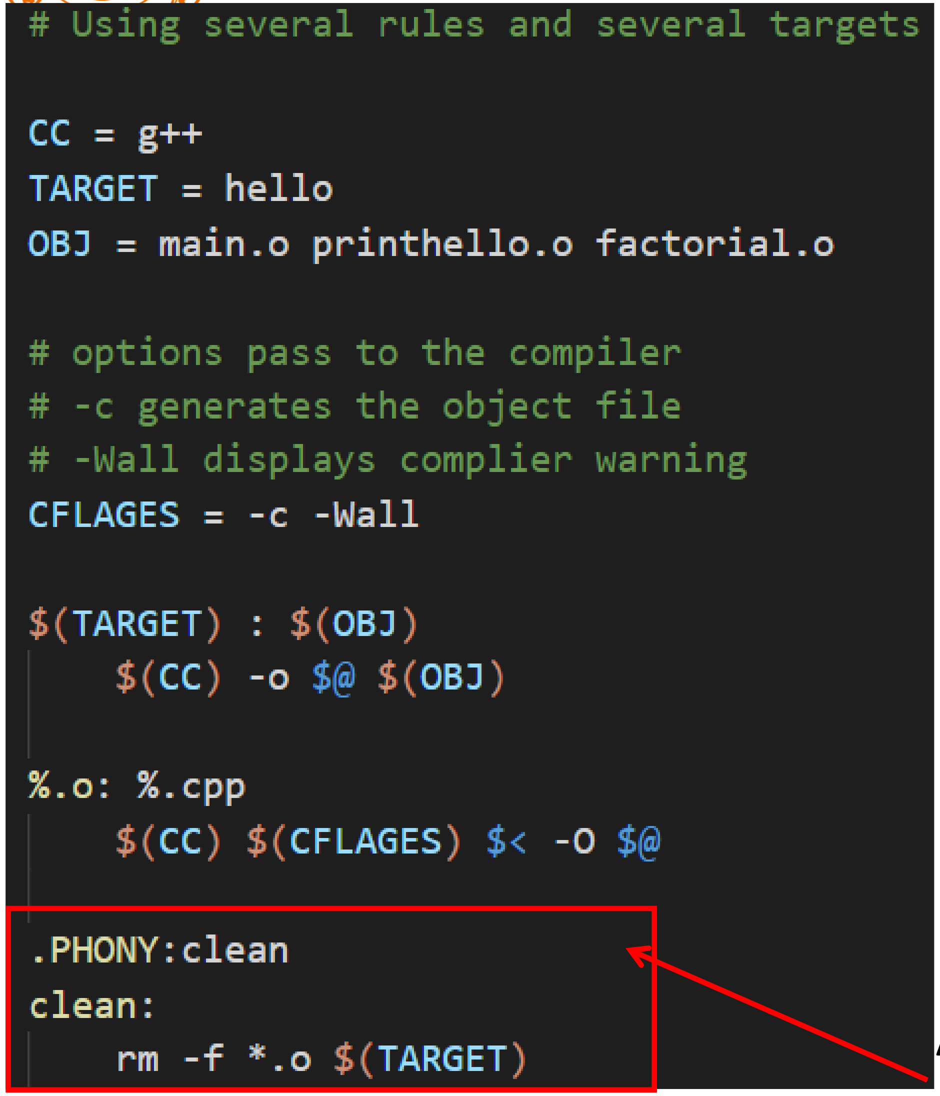
$^: all the prerequisites files

$<: the first prerequisite file



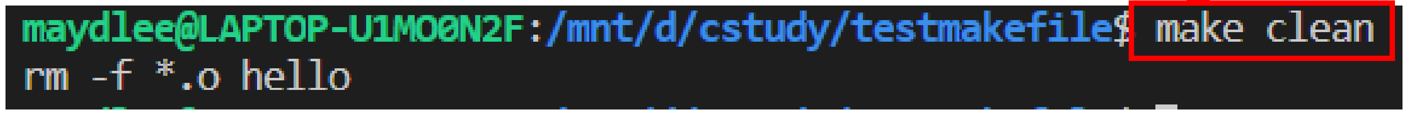
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ make
g++ -o hello main.o printhello.o factorial.o
```

Using phony target to clean up compiled results automatically



```
# Using several rules and several targets

CC = g++
TARGET = hello
OBJ = main.o printhello.o factorial.o

# options pass to the compiler
# -c generates the object file
# -Wall displays complier warning
CFLAGES = -c -Wall

$(TARGET) : $(OBJ)
    $(CC) -o $@ $(OBJ)

%.o: %.cpp
    $(CC) $(CFLAGES) $< -O $@

.PHONY:clean
clean:
    rm -f *.o $(TARGET)
```

Because **clean** is a label not a target, the command **make clean** can execute the clean part. Only **make** command can not execute clean part.



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ make clean
rm -f *.o hello
```

Adding **.PHONY** to a target will prevent making from confusing the phony target with a file name.

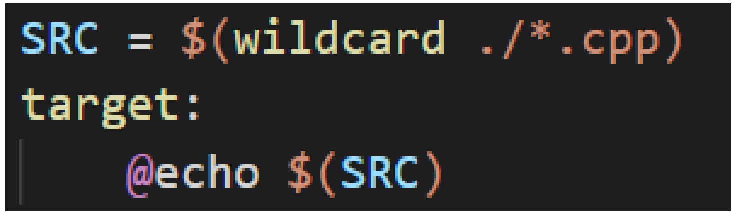
Functions in makefile

wildcard: search file

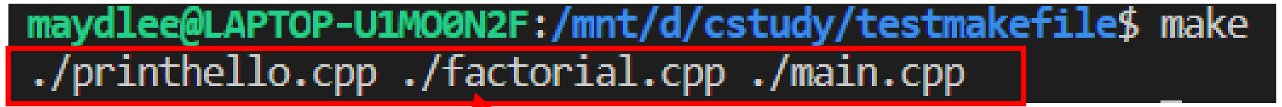
for example:

SRC = $(wildcard ./*.cpp)

Search all the .cpp files in the current directory, and return to SRC



```
SRC = $(wildcard ./*.cpp)
target:
	@echo $(SRC)
```



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ make
./printhello.cpp ./factorial.cpp ./main.cpp
```

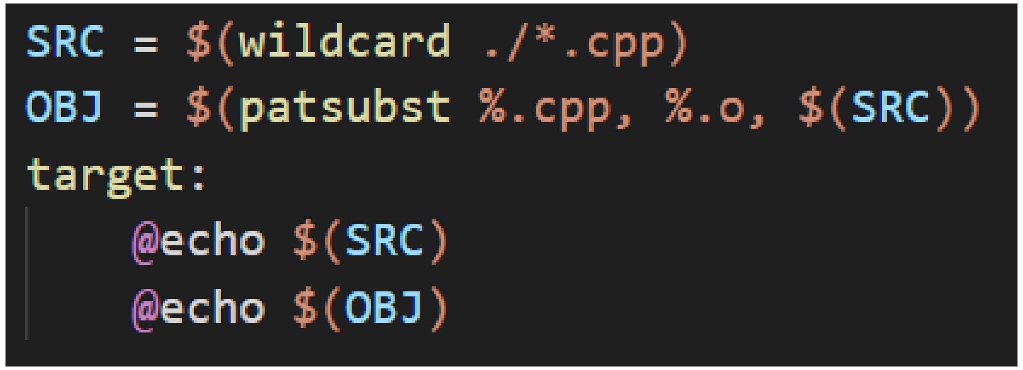
All .cpp files in the current directory

patsubst(pattern substitution): replace file
$(patsubst original pattern, target pattern, file list)

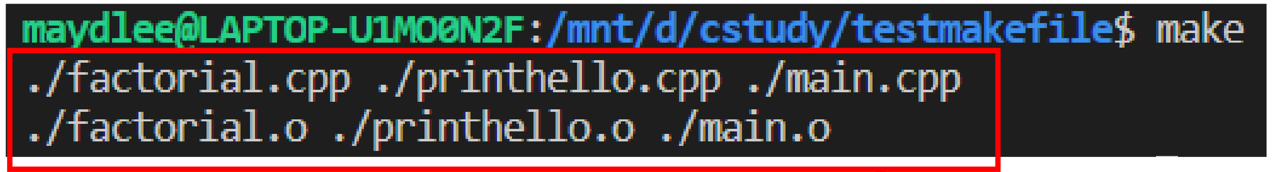
Replace all .cpp files with .o files

for example:

```
OBJ = $(patsubst  %.cpp,  %.o, $(SRC))
```

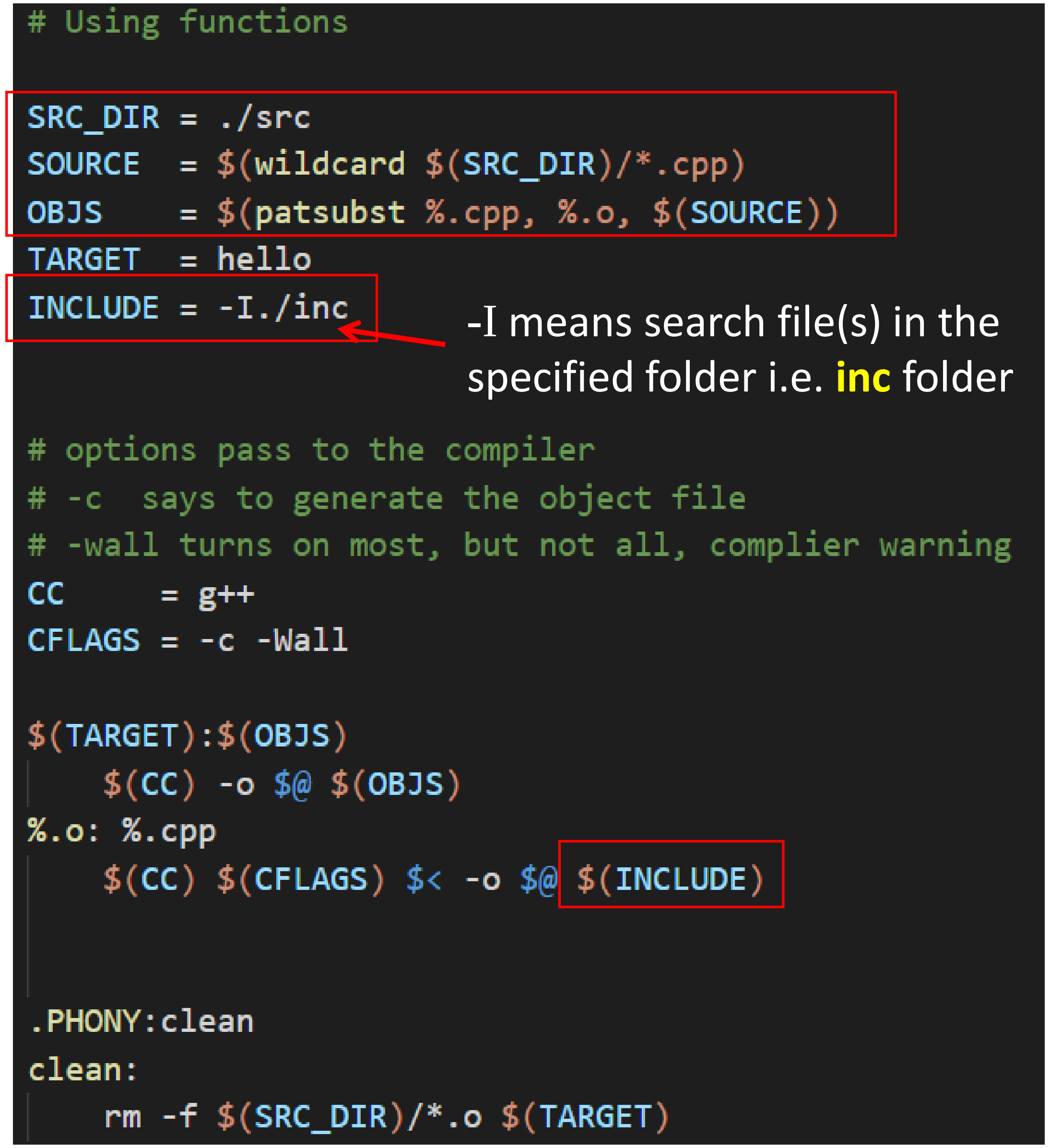


```
SRC = $(wildcard ./*.cpp)
OBJ = $(patsubst %.cpp, %.o, $(SRC))
target:
	@echo $(SRC)
	@echo $(OBJ)
```



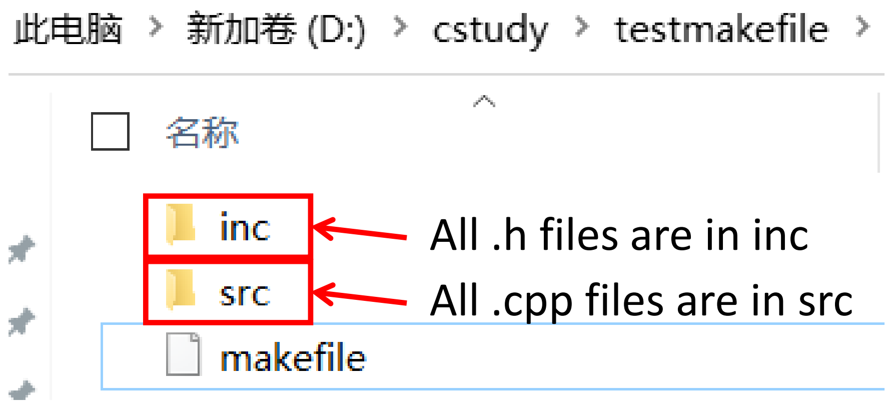
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ make
./factorial.cpp ./printhello.cpp ./main.cpp
./factorial.o ./printhello.o ./main.o
```

Replace all .cpp files with .o files



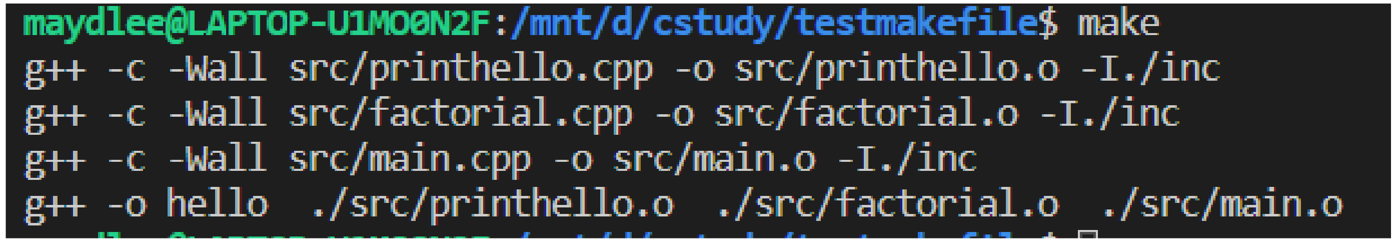
```
# Using functions

SRC_DIR = ./src
SOURCE  = $(wildcard $(SRC_DIR)/*.cpp)
OBJS    = $(patsubst %.cpp, %.o, $(SOURCE))
TARGET  = hello
INCLUDE = -I./inc


# options pass to the compiler
# -c  says to generate the object file
# -wall turns on most, but not all, complier warning
CC     = g++
CFLAGS = -c -Wall

$(TARGET):$(OBJS)
    $(CC) -o $@ $(OBJS)
%.o: %.cpp
    $(CC) $(CFLAGS) $< -o $@ $(INCLUDE)


.PHONY:clean
clean:
    rm -f $(SRC_DIR)/*.o $(TARGET)
```

-I means search file(s) in the specified folder i.e. **inc** folder



All .h files are in inc

All .cpp files are in src



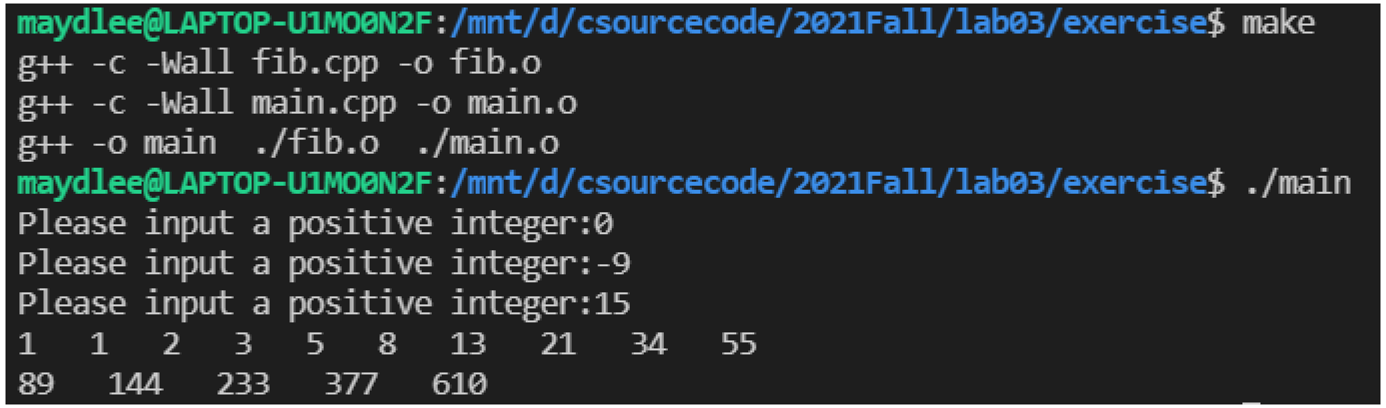
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ make
g++ -c -Wall src/printhello.cpp -o src/printhello.o -I./inc
g++ -c -Wall src/factorial.cpp -o src/factorial.o -I./inc
g++ -c -Wall src/main.cpp -o src/main.o -I./inc
g++ -o hello  ./src/printhello.o  ./src/factorial.o  ./src/main.o
```

GNU Make Manual

http://www.gnu.org/software/make/manual/make.html

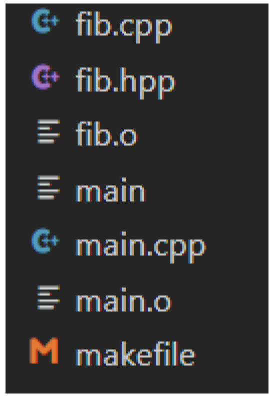
Exercises

1. The *Fibonacci numbers* are : 1,1,2,3,5,8……. Please define a function named **fib.cpp** to compute the *n*th Fibonacci number. In **main.cpp**, prompts the user to input an integer n, print Fibonacci numbers from 1 to n, 10 numbers per line. Write a **makefile** to manage the source files.

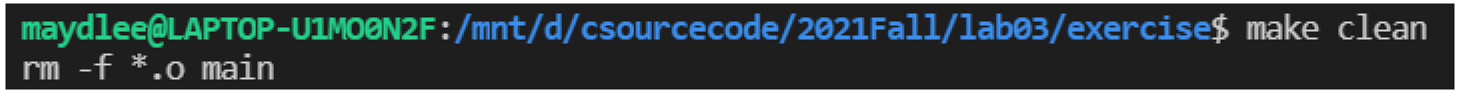


```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Fall/lab03/exercise$ make
g++ -c -Wall fib.cpp -o fib.o
g++ -c -Wall main.cpp -o main.o
g++ -o main  ./fib.o  ./main.o
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Fall/lab03/exercise$ ./main
Please input a positive integer:0
Please input a positive integer:-9
Please input a positive integer:15
1    1    2    3    5    8    13    21    34    55
89    144    233    377    610
```

Before clean:

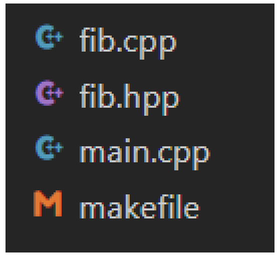


```
fib.cpp
fib.hpp
fib.o
main
main.cpp
main.o
makefile
```



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Fall/lab03/exercise$ make clean
rm -f *.o main
```

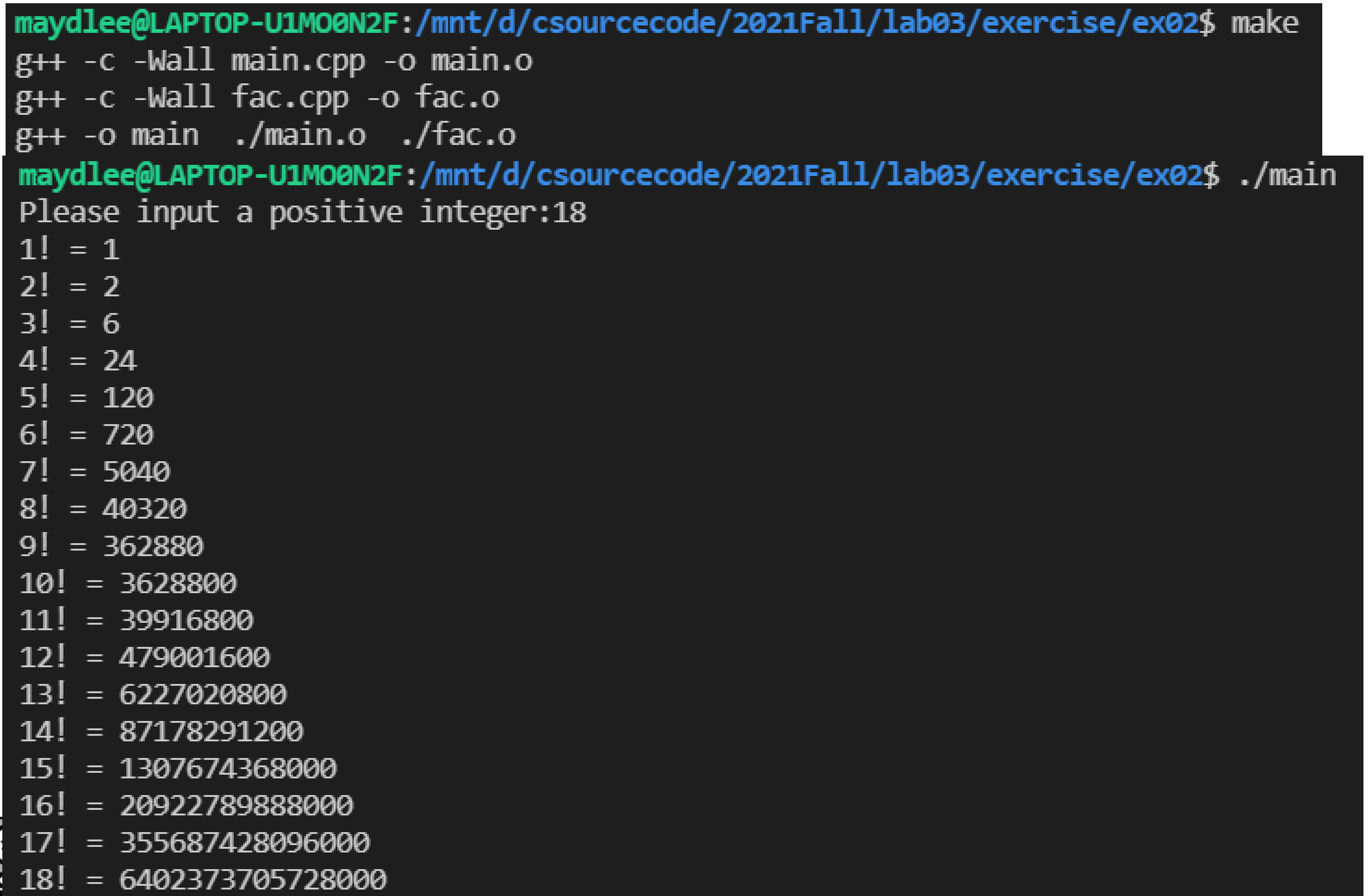
After clean:



```
fib.cpp
fib.hpp
main.cpp
makefile
```

Exercises

2. Define a function named **fac.cpp** to compute the factorial of an integer. In **main.cpp**, prompts the user to input an integer n, print factorials from 1 to n, one factorial per line. Write a **makefile** to manage the source files.



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Fall/lab03/exercise/ex02$ make
g++ -c -Wall main.cpp -o main.o
g++ -c -Wall fac.cpp -o fac.o
g++ -o main  ./main.o  ./fac.o
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Fall/lab03/exercise/ex02$ ./main
Please input a positive integer:18
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
```

If you input an integer that is greater than 20, how about the result? Is that correct? How to fix the error?