C/C++ Program Design

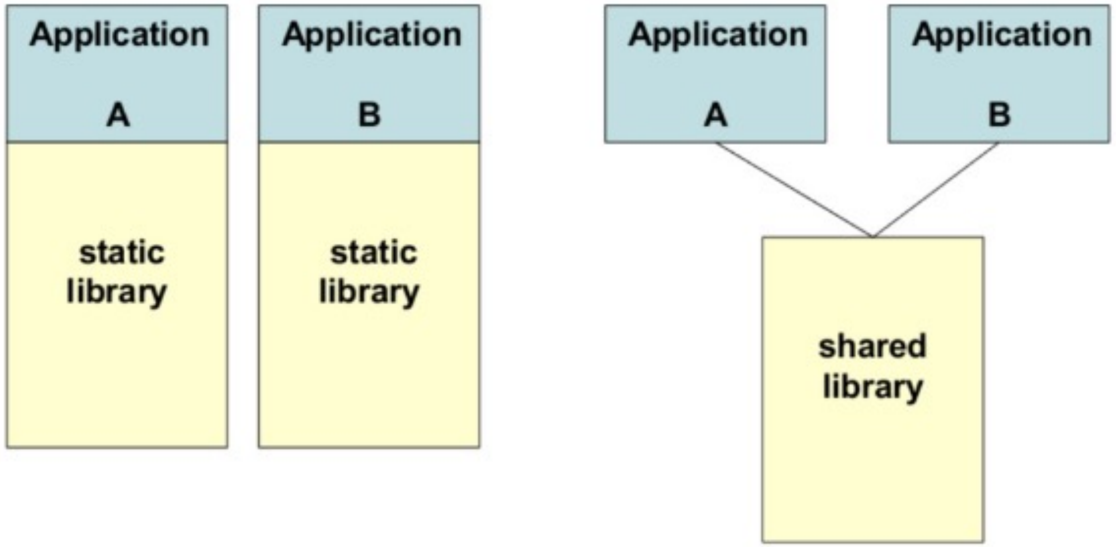
Lab 6, static library

廖琪梅 王大兴 于仕琪

Static library and Dynamic library

Static Linking and Static Libraries (also known as an **archive**) is the result of the linker making copy of all used library functions to the executable file. Static Linking creates larger binary files, and need more space on disk and main memory. Examples of static libraries are, ***.a*** files in Linux and ***.lib*** files in Windows.

Dynamic linking and Dynamic Libraries Dynamic Linking doesn't require the code to be copied, it is done by just placing name of the library in the binary file. The actual linking happens when the program is run, when both the binary file and the library are in memory. If multiple programs in the system link to the same dynamic link library, they all reference the library. Therefore, this library is shared by multiple programs and is called a "**shared library**" . Examples of Dynamic libraries are, ***.so*** in Linux and ***.dll*** in Windows.



	advantages	disadvantages
Static Library	1. Make the executable has fewer dependencies, has been packaged into the executable file. 2. The link is completed in the compilation stage, and the code is loaded quickly during execution.	1. Make the executable file larger. 2. Being a library dependent on another library will result in redundant copies because it must be packaged with the target file. 3. Upgrade is not convenient and easy. The entire executable needs to be replaced and recompiled.
Dynamic Library	1.Dynamic library can achieve resource sharing between processes, there can be only one library file. 2. The upgrade procedure is simple, do not need to recompile.	1. Loading during runtime will slow down the execution speed of code. 2. Add program dependencies that must be accompanied by an executable file.

Building a static library

- Suppose we have written the following code:

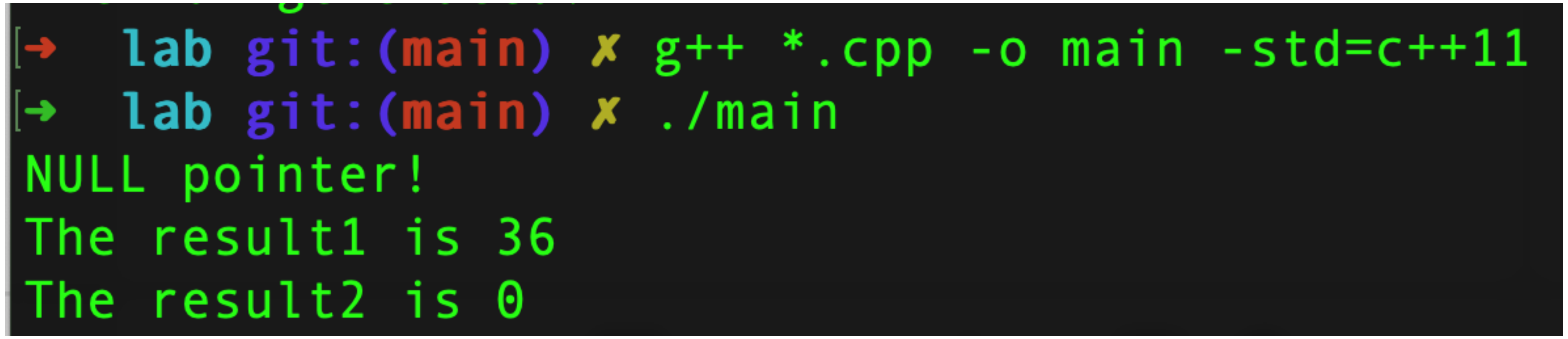
```
// mymath.h
#ifndef __MY_MATH_H__
#define __MY_MATH_H__
float arraySum(const float *array, size_t size);
#endif
```

```
// mymath.cpp
#include <iostream>
#include "mymath.h"

float arraySum(const float *array, size_t size)
{
    if(array == NULL)
    {
        std::cerr << "NULL pointer!" << std::endl;
        return 0.0f;
    }
    float sum = 0.0f;
    for(size_t i = 0; i < size; i++)
        sum += array[i];
    return sum;
}
```

```
// main.cpp
#include <iostream>
#include "mymath.h"
int main()
{
    float arr1[8]{1.f, 2.f, 3.f, 4.f, 5.f, 6.f, 7.f, 8.f};
    float * arr2 = NULL;

    float sum1 = arraySum(arr1, 8);
    float sum2 = arraySum(arr2, 8);

    std::cout << "The result1 is " << sum1 <<
std::endl;
    std::cout << "The result2 is " << sum2 <<
std::endl;

    return 0;
}
```

Building a static library

- In previous class we do the following:
- This will compile the “main.cpp” and “mymath.cpp” into “main”
- And then run “main”



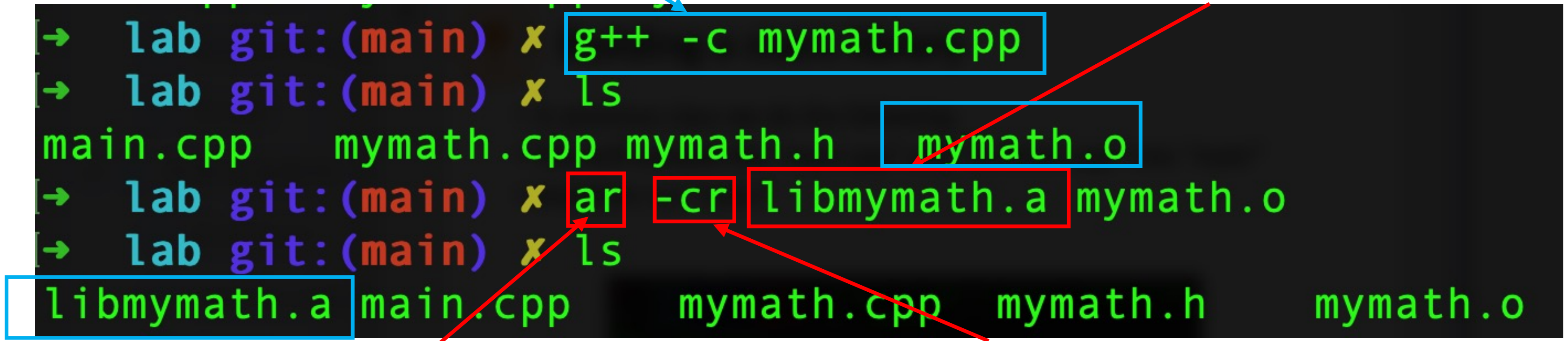
```
➜  lab git:(main) ✗ g++ *.cpp -o main -std=c++11
➜  lab git:(main) ✗ ./main
NULL pointer!
The result1 is 36
The result2 is 0
```

Building a static library

- A static library is created by **.o** file.
- Remember to use **“ar”** command with arguments “**-cr**” when building it.
- Now we should see “libmymath.a” in the current directory

Compile the source file to the object file.

The name of **.a** must be started with “**lib**” followed by the .cpp name in which a function is defined.



```
→  lab git:(main) ✗ g++ -c mymath.cpp
→  lab git:(main) ✗ ls
main.cpp    mymath.cpp mymath.h   mymath.o
→  lab git:(main) ✗ ar -cr libmymath.a mymath.o
→  lab git:(main) ✗ ls
libmymath.a main.cpp     mymath.cpp  mymath.h     mymath.o
```

ar is a linux command.

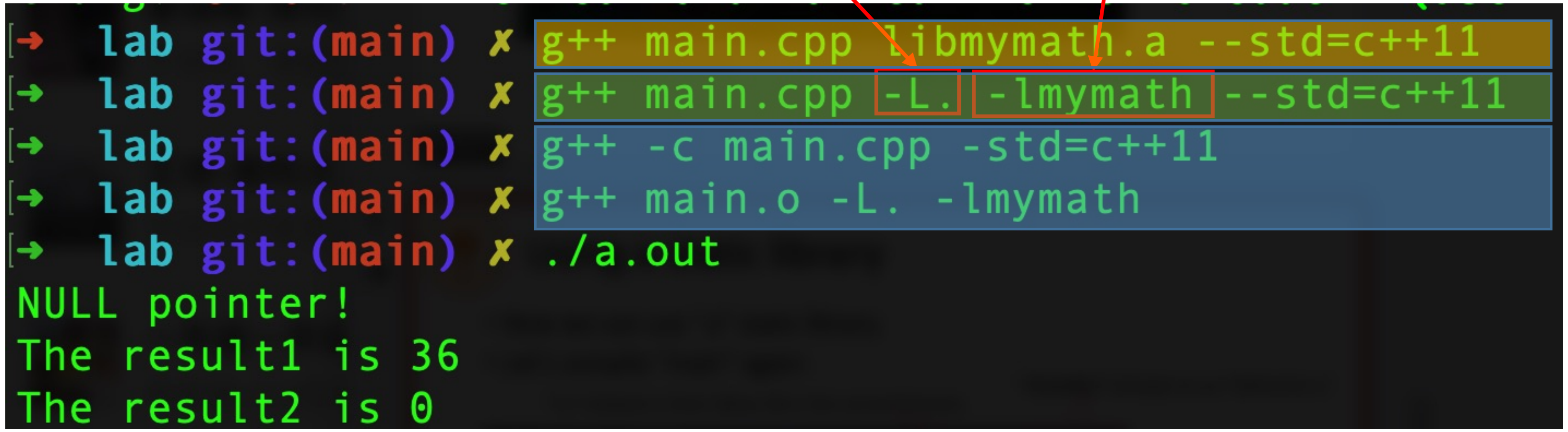
c: create a static library.
r: add the object file to the static library.

Using a static library

- Now we can use “.a” static library.
- Let’s compile “main” again:

“-lmymath” indicates to use “libmymath.a” or “libmymath.so”

“-L.” indicates to find a library file in the current directory.



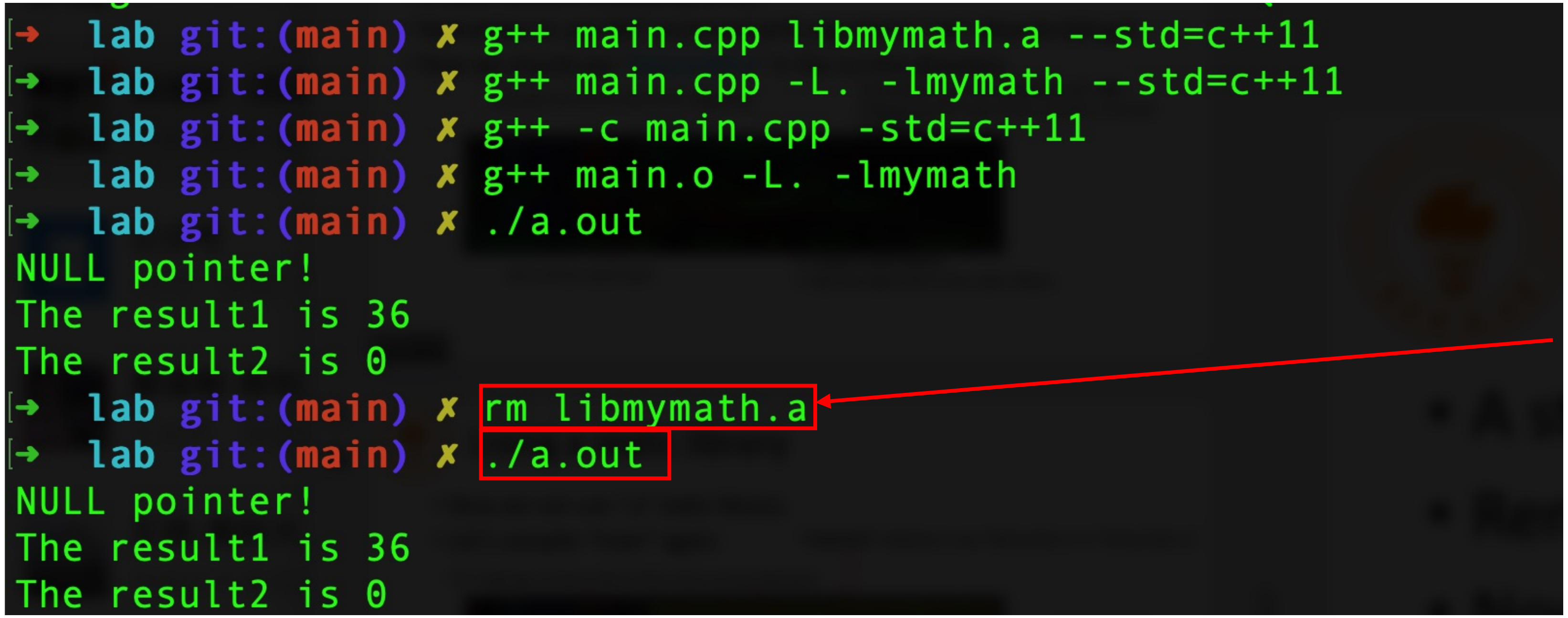
```
➜  lab git:(main) ✗ g++ main.cpp libmymath.a --std=c++11
➜  lab git:(main) ✗ g++ main.cpp -L. -lmymath --std=c++11
➜  lab git:(main) ✗ g++ -c main.cpp -std=c++11
➜  lab git:(main) ✗ g++ main.o -L. -lmymath
➜  lab git:(main) ✗ ./a.out
NULL pointer!
The result1 is 36
The result2 is 0
```

The 3 methods
are equivalent.

- **-L**: indicates the directory of libraries
- **-l**: indicates the library name, the compiler can give the “**lib**” prefix to the library name and follows with **.a** as extension name.

Using a static library

If the static library is removed, the program can run normally.



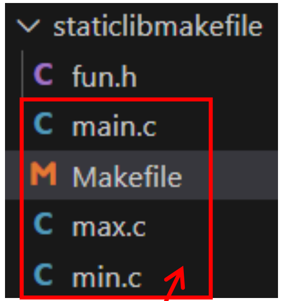
```
➜  lab git:(main) ✗ g++ main.cpp libmymath.a --std=c++11
➜  lab git:(main) ✗ g++ main.cpp -L. -lmymath --std=c++11
➜  lab git:(main) ✗ g++ -c main.cpp -std=c++11
➜  lab git:(main) ✗ g++ main.o -L. -lmymath
➜  lab git:(main) ✗ ./a.out
NULL pointer!
The result1 is 36
The result2 is 0
➜  lab git:(main) ✗ rm libmymath.a
➜  lab git:(main) ✗ ./a.out
NULL pointer!
The result1 is 36
The result2 is 0
```

remove the static library file.

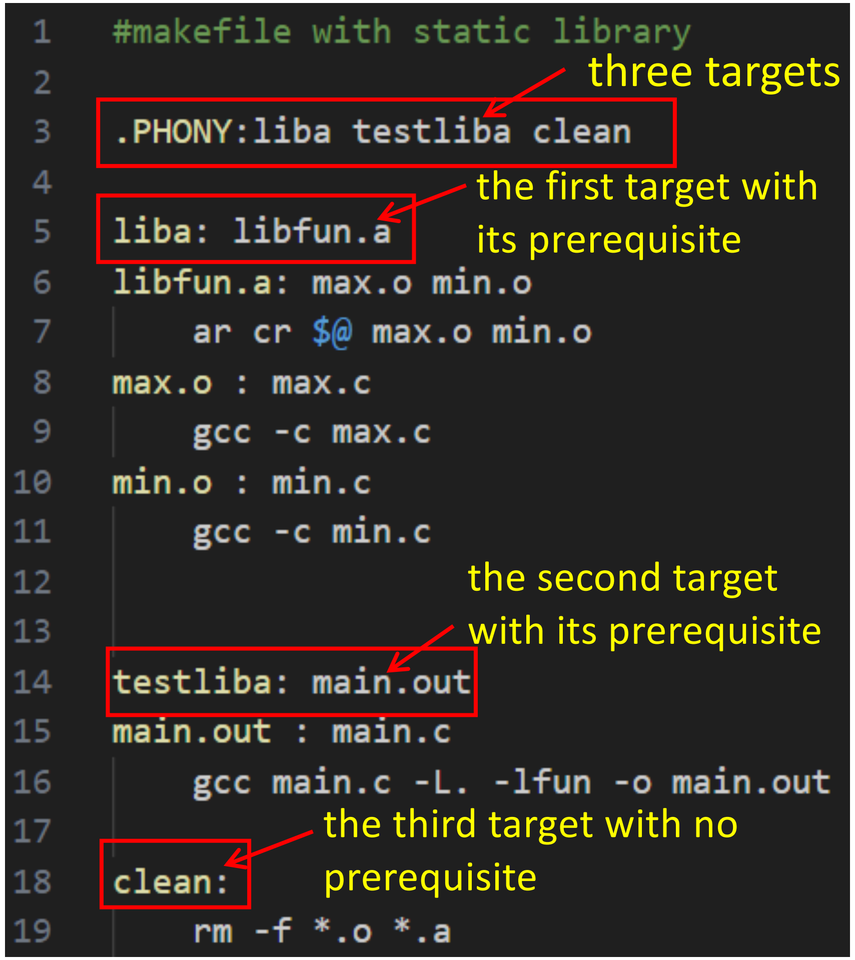
To create a static library from multiple object files:

ar –cr libtest.a test1.o test2.o

Static library in makefile



All the files are in the same folder.



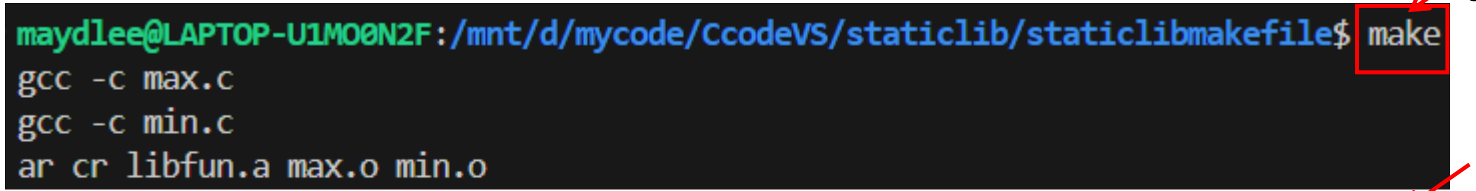
```
#makefile with static library

.PHONY:liba testliba clean

liba: libfun.a
libfun.a: max.o min.o
	ar cr $@ max.o min.o
max.o : max.c
	gcc -c max.c
min.o : min.c
	gcc -c min.c


testliba: main.out
main.out : main.c
	gcc main.c -L. -lfun -o main.out

clean:
	rm -f *.o *.a
```

three targets

the first target with its prerequisite

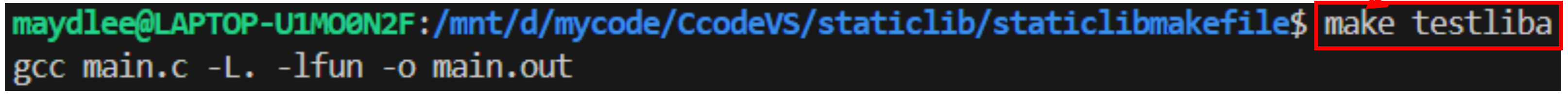
the second target with its prerequisite

the third target with no prerequisite



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefile$ make
gcc -c max.c
gcc -c min.c
ar cr libfun.a max.o min.o
```

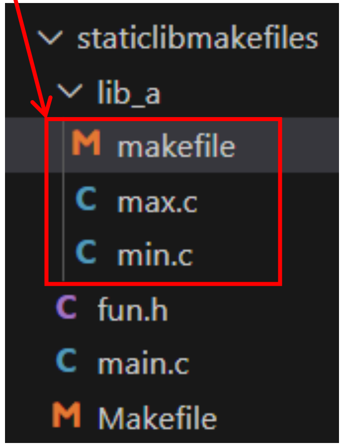
By default, the first target can run with only make command.



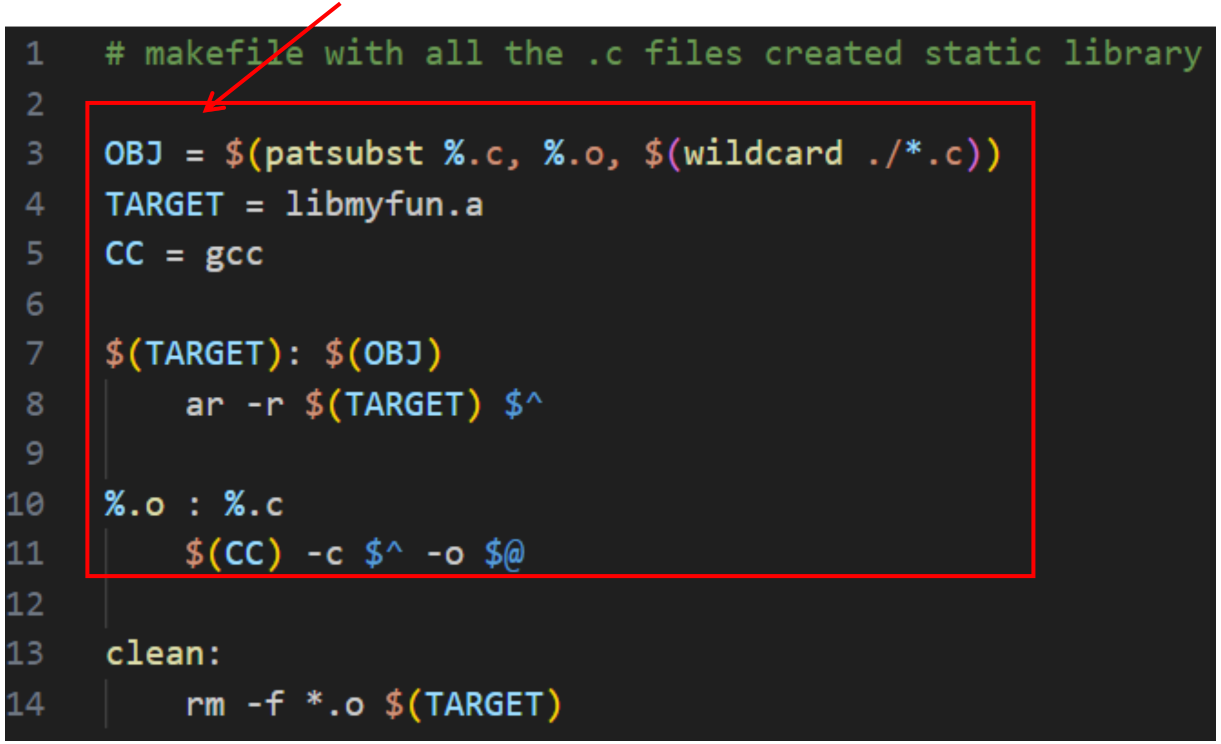
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefile$ make testliba
gcc main.c -L. -lfun -o main.out
```

The target name followed make command can run the target.

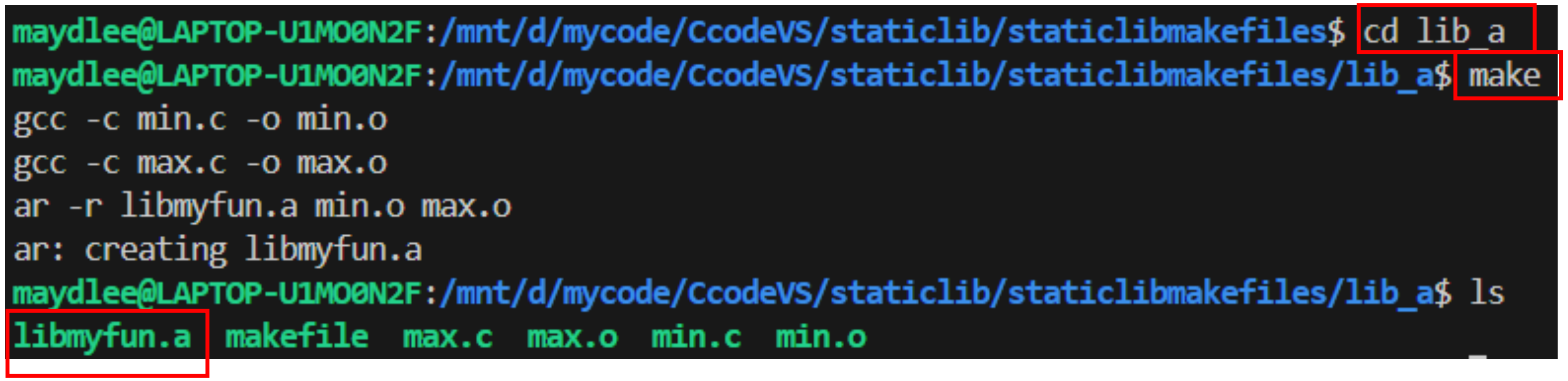
This time we put the functions in the “lib_a” folder, and create a makefile in this folder.



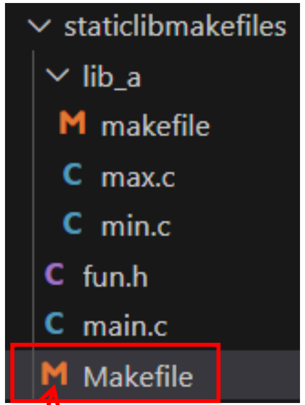
The first step, creates a static library file with these two .o files in the current makefile.



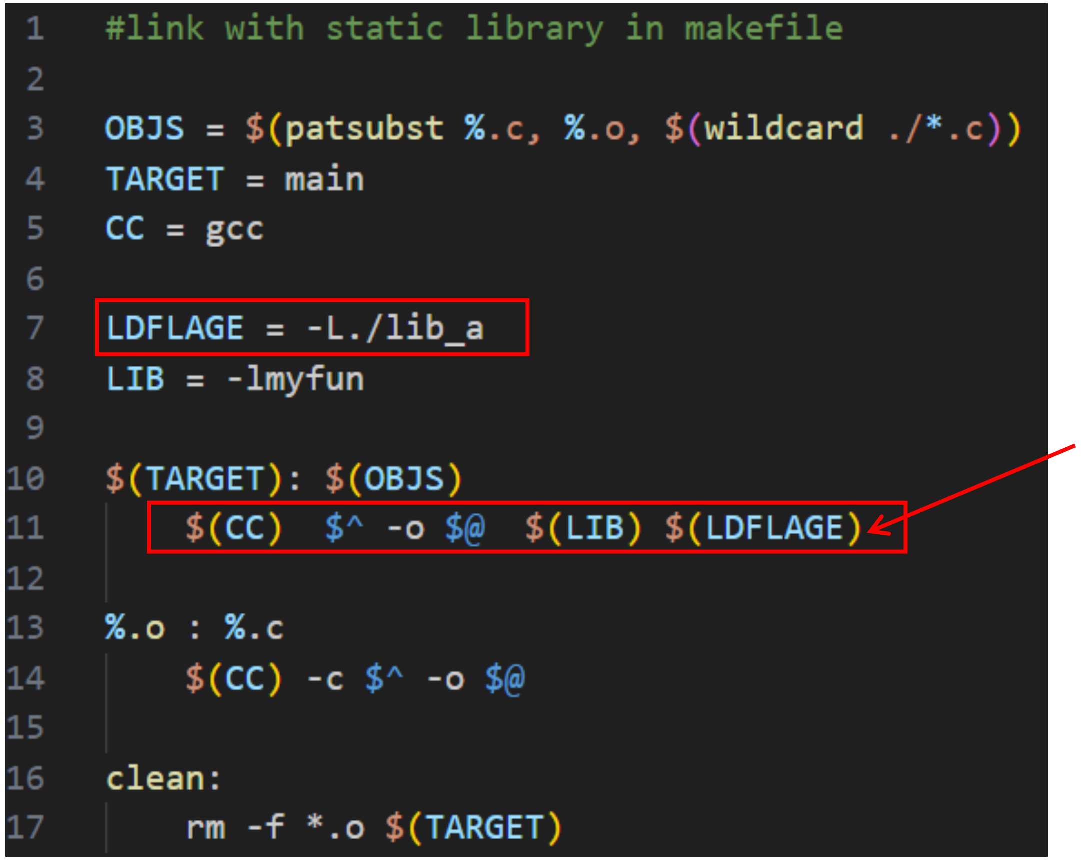
```
# makefile with all the .c files created static library

OBJ = $(patsubst %.c, %.o, $(wildcard ./*.c))
TARGET = libmyfun.a
CC = gcc

$(TARGET): $(OBJ)
	ar -r $(TARGET) $^

%.o : %.c
	$(CC) -c $^ -o $@

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



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefiles$ cd lib_a
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefiles/lib_a$ make
gcc -c min.c -o min.o
gcc -c max.c -o max.o
ar -r libmyfun.a min.o max.o
ar: creating libmyfun.a
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefiles/lib_a$ ls
libmyfun.a  makefile  max.c  max.o  min.c  min.o
```

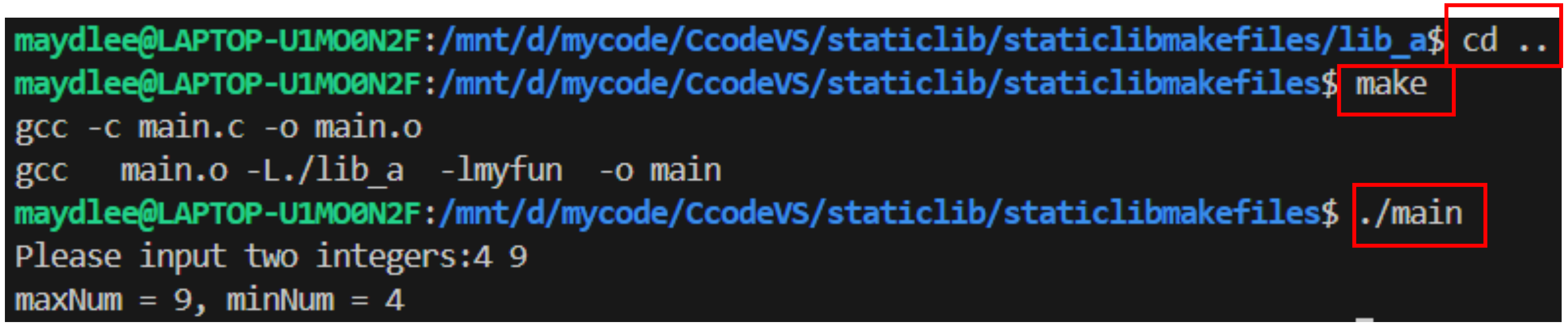


The second step, creates another makefile in the upper-level folder to link the static library into the executable file.

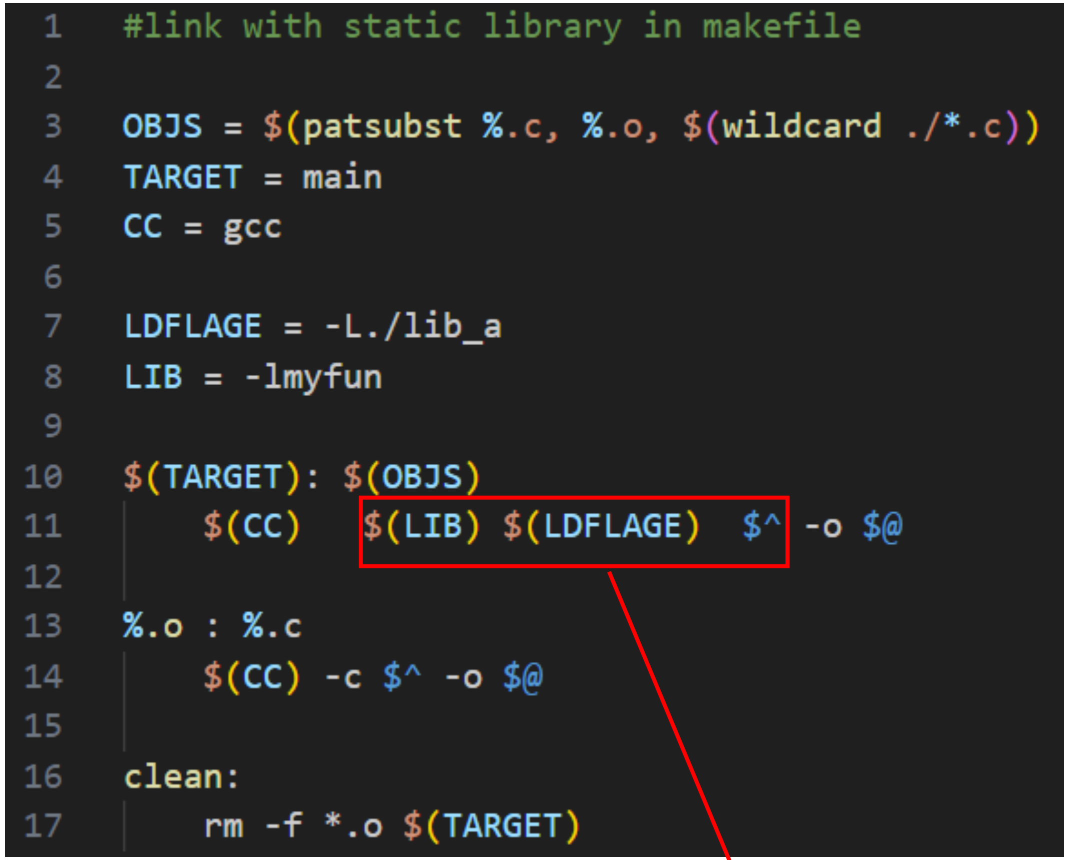


```
#link with static library in makefile

OBJS = $(patsubst %.c, %.o, $(wildcard ./*.c))
TARGET = main
CC = gcc

LDFLAGE = -L./lib_a
LIB = -lmyfun

$(TARGET): $(OBJS)
	$(CC)  $^ -o $@  $(LIB) $(LDFLAGE)

%.o : %.c
	$(CC) -c $^ -o $@

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

Links the executable file with the static library.

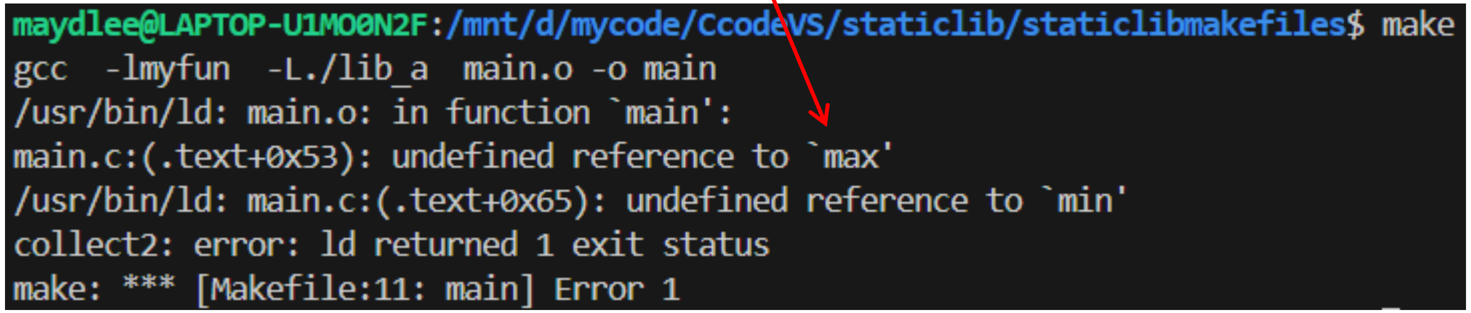


```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefiles/lib_a$ cd ..
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefiles$ make
gcc -c main.c -o main.o
gcc    main.o -L./lib_a  -lmyfun  -o main
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefiles$ ./main
Please input two integers:4 9
maxNum = 9, minNum = 4
```

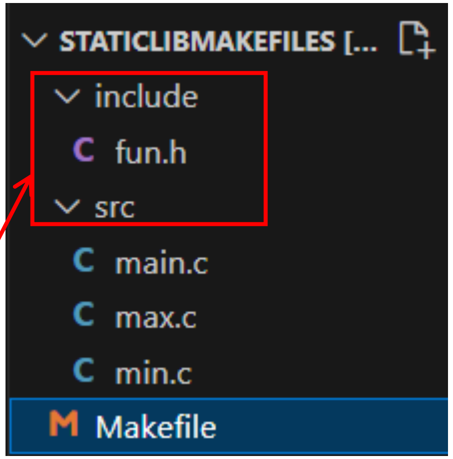


```
1   #link with static library in makefile
2
3   OBJS = $(patsubst %.c, %.o, $(wildcard ./*.c))
4   TARGET = main
5   CC = gcc
6
7   LDFLAGE = -L./lib_a
8   LIB = -lmyfun
9
10  $(TARGET): $(OBJS)
11      $(CC)   $(LIB) $(LDFLAGE)  $^ -o $@
12
13  %.o : %.c
14      $(CC) -c $^ -o $@
15
16  clean:
17      rm -f *.o $(TARGET)
```

If you put the flag before $^, it will cause error.

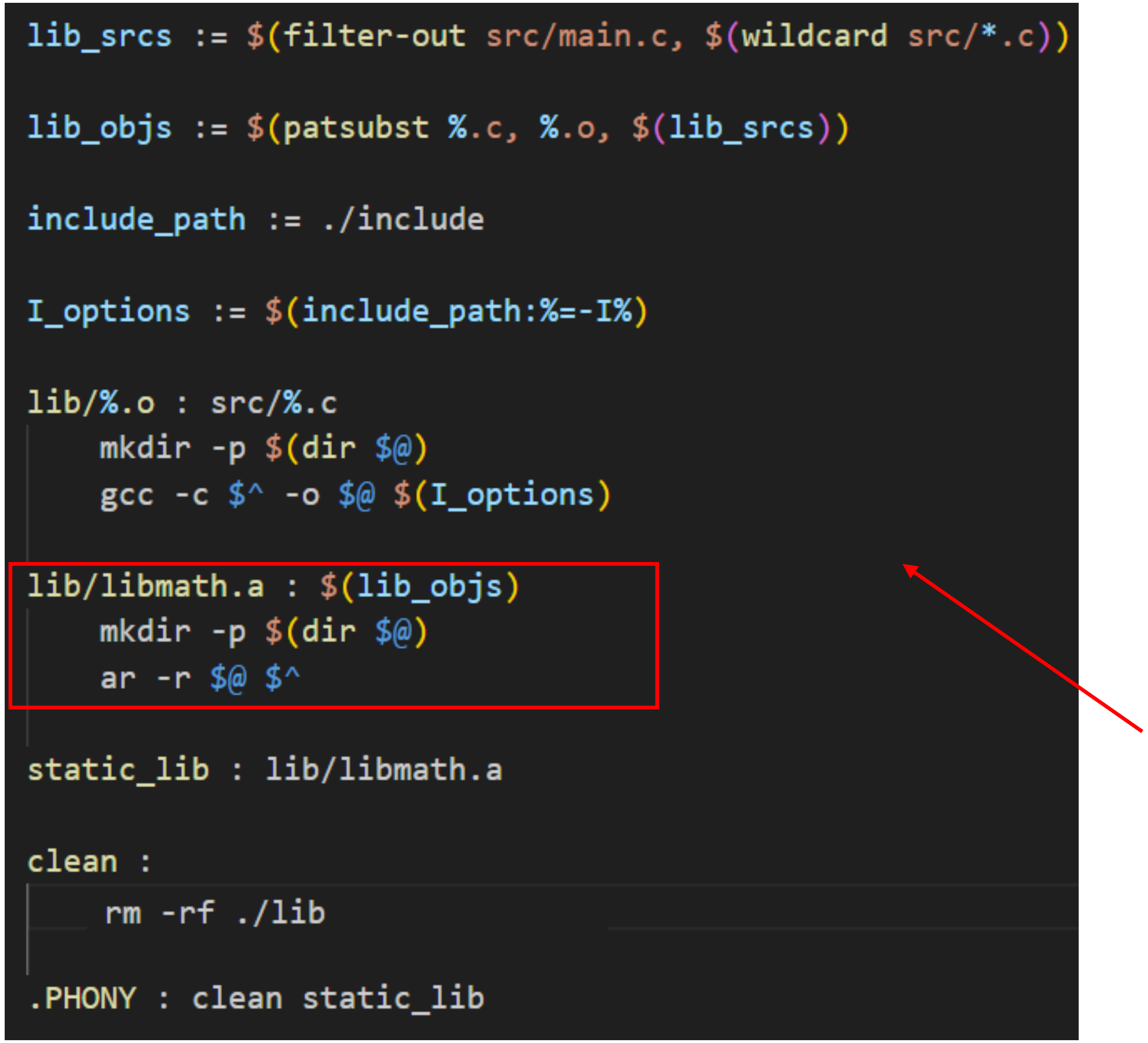


```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefiles$ make
gcc  -lmyfun  -L./lib_a  main.o -o main
/usr/bin/ld: main.o: in function `main':
main.c:(.text+0x53): undefined reference to `max'
/usr/bin/ld: main.c:(.text+0x65): undefined reference to `min'
collect2: error: ld returned 1 exit status
make: *** [Makefile:11: main] Error 1
```



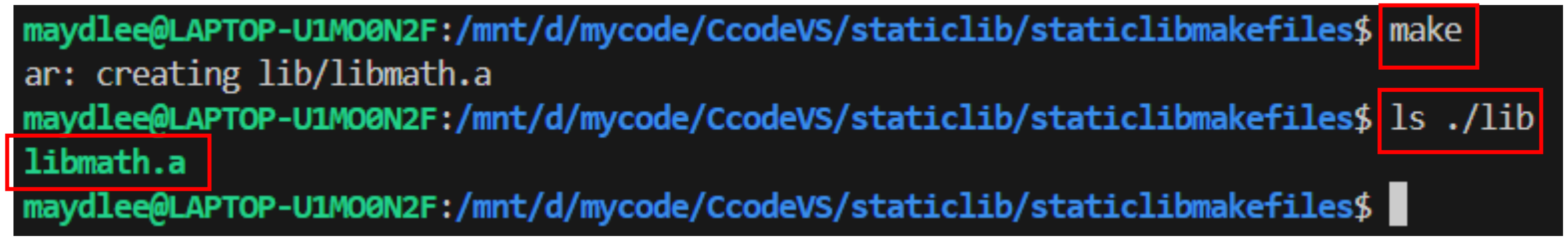
```
STATICLIBMAKEFILES [...
  include
    fun.h
  src
    main.c
    max.c
    min.c
  Makefile
```

This time we put all the source files in the “src” folder, the function header file in the “include” folder, and create a makefile in the current folder.(Only one makefile)

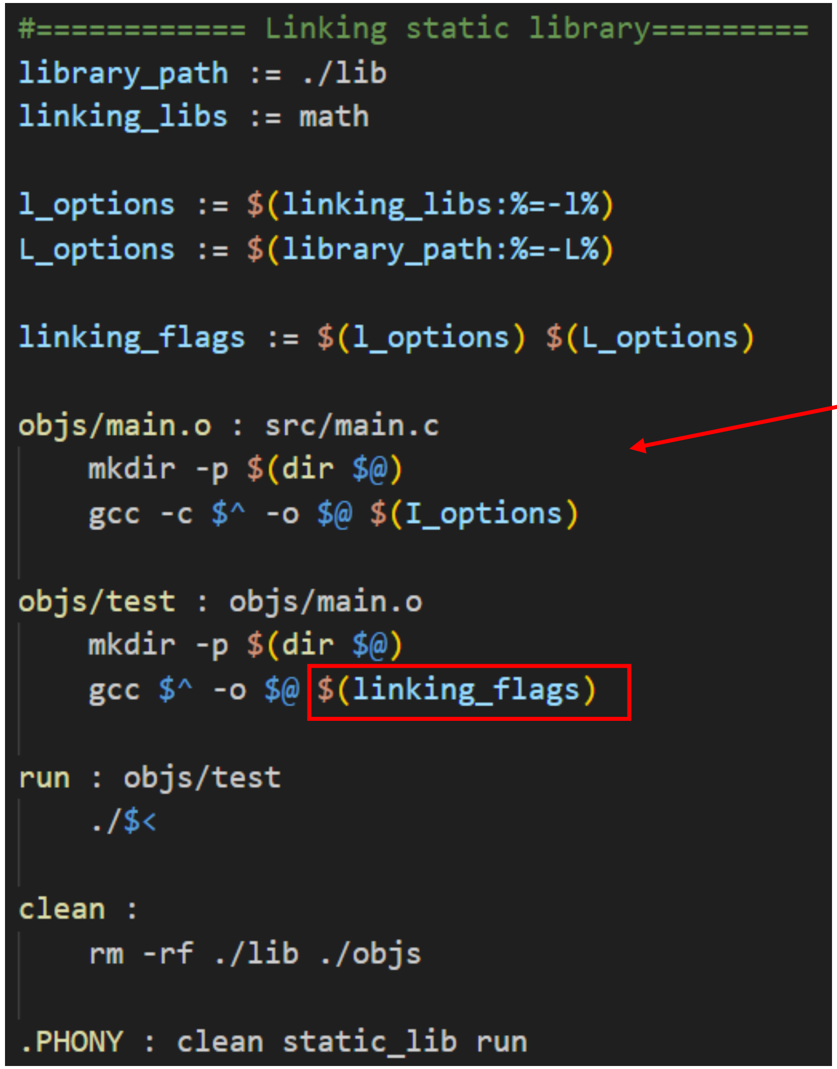


```
lib_srcs := $(filter-out src/main.c, $(wildcard src/*.c))

lib_objs := $(patsubst %.c, %.o, $(lib_srcs))

include_path := ./include

I_options := $(include_path:%=-I%)

lib/%.o : src/%.c
	mkdir -p $(dir $@)
	gcc -c $^ -o $@ $(I_options)

lib/libmath.a : $(lib_objs)
	mkdir -p $(dir $@)
	ar -r $@ $^

static_lib : lib/libmath.a

clean :
	rm -rf ./lib

.PHONY : clean static_lib
```

The first part of the makefile just creates a static library named **libmath.a**

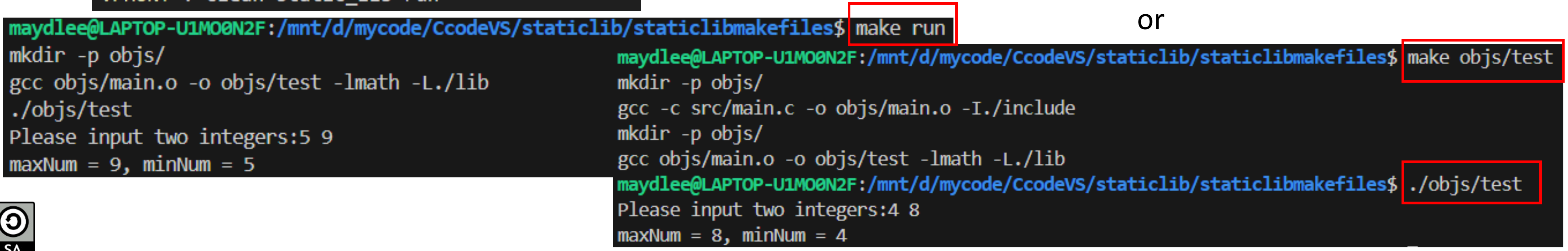


```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefiles$ make
ar: creating lib/libmath.a
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefiles$ ls ./lib
libmath.a
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefiles$ 
```



```
#============ Linking static library=========
library_path := ./lib
linking_libs := math

l_options := $(linking_libs:%=-l%)
L_options := $(library_path:%=-L%)

linking_flags := $(l_options) $(L_options)

objs/main.o : src/main.c
    mkdir -p $(dir $@)
    gcc -c $^ -o $@ $(I_options)

objs/test : objs/main.o
    mkdir -p $(dir $@)
    gcc $^ -o $@ $(linking_flags)

run : objs/test
    ./$<

clean :
    rm -rf ./lib ./objs

.PHONY : clean static_lib run
```

The second part of the makefile links the static library **libmath.a** to the executable file **test** in the “objs” folder.



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefiles$ make run
mkdir -p objs/
gcc objs/main.o -o objs/test -lmath -L./lib
./objs/test
Please input two integers:5 9
maxNum = 9, minNum = 5
```

or

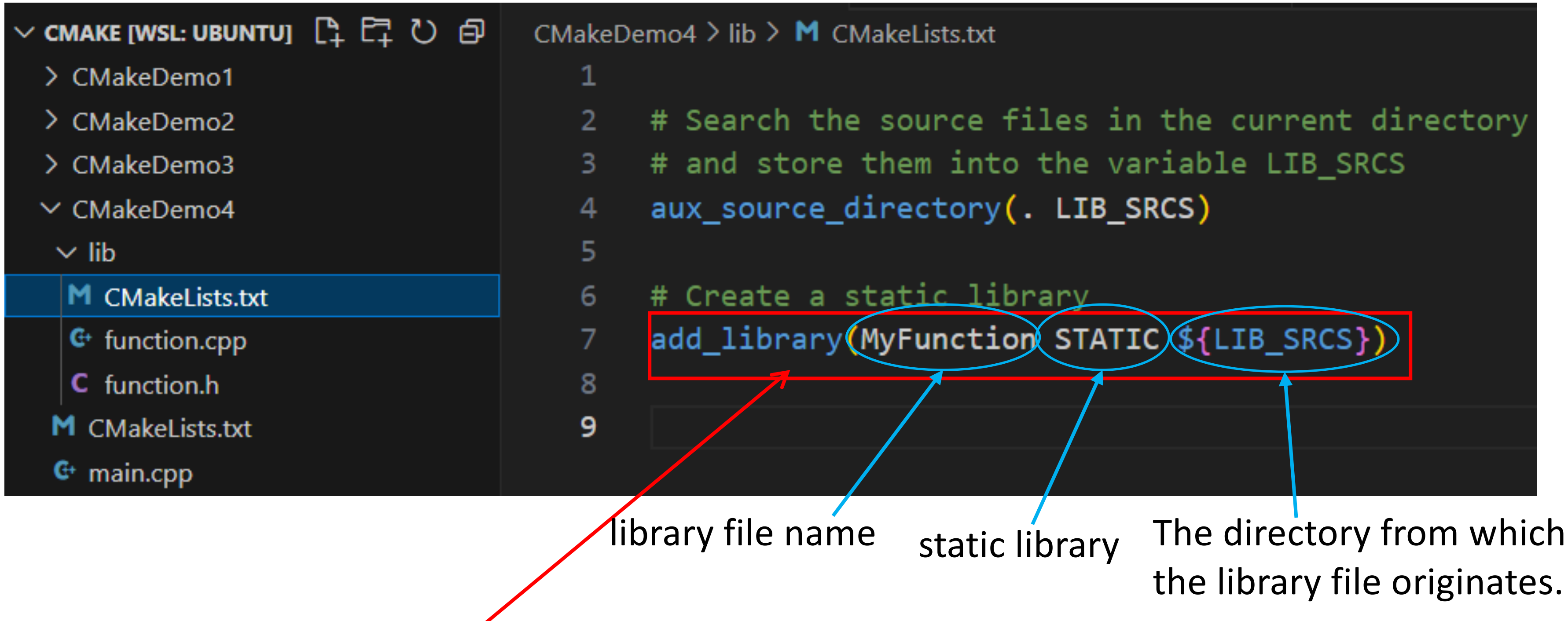
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefiles$ make objs/test
mkdir -p objs/
gcc -c src/main.c -o objs/main.o -I./include
mkdir -p objs/
gcc objs/main.o -o objs/test -lmath -L./lib
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/staticlib/staticlibmakefiles$ ./objs/test
Please input two integers:4 8
maxNum = 8, minNum = 4
```

Creating and linking a static library by CMake

We want to create a static(or dynamic) library by function.cpp and call the static library in main.cpp. This time we write two CMakeLists.txt files, one in **CmakeDemo4** folder and another in **lib** folder.

The CMakeLists.txt in **lib** folder creates a static library.

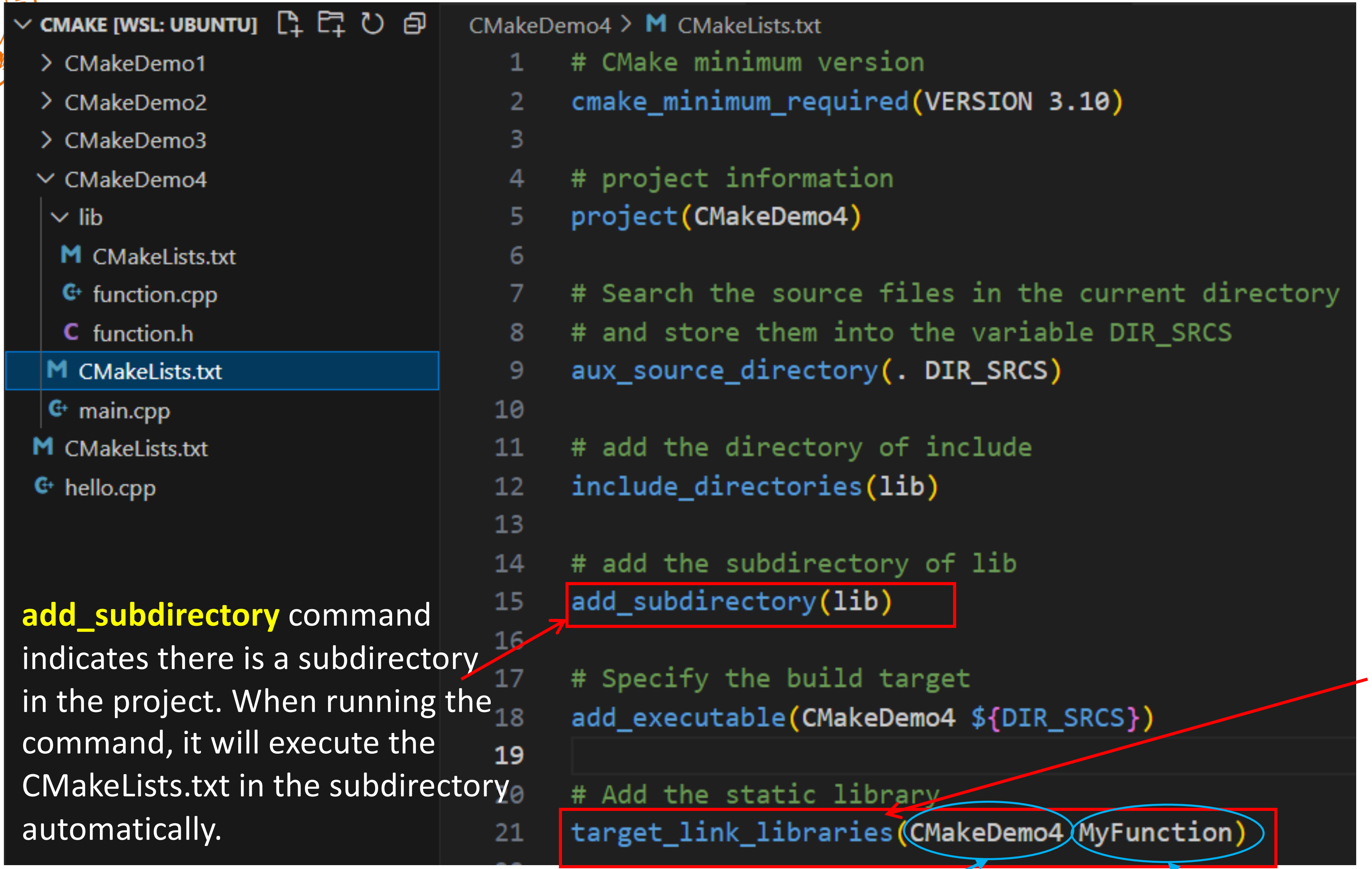
```
./CMakeDemo4
        |
        +--- main.cpp
        |
        +--- lib/
              |
              +--- function.h
              |
              +--- function.cpp
```



```
# Search the source files in the current directory
# and store them into the variable LIB_SRCS
aux_source_directory(. LIB_SRCS)

# Create a static library
add_library(MyFunction STATIC ${LIB_SRCS})
```

Create a static library named **libMyFunction.a** by the files in the current directory.

The CMakeLists.txt in **CMakeDemo4** folder creates the project.



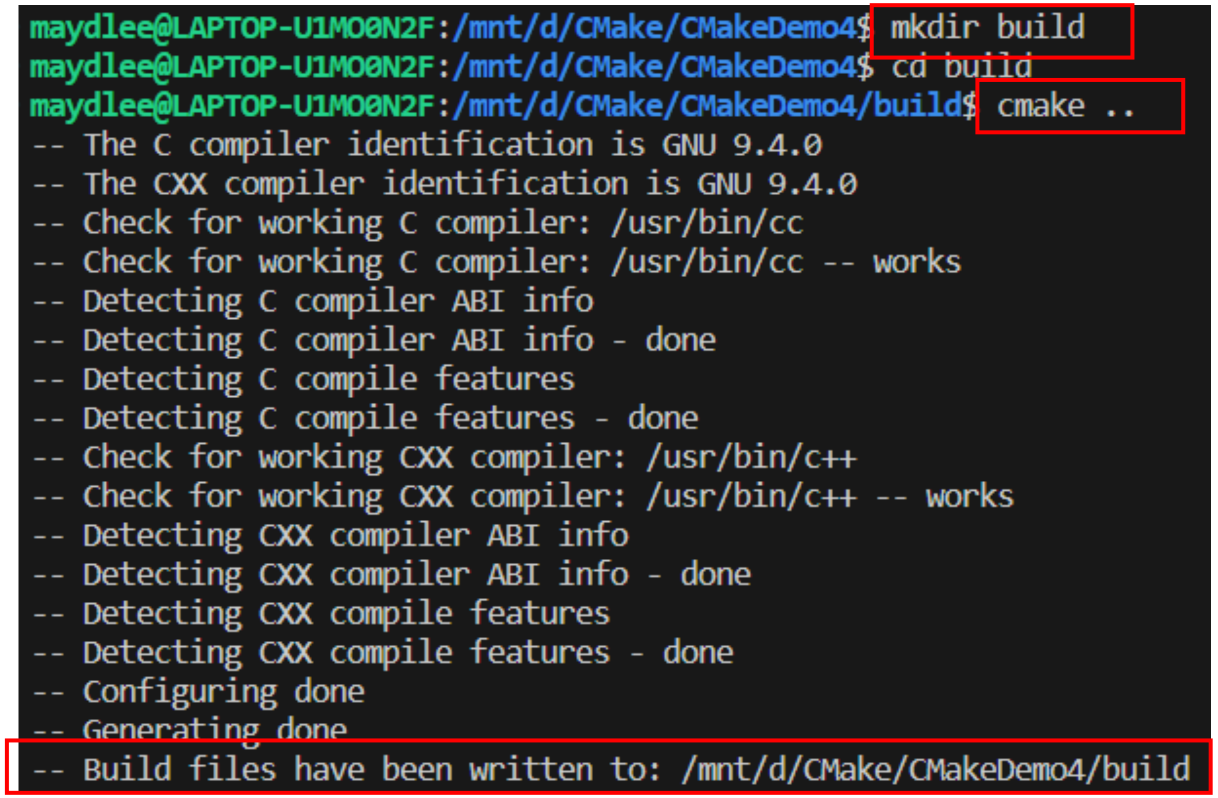
```
# CMake minimum version
cmake_minimum_required(VERSION 3.10)

# project information
project(CMakeDemo4)

# Search the source files in the current directory
# and store them into the variable DIR_SRCS
aux_source_directory(. DIR_SRCS)

# add the directory of include
include_directories(lib)

# add the subdirectory of lib
add_subdirectory(lib)

# Specify the build target
add_executable(CMakeDemo4 ${DIR_SRCS})

# Add the static library
target_link_libraries(CMakeDemo4 MyFunction)
```

add_subdirectory command indicates there is a subdirectory in the project. When running the command, it will execute the CMakeLists.txt in the subdirectory automatically.

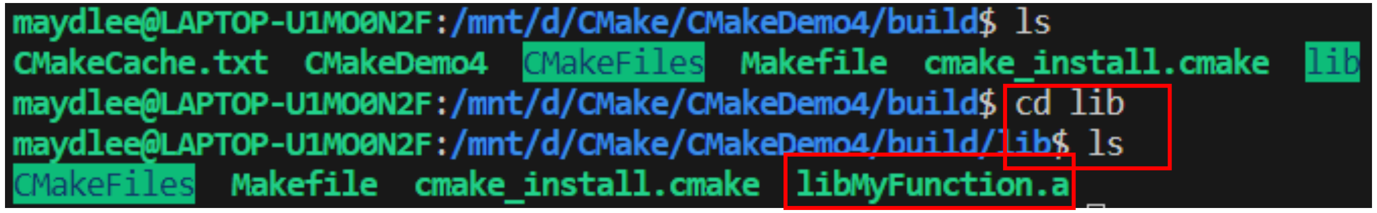
Indicates that the project needs link a library named **MyFunction**, MyFunction can be a static library file or a dynamic library file.

project name

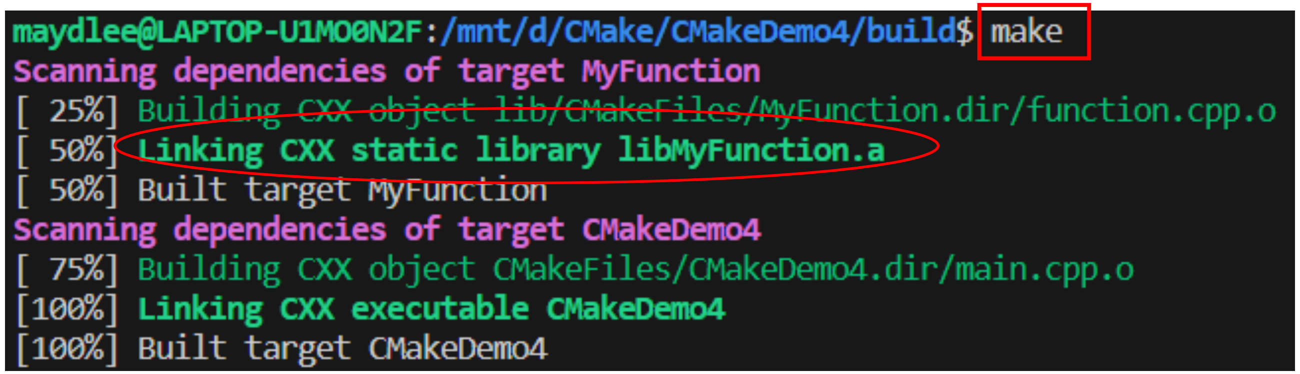
library file name
If there are more than one file, list them using space as the separator.



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/CMake/CMakeDemo4$ mkdir build
maydlee@LAPTOP-U1MO0N2F:/mnt/d/CMake/CMakeDemo4$ cd build
maydlee@LAPTOP-U1MO0N2F:/mnt/d/CMake/CMakeDemo4/build$ cmake ..
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /mnt/d/CMake/CMakeDemo4/build
```



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/CMake/CMakeDemo4/build$ ls
CMakeCache.txt  CMakeDemo4  CMakeFiles  Makefile  cmake_install.cmake  lib
maydlee@LAPTOP-U1MO0N2F:/mnt/d/CMake/CMakeDemo4/build$ cd lib
maydlee@LAPTOP-U1MO0N2F:/mnt/d/CMake/CMakeDemo4/build/lib$ ls
CMakeFiles  Makefile  cmake_install.cmake  libMyFunction.a
```



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/CMake/CMakeDemo4/build$ make
Scanning dependencies of target MyFunction
[ 25%] Building CXX object lib/CMakeFiles/MyFunction.dir/function.cpp.o
[ 50%] Linking CXX static library libMyFunction.a
[ 50%] Built target MyFunction
Scanning dependencies of target CMakeDemo4
[ 75%] Building CXX object CMakeFiles/CMakeDemo4.dir/main.cpp.o
[100%] Linking CXX executable CMakeDemo4
[100%] Built target CMakeDemo4
```

Exercise 1

```cpp
#include <iostream>
using namespace std;
int * create_array(int size)
{
   int arr[size];

   for(int i = 0; i < size; i++)
      arr[i] = i * 10;
   return arr;
}
int main()
{
   int len = 16;
   int *ptr = create_array(len);
   for(int i = 0; i < len; i++)
      cout << ptr[i] << " ";
   return 0;
}
```

What compilation warnings occur when you compile the program? Why?

What will happen if you ignore the warning and run the program?

Fix bugs of the program and run it correctly without memory leak.

Exercise 2

Define a function that swaps two values of integers. Write a test program to call the function and display the result.

You are required to compile the function into a static library “libswap.a”, and then compile and run your program with this static library.