C/C++ Program Design

Lab 4, Makefile

廖琪梅, 王大兴, 于仕琪

Makefile

What is a Makefile?

Makefile is a tool to simplify and organize 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 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**.

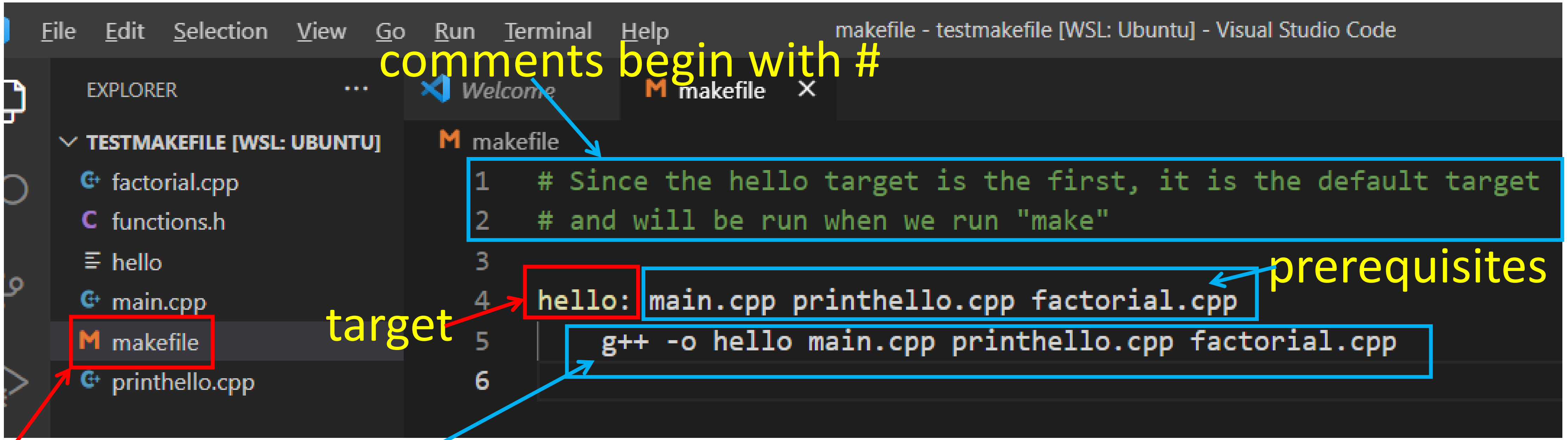
```
targets :  prerequisites

<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.



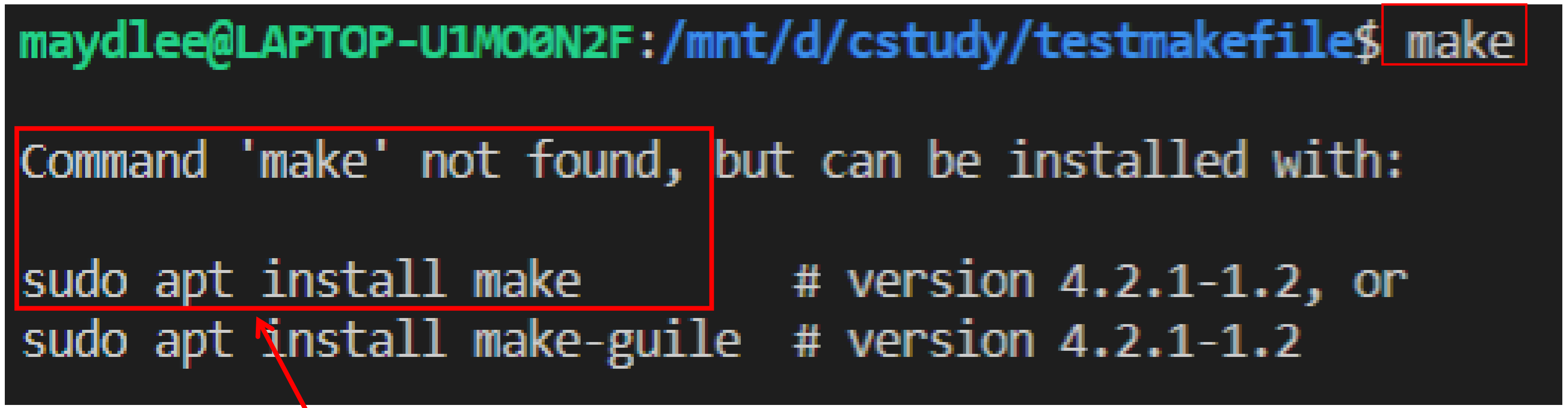
```
# Since the hello target is the first, it is the default target
# and will be run when we run "make"

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

Put the makefile together with your programs.

commands

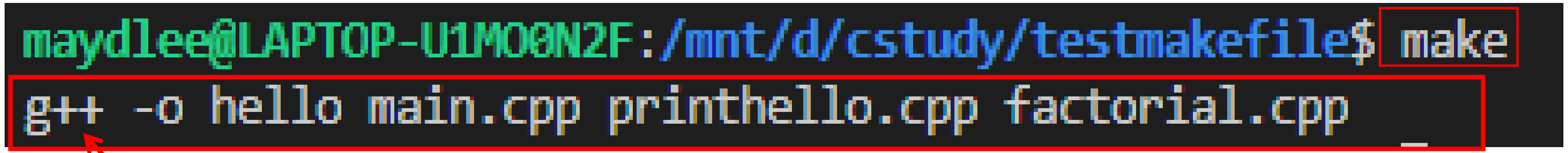
g++ is compiler name, -o is linker flag and hello is binary file name.

Type the command make in VScode



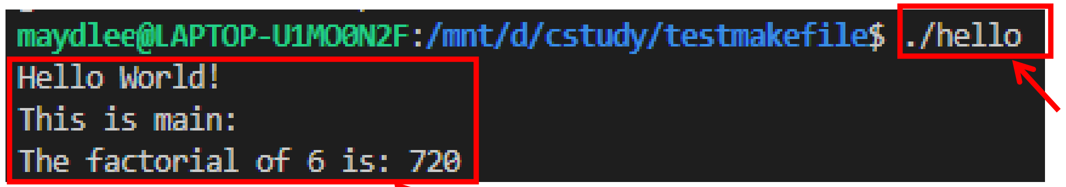
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ make

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.



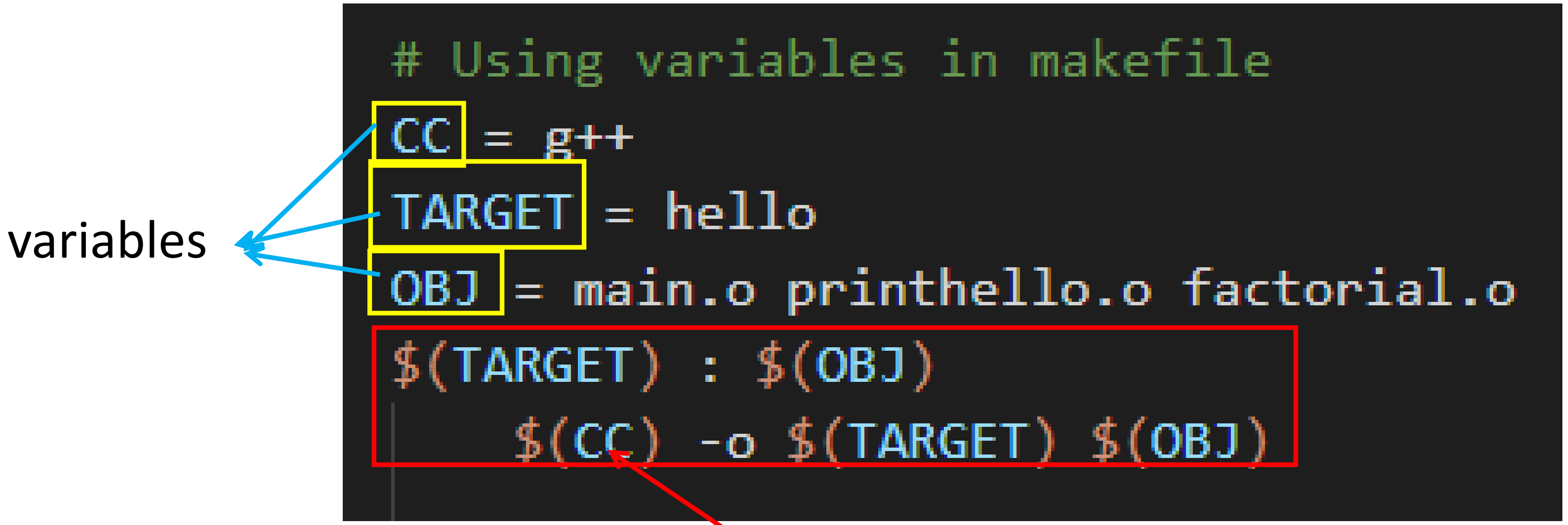
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ ./hello
Hello World!
This is main:
The factorial of 6 is: 720
```

Run your program

output

Defining Macros/Variables in the makefile

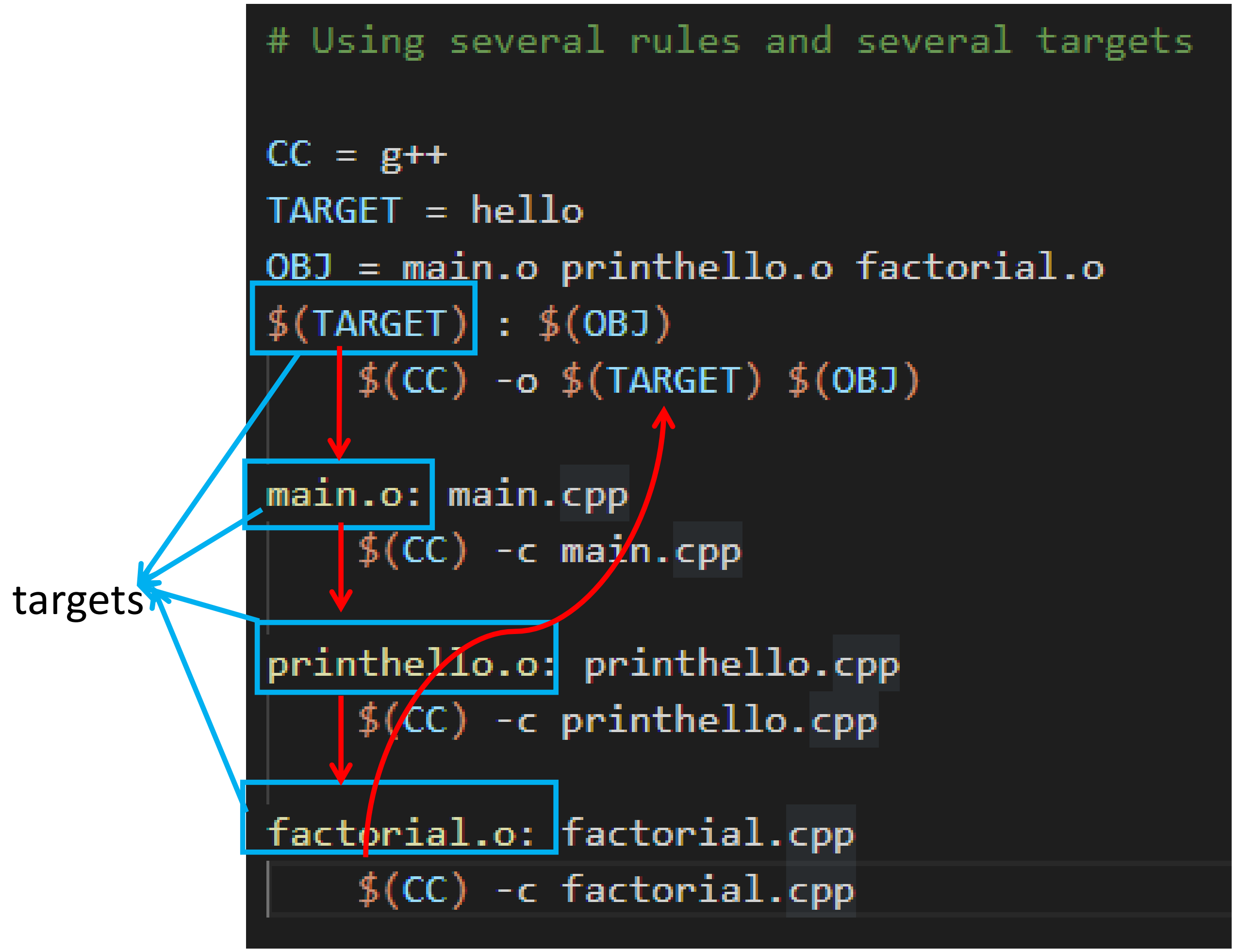
To improve the efficiency of the makefile, we use variables.



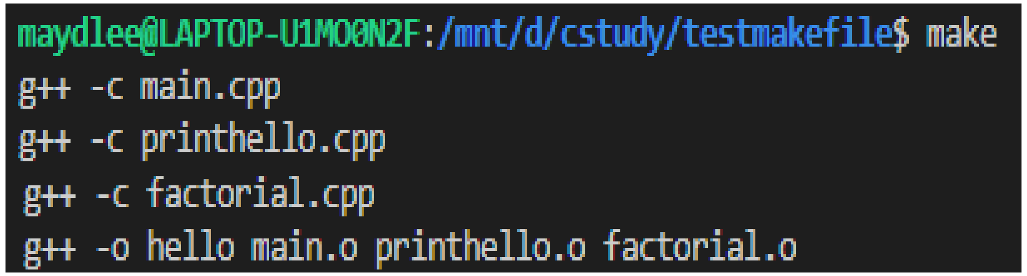
```
# Using variables in makefile
CC = g++
TARGET = hello
OBJ = main.o printhello.o factorial.o
$(TARGET) : $(OBJ)
	$(CC) -o $(TARGET) $(OBJ)
```

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.

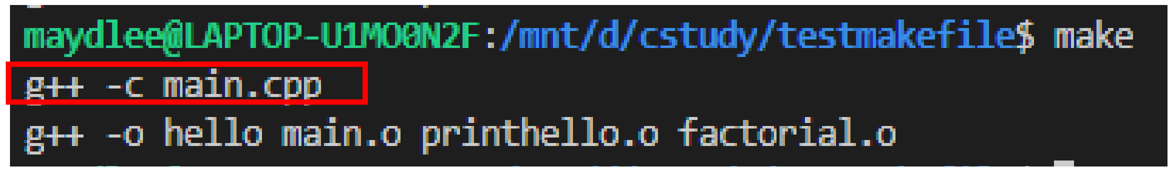


```
# Using several rules and several targets

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
```



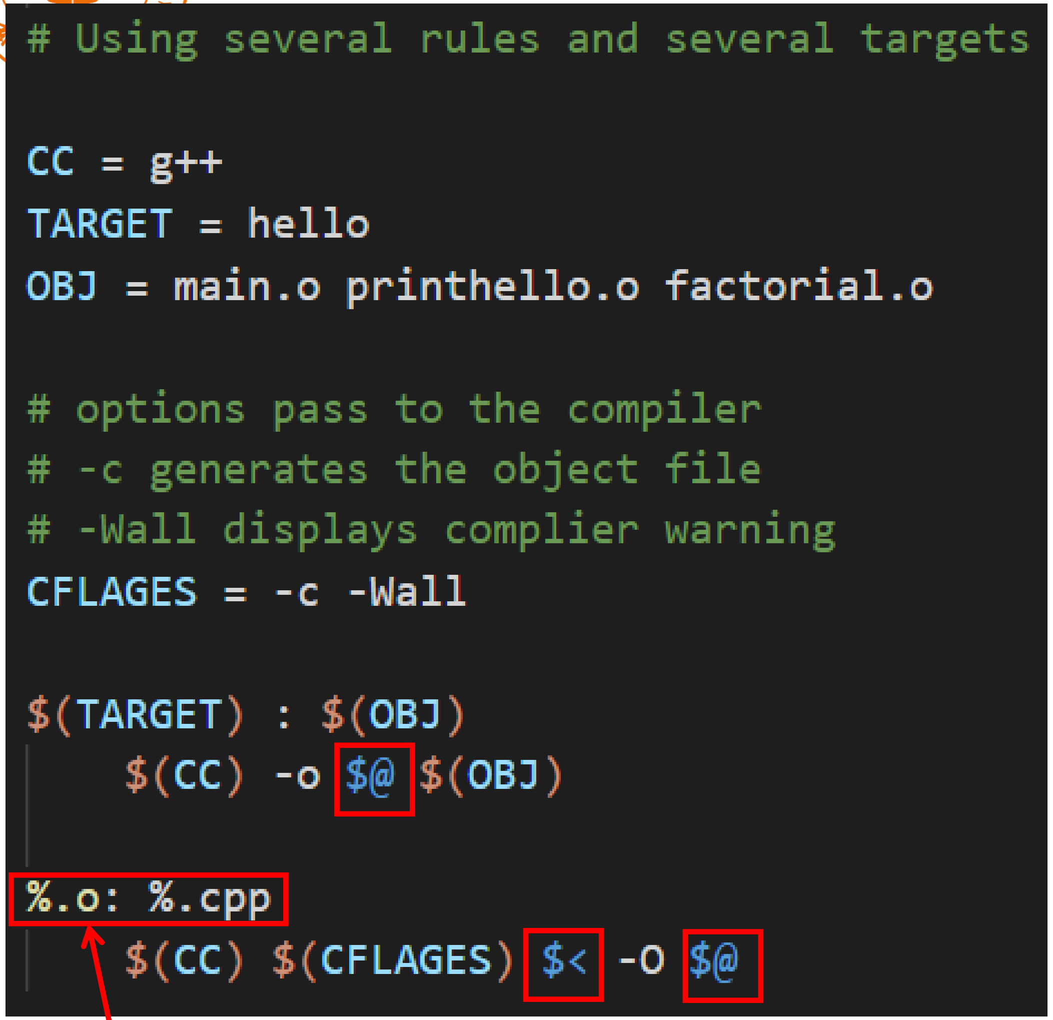
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ make
g++ -c main.cpp
g++ -c printhello.cpp
g++ -c factorial.cpp
g++ -o hello main.o printhello.o factorial.o
```

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:



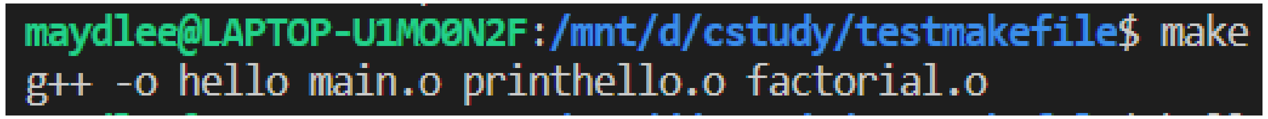
```
# Using several rules and several targets

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

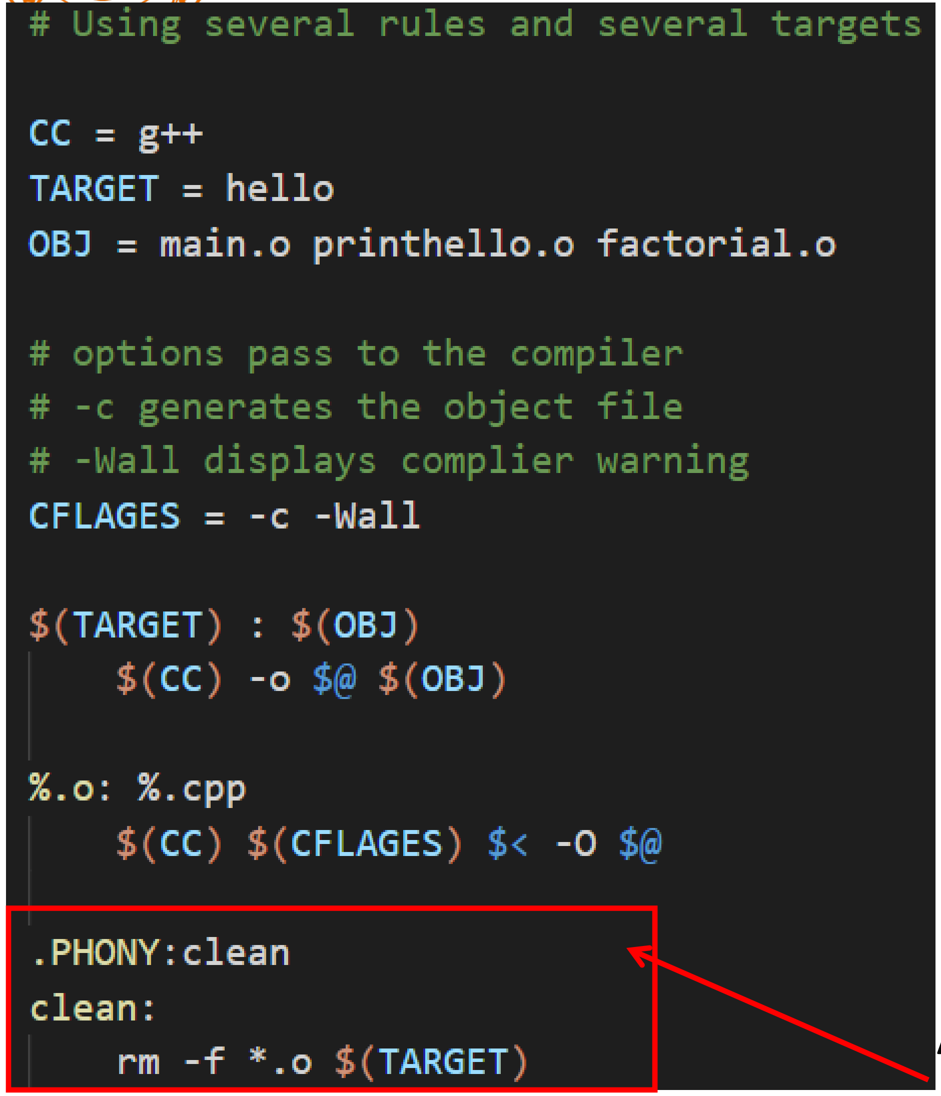
$^: all the prerequisites files

$<: the first prerequisite file



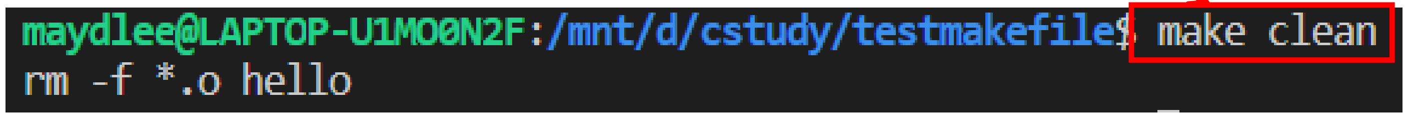
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ make
g++ -o hello main.o printhello.o factorial.o
```

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.

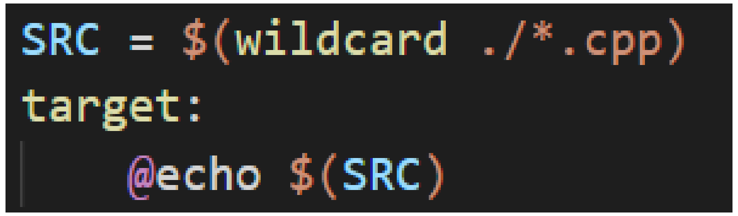
Functions in makefile

wildcard: search file

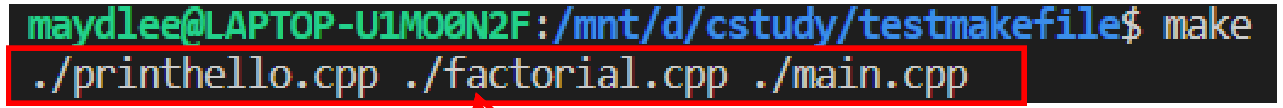
for example:

```
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
```

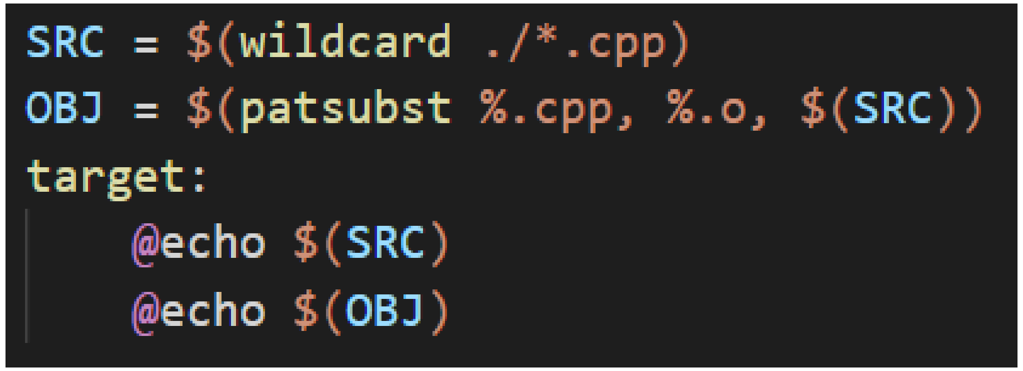
All .cpp files in the current directory

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

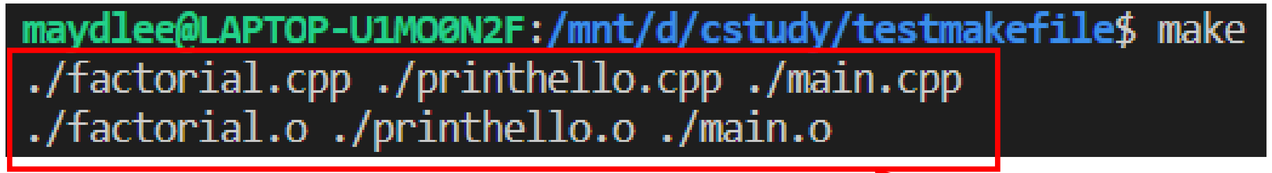
Replace all .cpp files with .o files

for example:

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

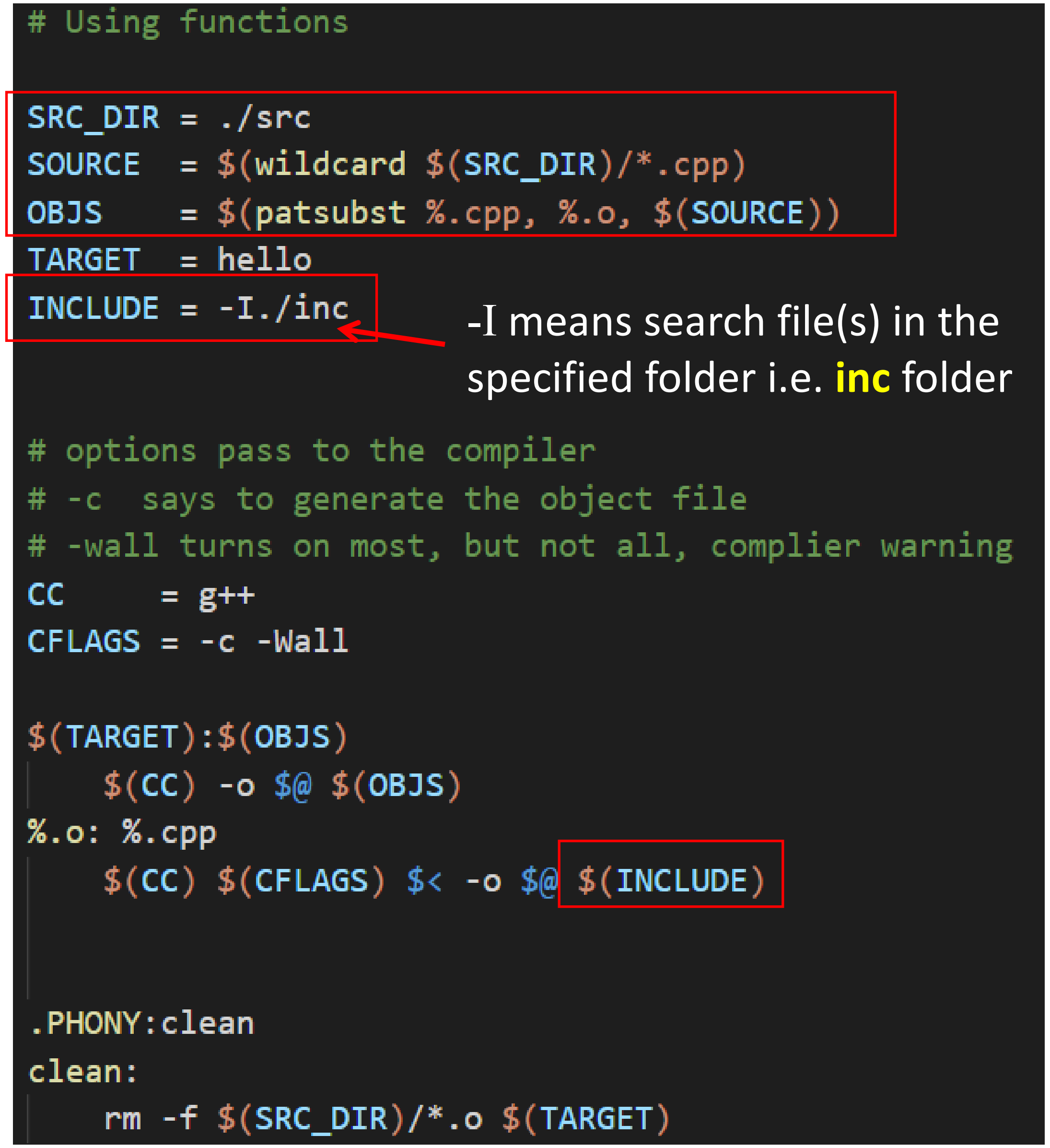


```
SRC = $(wildcard ./*.cpp)
OBJ = $(patsubst %.cpp, %.o, $(SRC))
target:
	@echo $(SRC)
	@echo $(OBJ)
```



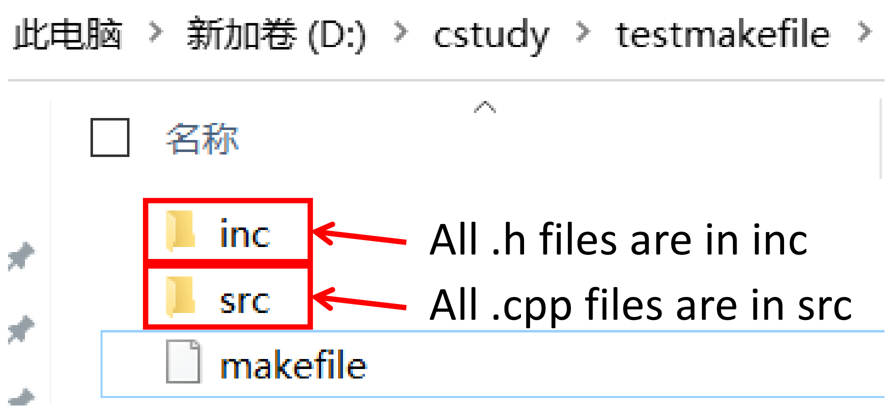
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ make
./factorial.cpp ./printhello.cpp ./main.cpp
./factorial.o ./printhello.o ./main.o
```

Replace all .cpp files with .o files



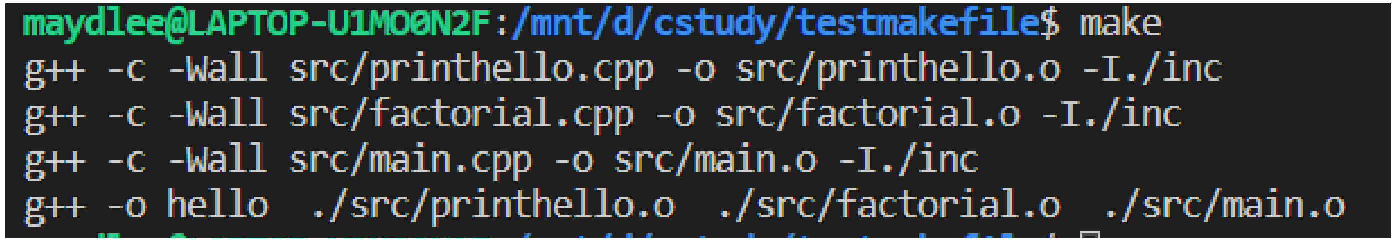
```
# Using functions

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



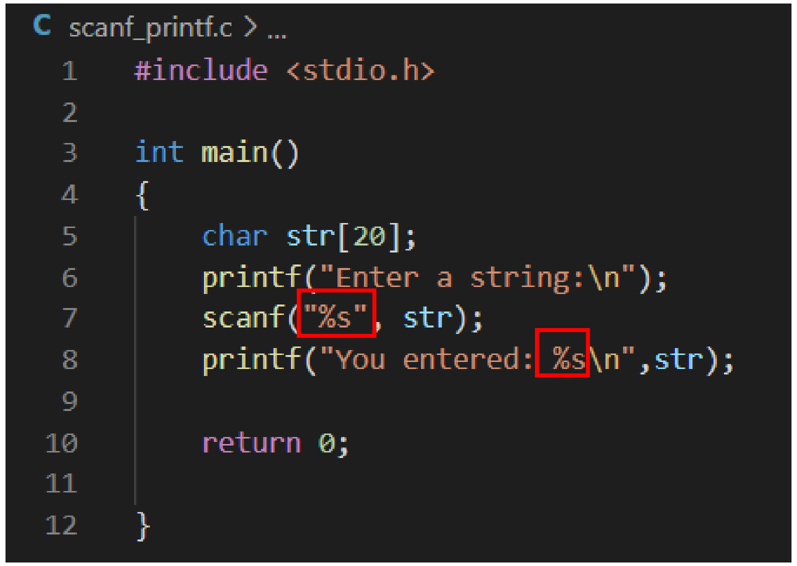
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/testmakefile$ make
g++ -c -Wall src/printhello.cpp -o src/printhello.o -I./inc
g++ -c -Wall src/factorial.cpp -o src/factorial.o -I./inc
g++ -c -Wall src/main.cpp -o src/main.o -I./inc
g++ -o hello  ./src/printhello.o  ./src/factorial.o  ./src/main.o
```

GNU Make Manual

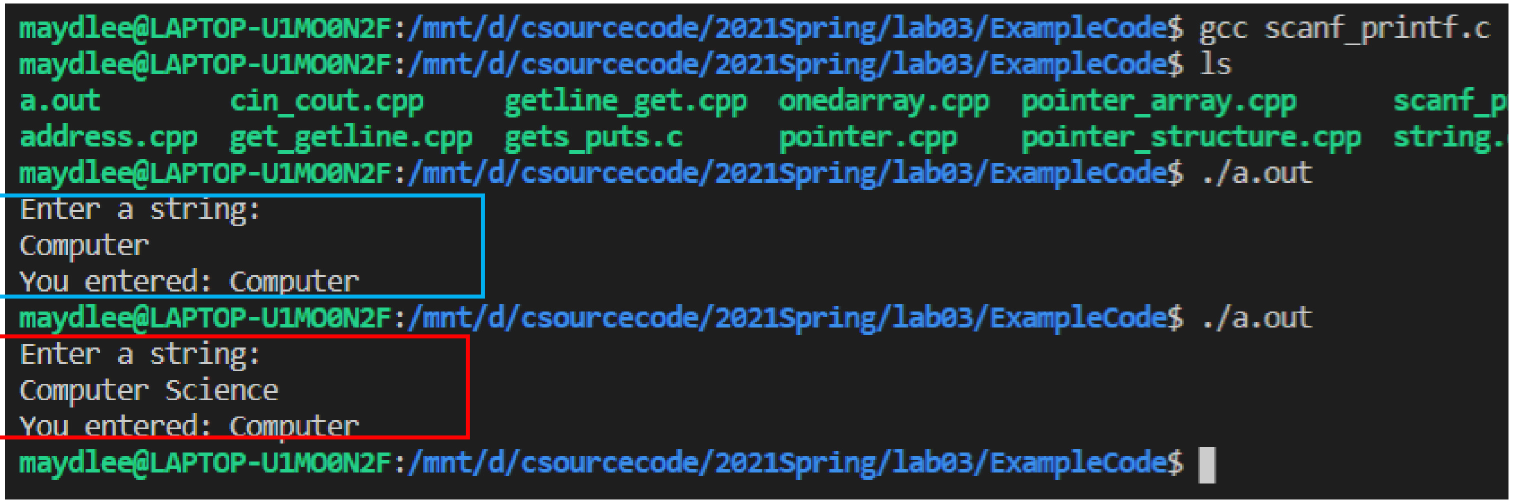
http://www.gnu.org/software/make/manual/make.html

Keyboard input and terminal output of string

1. C: scanf & printf
   - %d ----int
   - %f ----float
   - %c -----char
   - %s -----string



```
scanf_printf.c > ...
#include <stdio.h>

int main()
{
    char str[20];
    printf("Enter a string:\n");
    scanf("%s", str);
    printf("You entered: %s\n",str);

    return 0;

}
```

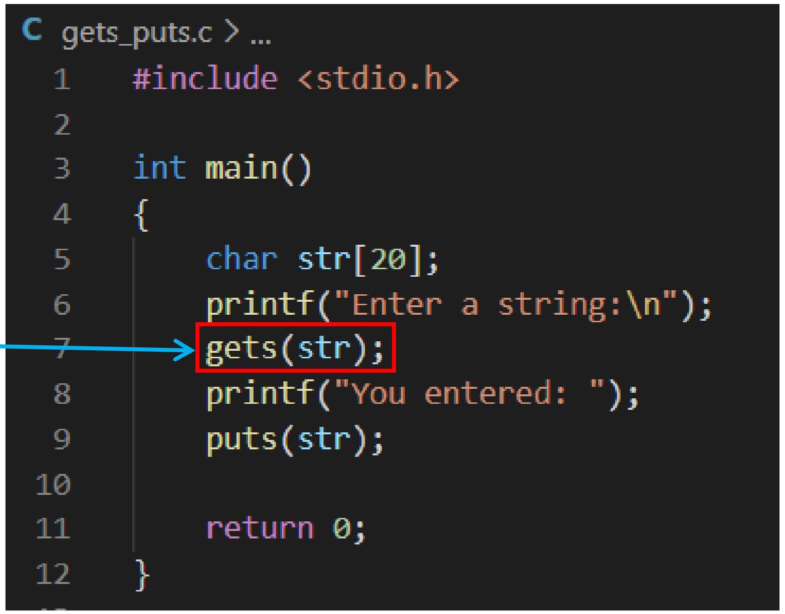


```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ gcc scanf_printf.c
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ ls
a.out         cin_cout.cpp     getline_get.cpp  onedarray.cpp  pointer_array.cpp      scanf_p
address.cpp   get_getline.cpp  gets_puts.c      pointer.cpp    pointer_structure.cpp  string.
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ ./a.out
Enter a string:
Computer
You entered: Computer
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ ./a.out
Enter a string:
Computer Science
You entered: Computer
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$
```

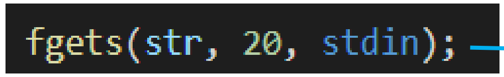
Why only Computer?

scanf uses whitespace—spaces, tabs, and newlines to separate a string.

2. C: gets & puts

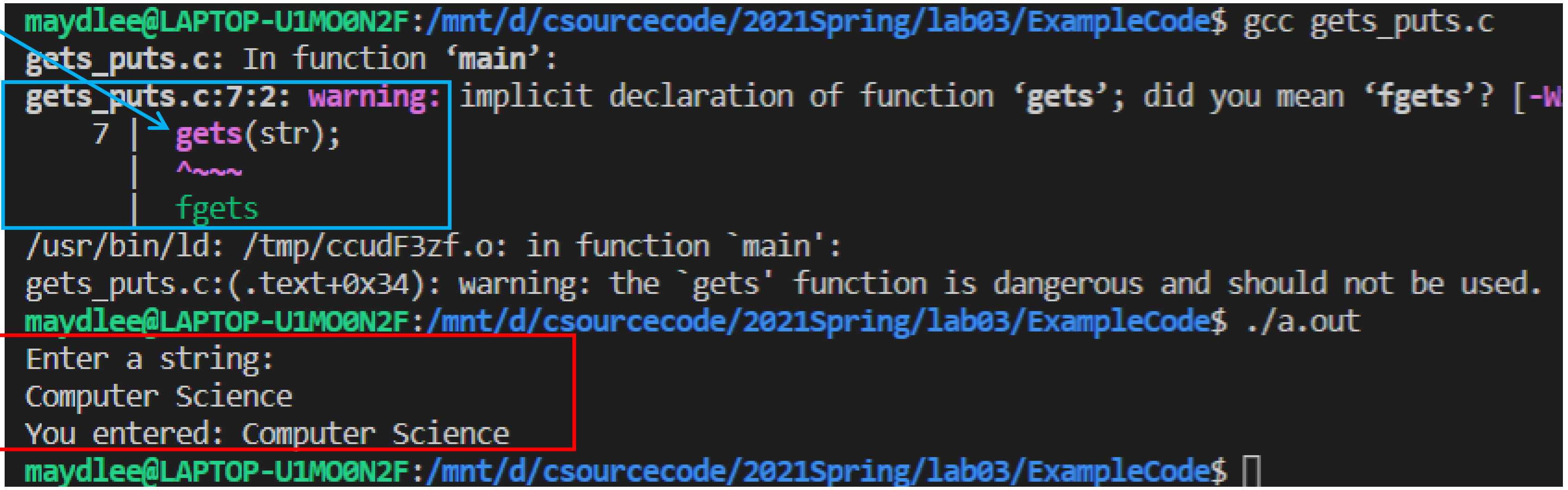


```
C gets_puts.c > ...
 1  #include <stdio.h>
 2
 3  int main()
 4  {
 5      char str[20];
 6      printf("Enter a string:\n");
 7      gets(str);
 8      printf("You entered: ");
 9      puts(str);
10
11      return 0;
12  }
```



```
fgets(str, 20, stdin);
```

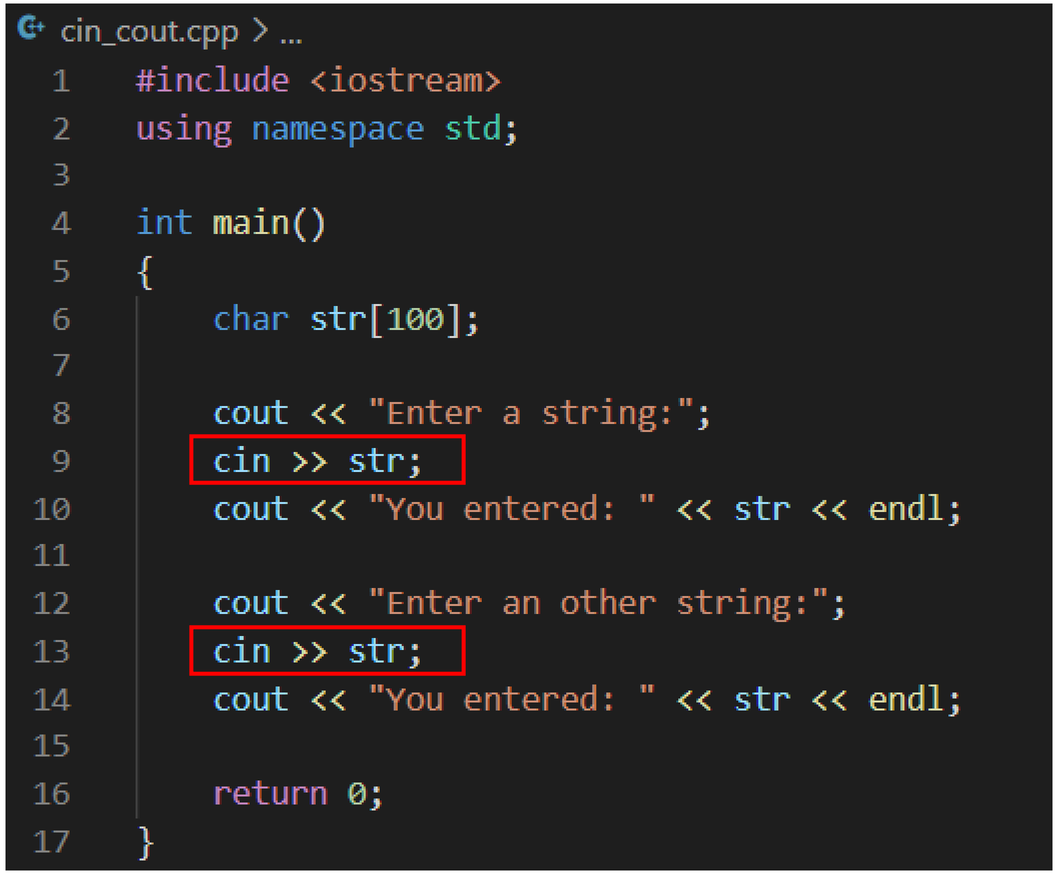
There is a warning due to using gets().
You can use fgets() function instead.



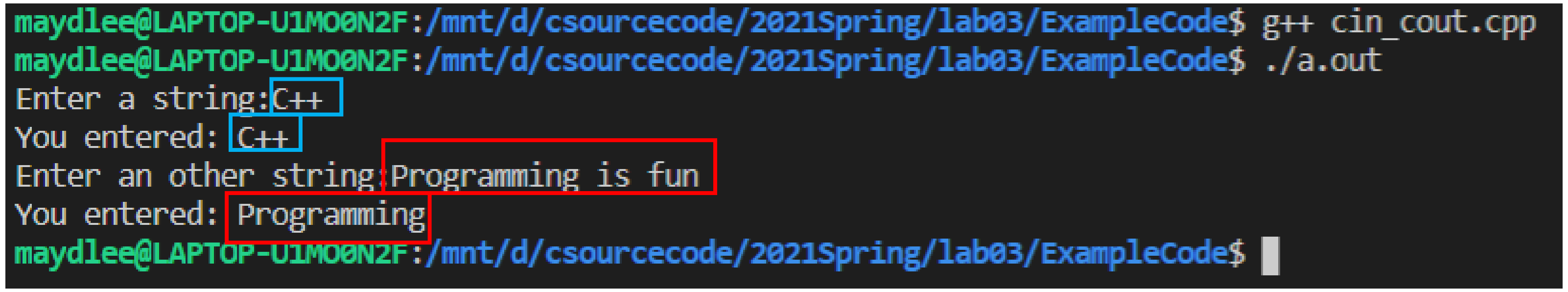
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ gcc gets_puts.c
gets_puts.c: In function ‘main’:
gets_puts.c:7:2: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-W
    7 |  gets(str);
      |  ^~~~
      |  fgets
/usr/bin/ld: /tmp/ccudF3zf.o: in function `main':
gets_puts.c:(.text+0x34): warning: the `gets' function is dangerous and should not be used.
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ ./a.out
Enter a string:
Computer Science
You entered: Computer Science
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ 
```

Use gets to gain a sentence with a space.
gets() stops reading input when it encounters a newline or end of file.

3. C++: cin & cout



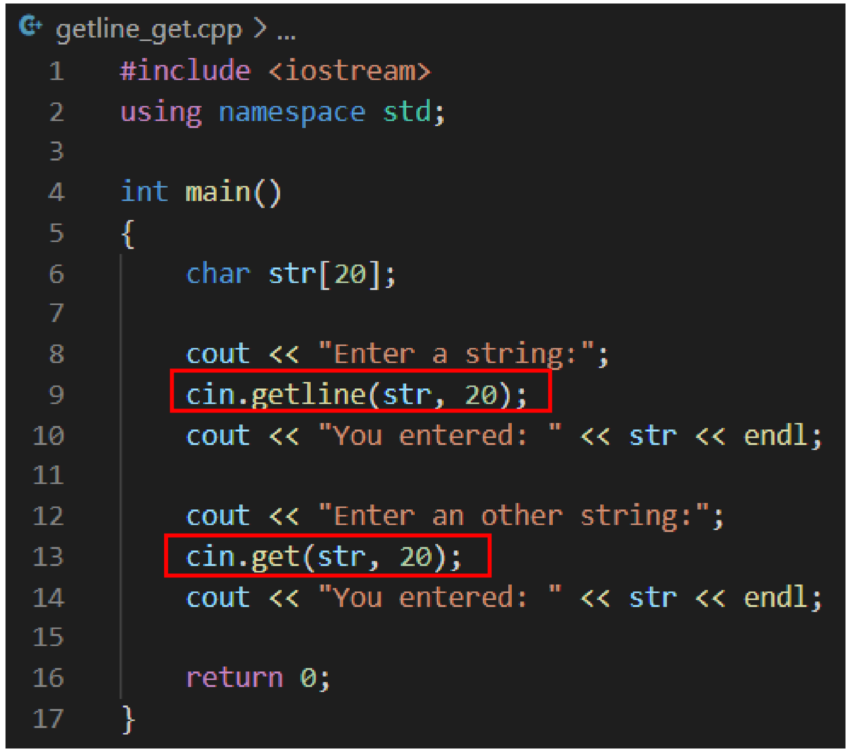
```
cin_cout.cpp > ...
#include <iostream>
using namespace std;

int main()
{
    char str[100];

    cout << "Enter a string:";
    cin >> str;
    cout << "You entered: " << str << endl;

    cout << "Enter an other string:";
    cin >> str;
    cout << "You entered: " << str << endl;

    return 0;
}
```



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ g++ cin_cout.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ ./a.out
Enter a string:C++
You entered: C++
Enter an other string:Programming is fun
You entered: Programming
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$
```

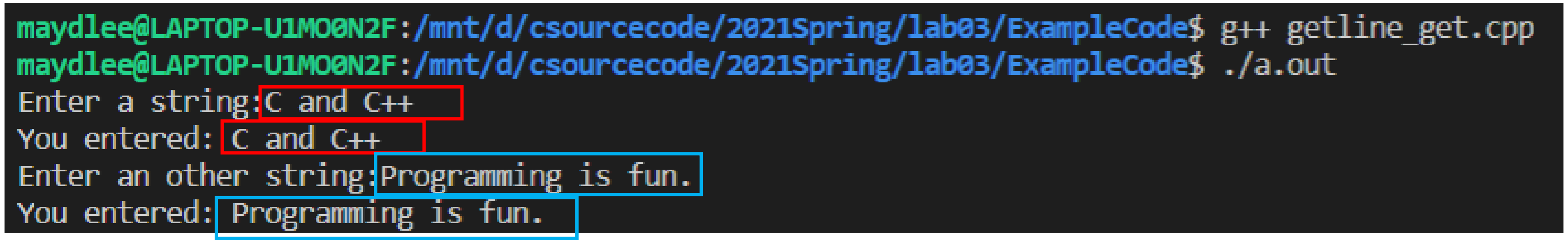
The cin is to use whitespace-- spaces, tabs, and newlines to separate a string.

4. C++: cin.getline( ) & cin.get( )



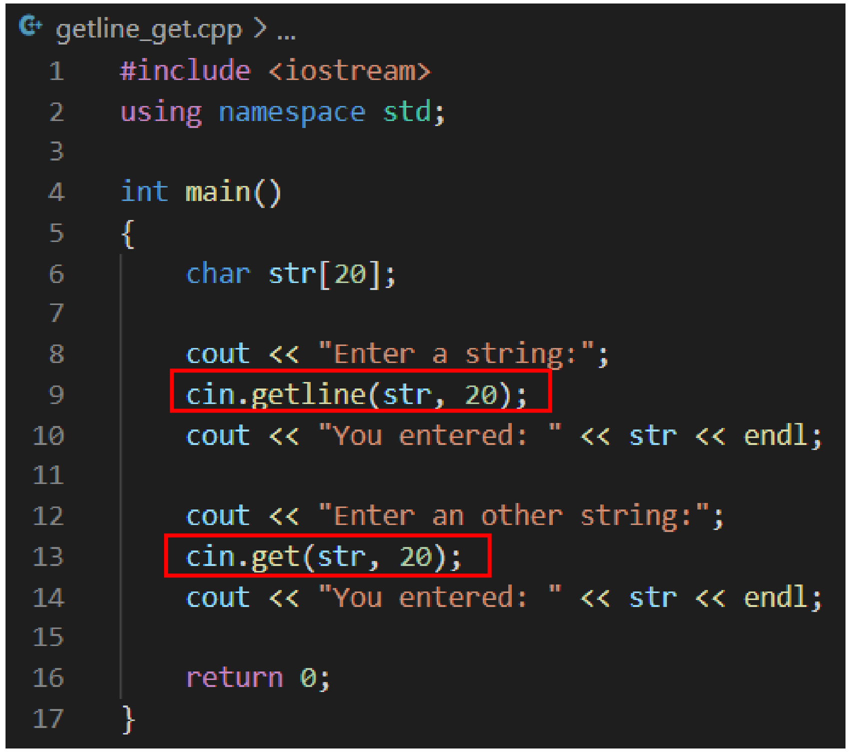
getline_get.cpp > ...

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

int main()
{
    char str[20];

    cout << "Enter a string:";
    cin.getline(str, 20);
    cout << "You entered: " << str << endl;

    cout << "Enter an other string:";
    cin.get(str, 20);
    cout << "You entered: " << str << endl;

    return 0;
}
```

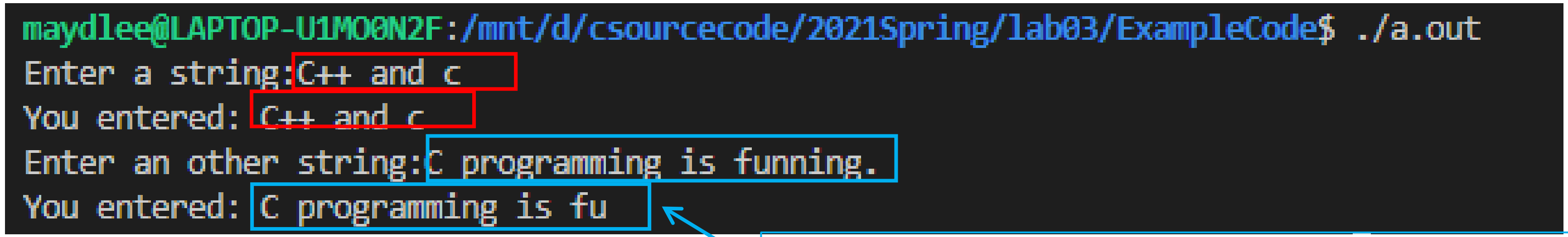


```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ g++ getline_get.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ ./a.out
Enter a string:C and C++
You entered: C and C++
Enter an other string:Programming is fun.
You entered: Programming is fun.
```

4. C++: cin.getline( ) & cin.get( )



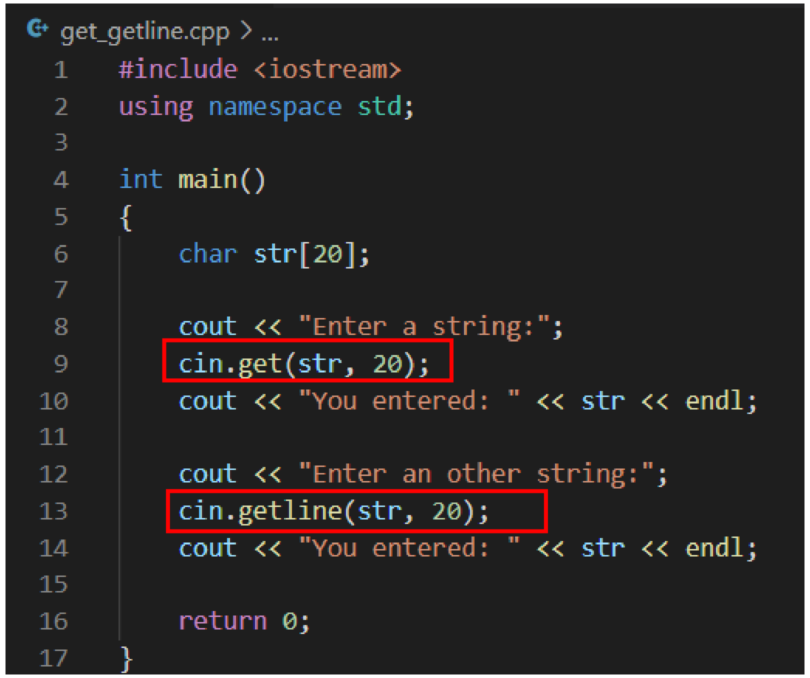
```
getline_get.cpp > ...
#include <iostream>
using namespace std;

int main()
{
    char str[20];

    cout << "Enter a string:";
    cin.getline(str, 20);
    cout << "You entered: " << str << endl;

    cout << "Enter an other string:";
    cin.get(str, 20);
    cout << "You entered: " << str << endl;

    return 0;
}
```



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ ./a.out
Enter a string:C++ and c
You entered: C++ and c
Enter an other string:C programming is funning.
You entered: C programming is fu
```

If the length of input string is greater than 20, it can only store first 19 characters in str.

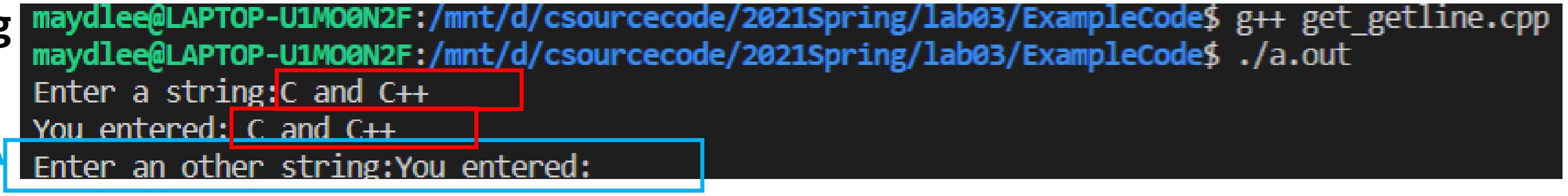
4. C++: cin.getline( ) & cin.get( )



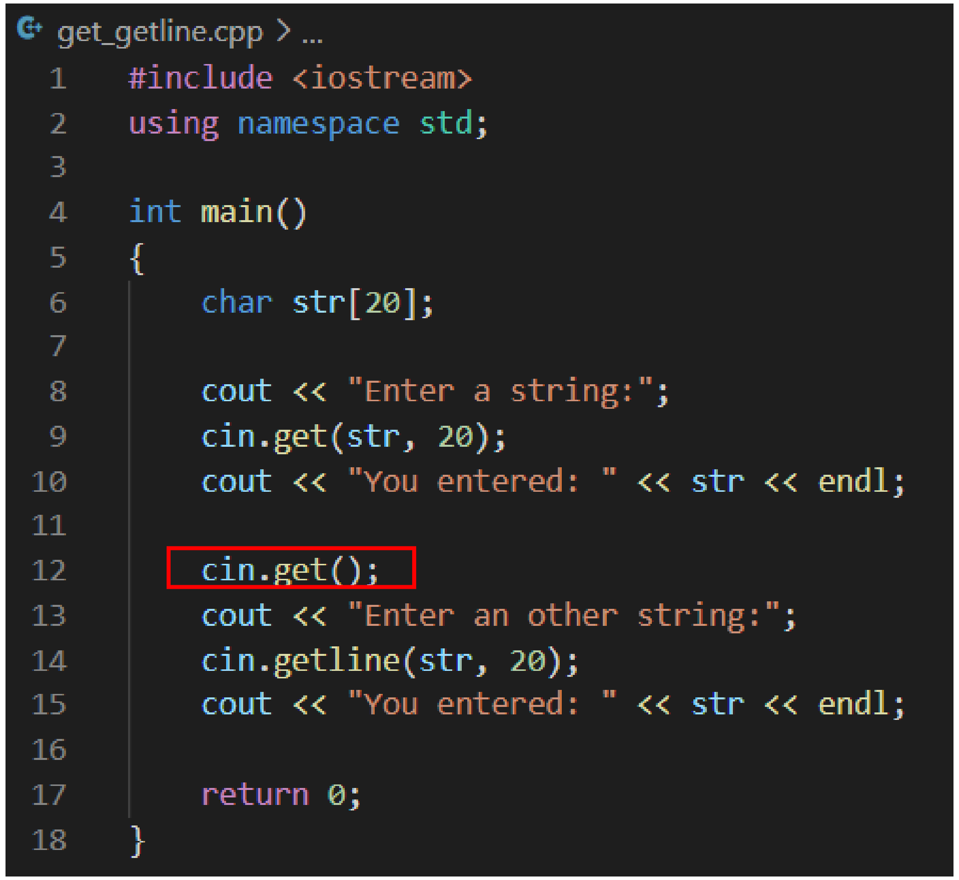
```
get_getline.cpp > ...
#include <iostream>
using namespace std;

int main()
{
    char str[20];

    cout << "Enter a string:";
    cin.get(str, 20);
    cout << "You entered: " << str << endl;

    cout << "Enter an other string:";
    cin.getline(str, 20);
    cout << "You entered: " << str << endl;

    return 0;
}
```

getline() and get() both read an entire input line—that is, up until a newline character. However, getline() discard the newline character, whereas get() leave it in the input queue.

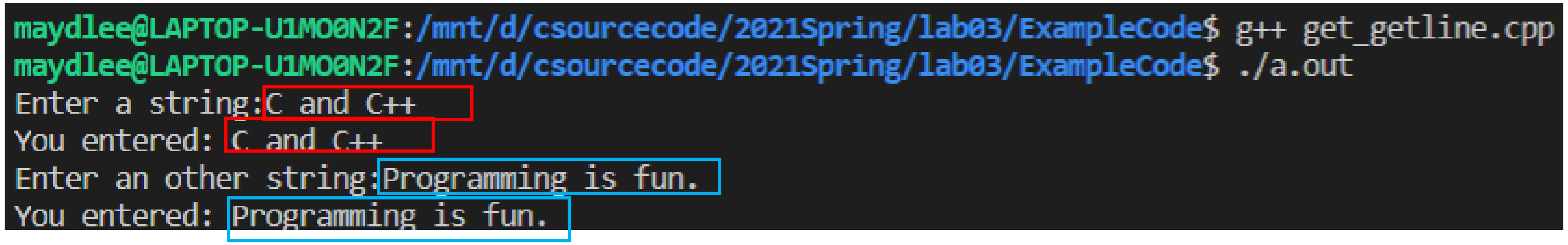
Program runs without entering another string



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ g++ get_getline.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ ./a.out
Enter a string:C and C++
You entered: C and C++
Enter an other string:You entered:
```

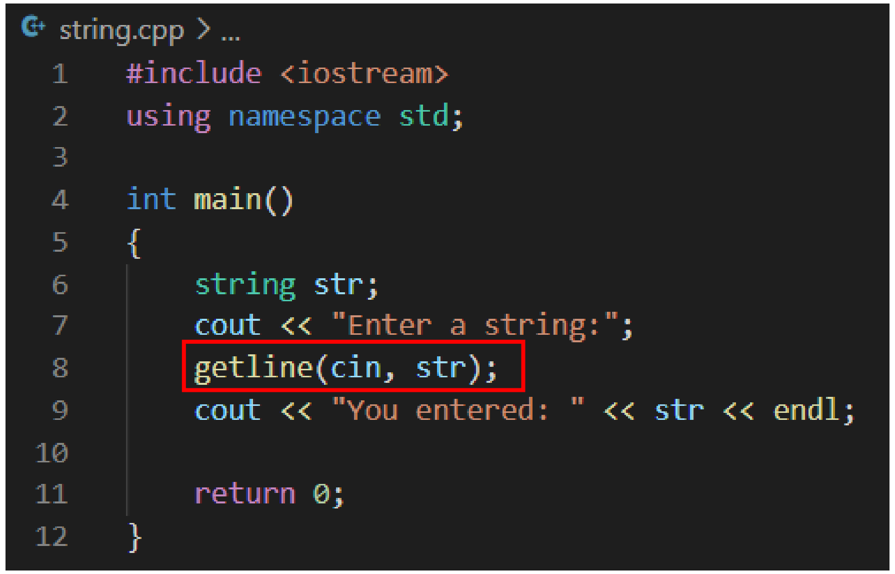


```cpp
get_getline.cpp > ...
#include <iostream>
using namespace std;

int main()
{
    char str[20];

    cout << "Enter a string:";
    cin.get(str, 20);
    cout << "You entered: " << str << endl;

    cin.get();
    cout << "Enter an other string:";
    cin.getline(str, 20);
    cout << "You entered: " << str << endl;

    return 0;
}
```



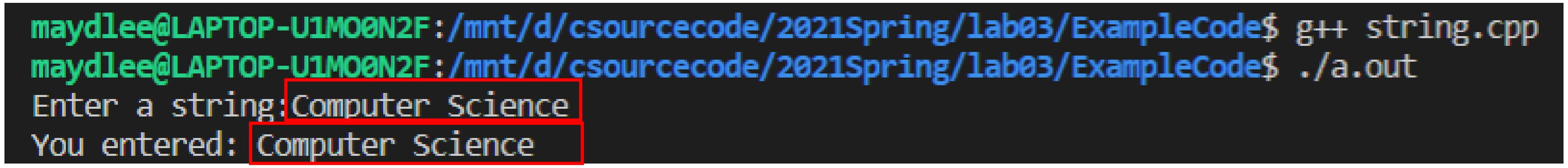
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ g++ get_getline.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ ./a.out
Enter a string:C and C++
You entered: C and C++
Enter an other string:Programming is fun.
You entered: Programming is fun.
```

C++ string using string data type



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

int main()
{
    string str;
    cout << "Enter a string:";
    getline(cin, str);
    cout << "You entered: " << str << endl;

    return 0;
}
```

getline() function takes the input stream as the first parameter which is cin and str as the location of the line to be stored.



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ g++ string.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab03/ExampleCode$ ./a.out
Enter a string:Computer Science
You entered: Computer Science
```

Exercise 1

```
#include <iostream>
#include <string.h>
using namespace std;

int main()
{
   int cards[4]{};
   int hands[4];

   int price[] = {2.8,3.7,5,9};
   char direction[4] {'L',82,'U',68};

   char title[] = "ChartGPT is an awesome tool.";

   cout << "sizeof(cards) = " << sizeof(cards) << ",sizeof of cards[0] = " << sizeof(cards[0]) << endl;
   cout << "sizeof(price) = " << sizeof(price) << ",sizeof of price[0] = " << sizeof(price[1]) << endl;
   cout << "sizeof(direction) = " << sizeof(direction) << ",length of direction = " << strlen(direction) << endl;
   cout << "sizeof(title) = " << sizeof(title) << ",length of title = " << strlen(title) << endl;

   //Print the value and address of each element in cards and hands respectively.
   …….

   return 0;
}
```

First, complete the code, then run the program and explain the result to SA. If it has bugs, fix them.

Exercise 2

```
#include <stdio.h>

union data{
    int n;
    char ch;
    short m;
};

int main()
{
    union data a;
    printf("%d, %d\n", sizeof(a), sizeof(union data) );
    a.n = 0x40;
    printf("%X, %c, %hX\n", a.n, a.ch, a.m);
    a.ch = '9';
    printf("%X, %c, %hX\n", a.n, a.ch, a.m);
    a.m = 0x2059;
    printf("%X, %c, %hX\n", a.n, a.ch, a.m);
    a.n = 0x3E25AD54;
    printf("%X, %c, %hX\n", a.n, a.ch, a.m);

    return 0;
}
```

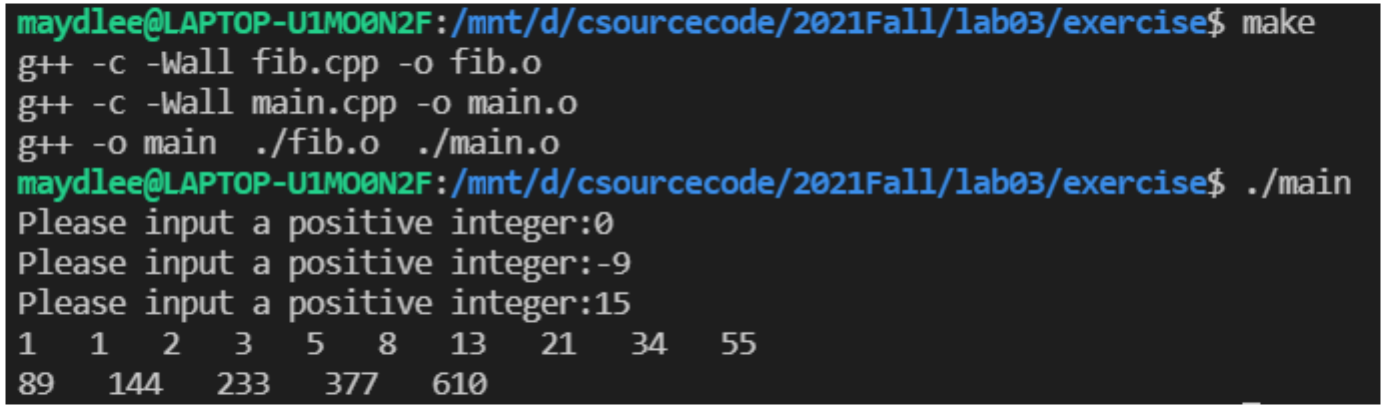
Run the program and explain the result to SA. You can write a program to check whether you system is little-endian or big-endian.

Exercise 3

- Design a struct “DayInfo” which contains two enumeration types as its member. The first is an enum “Day” for (Sunday, Monday, ...), and the second is an enum “Weather” for (Sunny, Rainy, ...).
- Define a boolean function “bool canTravel(DayInfo)” . It will return true if the day is at weekend and the weather is good.
- Call function canTravel() in main().

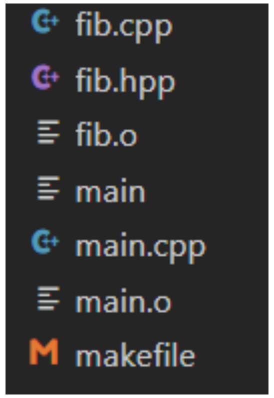
Exercise 4

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



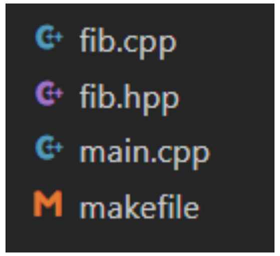
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Fall/lab03/exercise$ make
g++ -c -Wall fib.cpp -o fib.o
g++ -c -Wall main.cpp -o main.o
g++ -o main  ./fib.o  ./main.o
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Fall/lab03/exercise$ ./main
Please input a positive integer:0
Please input a positive 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:

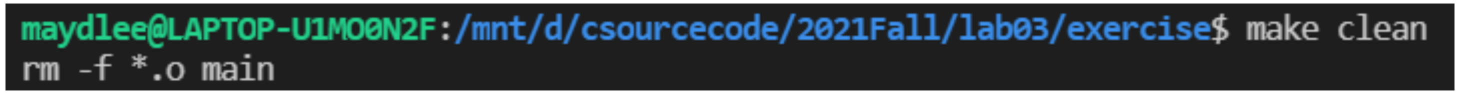


```
fib.cpp
fib.hpp
fib.o
main
main.cpp
main.o
makefile
```

After clean:



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



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