南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

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:

multifiles > functions.h > ...

```h
#pragma once

#define N 5

void printinfo();
int factorial(int n);
```

multifiles > printinfo.cpp > ...

```cpp
#include <iostream>
#include "functions.h"

void printinfo()
{
    std::cout << "Let's go!" << std::endl;
}
```

multifiles > factorial.cpp > factorial(int)

```cpp
#include "functions.h"

int factorial(int n)
{
    if(n == 1)
        return 1;
    else
        return n * factorial(n-1);
}
```

multifiles > main.cpp > ...

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

int main()
{
    printinfo();

    cout << "The factorial of "<< N << " is:" << factorial(N) << endl;

    return 0;
}
```

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

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/makefile/multifiles$ g++ -o testfiles main.cpp printinfo.cpp factorial.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/makefile/multifiles$ ./testfiles
Let's go!
The factorial of 5 is:120
```

or `g++ *.cpp`

How about if there are hundreds of files to compile? If only one source file is modified, need we compile all the files? Makefile will help you.

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**.

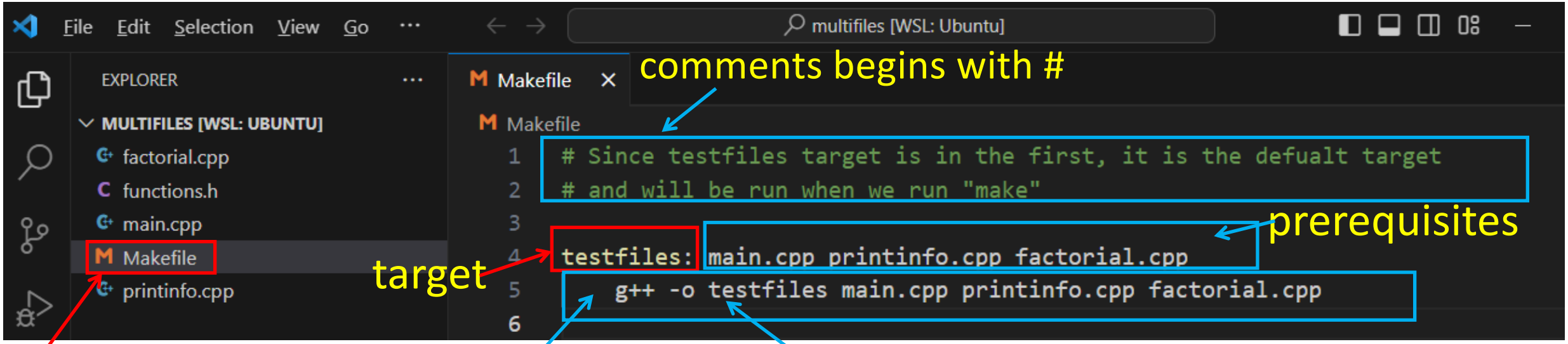
```
targets :  prerequisites

<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.



Place the **Makefile** together with your programs.

start with <TAB>

commands

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

Type the command **make** in VScode

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/makefile/multifiles$ make
```

If you don't install make in VScode, the information will display on the screen.

```
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
```

Install it first according to the instruction.

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/makefile/multifiles$ make
g++ -o testfiles main.cpp printinfo.cpp factorial.cpp
```

Run the commands in the **makefile** automatically.

Run your program

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/makefile/multifiles$ ./testfiles
Let's go!
The factorial of 5 is:120
```

output

Define Macros/Variables in the makefile

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

variables

```
Makefile
1   # Using variables in makefile
2   CXX = g++
3   TARGET = testfiles
4   OBJ = main.o printinfo.o factorial.o
5   $(TARGET) : $(OBJ)
6       $(CXX) -o $(TARGET) $(OBJ)
```

start with <TAB>

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

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/makefile/multifiles$ make
g++     -c -o main.o main.cpp
g++     -c -o printinfo.o printinfo.cpp
g++     -c -o factorial.o factorial.cpp
g++ -o testfiles main.o printinfo.o factorial.o
```

Compile and link the source file one by one

Note: Deletes all the .o files and executable file created previously before using make command. Otherwise, it'll display:

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/makefile/multifiles$ make
make: 'testfiles' is up to date.
```

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

Makefile

```
1   # Using variables in makefile
2   CXX = g++
3   TARGET = testfiles
4   OBJ = main.o printinfo.o factorial.o
5   $(TARGET) : $(OBJ)
6       $(CXX) -o $(TARGET) $(OBJ)
7
8   main.o : main.cpp
9       $(CXX) -c main.cpp
10
11  printinfo.o : printinfo.cpp
12      $(CXX) -c printinfo.cpp
13
14  factorial.o : factorial.cpp
15      $(CXX) -c factorial.cpp
```

targets

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

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/makefile/multifiles$ make
g++ -c main.cpp
g++ -o testfiles main.o printinfo.o factorial.o
```

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

```
# Using several ruls and several targes

CXX = g++
TARGET = testfiles
OBJ = main.o printinfo.o factorial.o

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

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

$(TARGET) : $(OBJ)
    $(CXX) $^ -o $@

%.o : %.cpp
    $(CXX) $(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/makefile/multifiles$ make
g++ -c -Wall main.cpp -o main.o
g++ -c -Wall printinfo.cpp -o printinfo.o
g++ -c -Wall factorial.cpp -o factorial.o
g++ main.o printinfo.o factorial.o -o testfiles
```

```
%.o : %.cpp
    $(CXX) $(CFLAGS) $<
```

or

```
$(CXX) $(CFLAGES) $^
```

Using phony target to clean up compiled results automatically

```
# Using several ruls and several targes

CXX = g++
TARGET = testfiles
OBJ = main.o printinfo.o factorial.o

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

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

%.o : %.cpp
    $(CXX) $(CFLAGES) $^

.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/makefile/multifiles$ make clean
rm -f *.o testfiles
```

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/makefile/multifiles$ make
g++ -c -Wall main.cpp
g++ -c -Wall printinfo.cpp
g++ -c -Wall factorial.cpp
g++ -o testfiles main.o printinfo.o factorial.o
```

After clean, you can run make again

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:

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

```
SRC = $(wildcard ./*.cpp)
```

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

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/makefile/multifiles$ make
./printinfo.cpp ./factorial.cpp ./main.cpp
```

All .cpp files in the current directory

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

for example:

Replace all .cpp files with .o files

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

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

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/makefile/multifiles$ make
./printinfo.cpp ./factorial.cpp ./main.cpp
./printinfo.o ./factorial.o ./main.o
```

Replace all .cpp files with .o files


```
# Using functions

SRC = $(wildcard ./*.cpp)
OBJS = $(patsubst %.cpp, %.o, $(SRC))
TARGET = testfiles

CXX = g++
CFLAGES = -c -Wall

$(TARGET) : $(OBJS)
    $(CXX) -o $@ $(OBJS)
%.o : %.cpp
    $(CXX) $(CFLAGES) $<

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

vs

```
OBJ = main.o printinfo.o factorial.o
```

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/makefile/multifiles$ make
g++ -c -Wall printinfo.cpp
g++ -c -Wall factorial.cpp
g++ -c -Wall main.cpp
g++ -o testfiles  ./printinfo.o  ./factorial.o  ./main.o
```

Use Options to Control Optimization

-O1, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.

-O2,Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. As compared to -O1, this option increases both compilation time and the performance of the generated code.

-O3, Optimize yet more. O3 turns on all optimizations specified by -O2.

https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

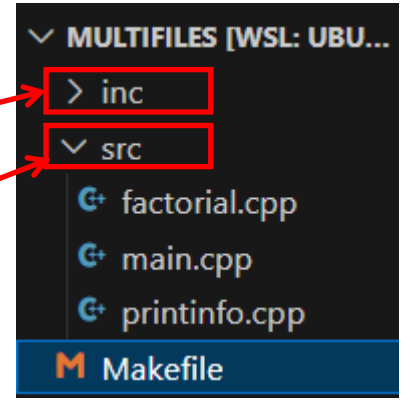
https://blog.csdn.net/xinianbuxiu/article/details/51844994

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

# Options pass to complier
# -c: generates the object file
# -Wall: displays complier warnings
# -O0: no optimization
# -O1: dafault optimization
# -O2: represents the second level optimization
# -O3: represents the highest leverl optimization

CXX      = g++
CFLAGES = -c -Wall
CXXFLAGES = $(CFLAGES) -O3

$(TARGET) : $(OBJS)
    $(CXX) -o $@ $(OBJS)
%.o : %.cpp
    $(CXX) $(CXXFLAGES) $< -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/makefile/multifiles$ make
g++ -c -Wall -O3  src/printinfo.cpp -o src/printinfo.o -I./inc
g++ -c -Wall -O3  src/factorial.cpp -o src/factorial.o -I./inc
g++ -c -Wall -O3  src/main.cpp -o src/main.o -I./inc
g++ -o testfiles  ./src/printinfo.o  ./src/factorial.o  ./src/main.o
maydlee@LAPTOP-U1MO0N2F:/mnt/d/makefile/multifiles$ ls
Makefile  inc  src  testfiles
```

GNU Make Manual

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

Keyboard input and terminal output of character array

1. C: scanf & printf

%d ----int

%f ----float

%c -----char

%s -----string

```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;
}
```

There is no &

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/IO$ gcc scanf_printf.c
maydlee@LAPTOP-U1MO0N2F:/mnt/d/IO$ ./a.out
Enter a string:
Computer
You entered: Computer
maydlee@LAPTOP-U1MO0N2F:/mnt/d/IO$ ./a.out
Enter a string:
Computer Science
You entered: Computer
```

Why only Computer?

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

2. C: gets & puts

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

```c
#include <stdio.h>

int main()
{
    char str[20];
    printf("Enter a string:\n");
    gets(str);

    printf("You entered: ");
    puts(str);

    return 0;
}
```

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

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

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/IO$ 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’? [-Wimplicit-function-declaration]
    7 |  gets(str);
      |  ^~~~
      |  fgets
/usr/bin/ld: /tmp/cc3CJcP4.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/IO$ ./a.out
Enter a string:
Computer Science
You entered: Computer Science
```

3. C++: cin & cout

cin_cout.cpp > ...

```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;
}
```

```
Enter a string:C++
You entered: C++
Enter an other string:programming is funny.
You entered: programming
```

```
Enter a string:C++ programming is funny.
You entered: C++
Enter an other string:You entered: programming
```

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

4. C++: cin.get( )

Input a single character:
istream& get(char&);
int get(void);

Input a string:
istream& get(char*,int);

cin_get.cpp > ...

```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.get(str, 20);
    cout << "You entered: " << str << endl;

    return 0;
}
```

If the statement is omitted, what will be the output?

```
Enter a string:C and C++
You entered: C and C++
Enter an other string:C/C++ programming is funny.
You entered: C/C++ programming i
```

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

```
Enter a string:C and C++
You entered: C and C++
Enter an other string:You entered:
```

5. C++: cin.getline( )

Input a string:

istream& getline(char*,int);

cin_getline.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.getline(str, 20);
    cout << "You entered: " << str << endl;

    return 0;
}
```

```
Enter a string:C and C++
You entered: C and C++
Enter an other string:C/C++ programming is funny.
You entered: C/C++ programming i
```

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

cin.get( ) vs cin.getline( )

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.

```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;
}
```

Program runs without entering another string

```
Enter a string:C and C++
You entered: C and C++
Enter an other string:You entered:
```

6. string class I/O

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

string_input.cpp > ...

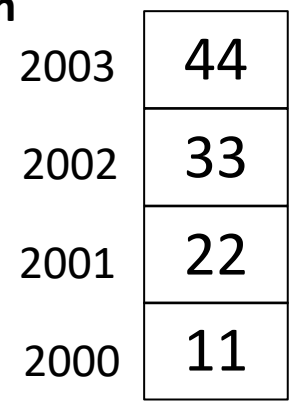
```cpp
#include <iostream>
using namespace std;

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

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

    return 0;
}
```

```
Enter a string:C and C++
You entered: C and C++
Enter another string:C/C++ programming is funny.
You entered: C/C++ programming is funny.
```

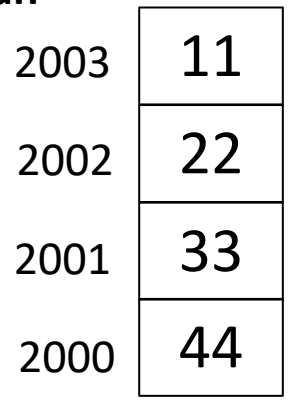
Big-Endian and Little-Endian

BE stores the big-end first, the lowest memory address is the biggest.
LE stores the little-end first, the lowest memory address is the littlest.

Big-Endian



Little-Endian




```
#include<stdio.h>
union data
{
   int a;
   char c;
};

int main()
{
   union data endian;
   endian.a = 0x11223344;

   if(endian.c == 0x11)
      printf("Big-Endian\n");
   else if(endian.c == 0x44)
      printf("Little-Endian\n");

   return 0;
}
```

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

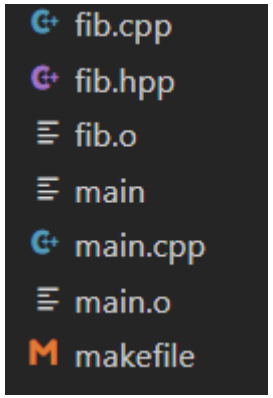
- Design a struct “DayInfo” which contains two enumeration types as its member. The first is an enum “Day” for (Sunday, Monday, ...), and the second is an enum “Weather” for (Sunny, Rainy, ...).
- Define a boolean function “bool canTravel(DayInfo)” . It will return true if the day is at weekend and the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

```
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:



After clean: