南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

Lab 5, CMake

廖琪梅，王大兴

What is CMake?

CMake is an open-source, cross-platform family of tools designed to build, test and package software. **CMake** is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles and workspaces that can be used in the compiler environment of your choice.

For more information https://cmake.org/

CMake needs **CMakeLists.txt** to run properly.

A CMakeLists.txt consists of **commands** , **comments** and **spaces**.

- The **commands** include command name, brackets and parameters , the parameters are separated by spaces. Commands are not case sensitive.
- **Comments** begins with '#'.

1. A single source file in a project

The most basic project is an executable built from source code files. For simple projects, a three-line **CMakeLists.txt** file is all that is required.

```
cmake_minimum_required(VERSION 3.16)

project(hello)

add_executable(hello main.cpp)
```

Specifies the minimum required version of CMake. Use **cmake --version** in Vscode terminal window to check the cmake version in your computer.

Defines the project name.

Adds the hello executable target which will be built from main.cpp.

The first parameter indicates the filename of executable file.

The second parameter indicates the source file.

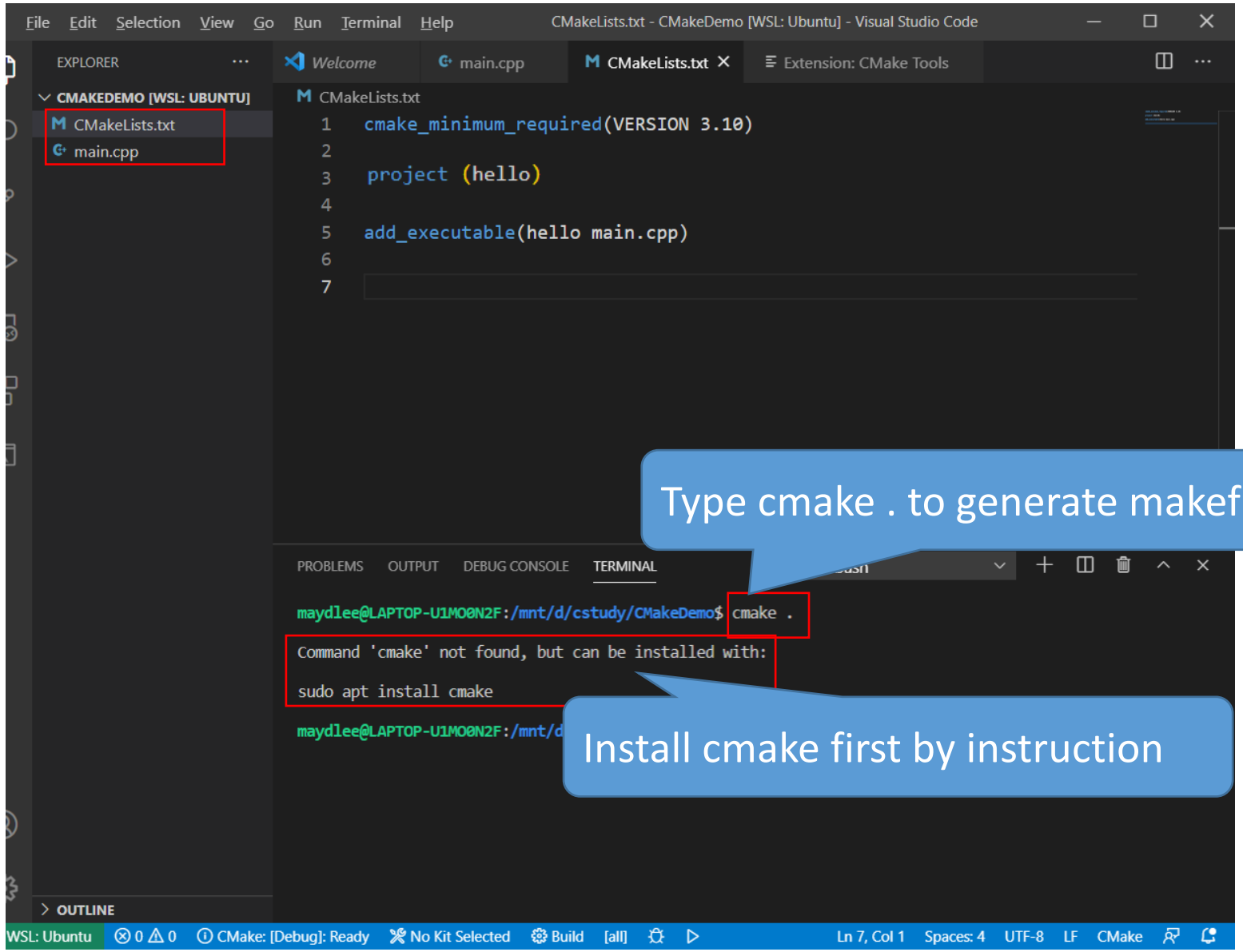
Store the CMakeLists.txt file in the same directory as the main.cpp.

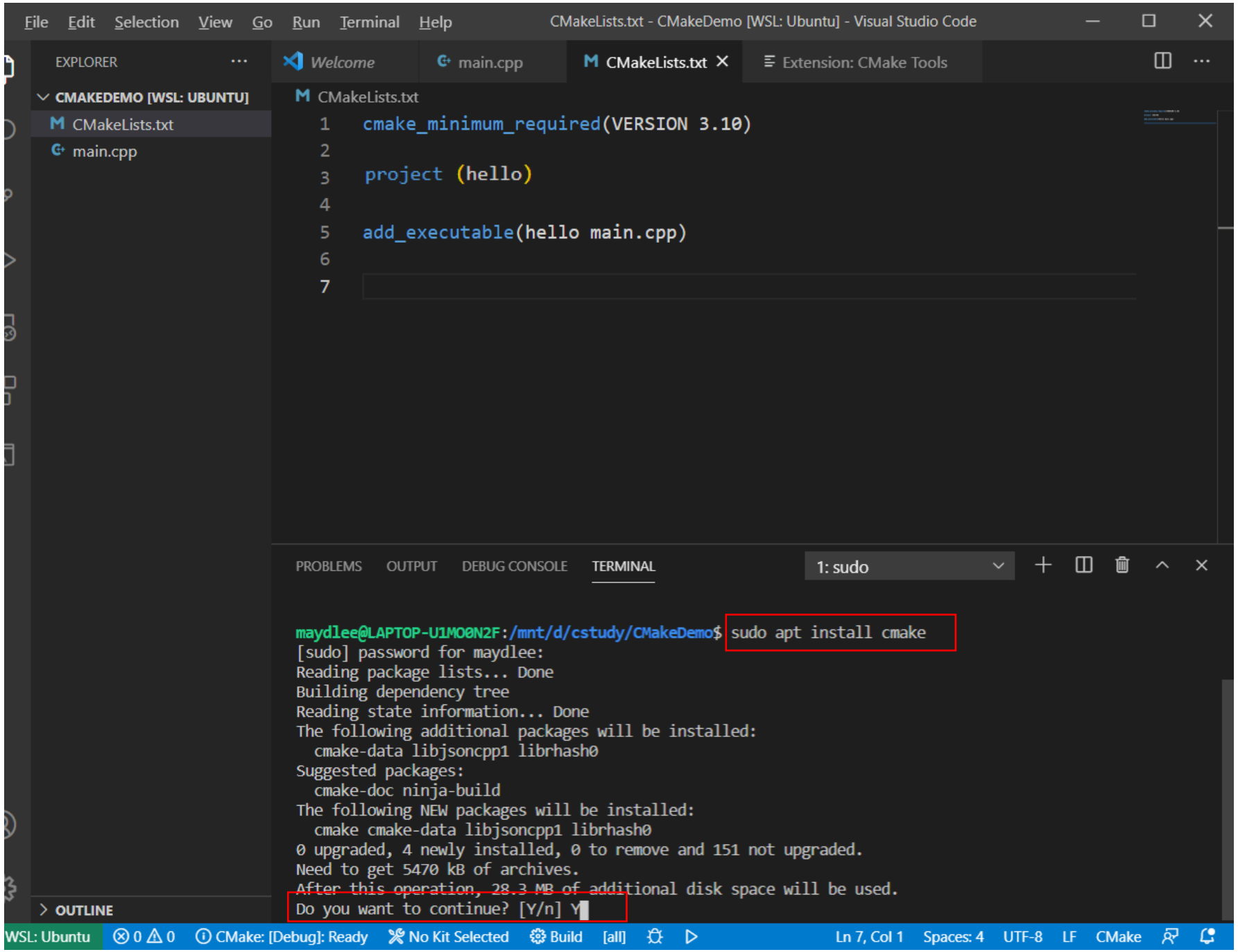
Suppose we have a main.cpp file

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

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

    return 0;
}
```


File Edit Selection View Go Run Terminal Help
CMakeLists.txt - CMakeDemo [WSL: Ubuntu] - Visual Studio Code
EXPLORER
CMAKEDEMO [WSL: UBUNTU]
CMakeLists.txt
main.cpp
Welcome
main.cpp
CMakeLists.txt
Extension: CMake Tools
CMakeLists.txt
1 cmake_minimum_required(VERSION 3.10)
2
3 project (hello)
4
5 add_executable(hello main.cpp)
6
7
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/CMakeDemo$ cmake .
Command 'cmake' not found, but can be installed with:
sudo apt install cmake
maydlee@LAPTOP-U1MO0N2F:/mnt/d
OUTLINE
WSL: Ubuntu
0 0
CMake: [Debug]: Ready
No Kit Selected
Build
[all]
Ln 7, Col 1
Spaces: 4
UTF-8
LF
CMake
Type cmake . to generate makefile
Install cmake first by instruction

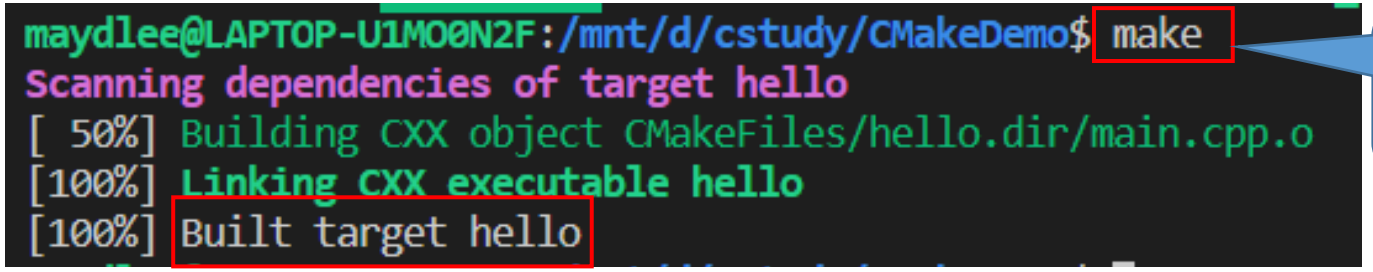

File Edit Selection View Go Run Terminal Help
CMakeLists.txt - CMakeDemo [WSL: Ubuntu] - Visual Studio Code
EXPLORER
CMAKEDEMO [WSL: UBUNTU]
CMakeLists.txt
main.cpp
Welcome
main.cpp
CMakeLists.txt
Extension: CMake Tools
CMakeLists.txt
1 cmake_minimum_required(VERSION 3.10)
2
3 project (hello)
4
5 add_executable(hello main.cpp)
6
7
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: sudo
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/CMakeDemo$ sudo apt install cmake
[sudo] password for maydlee:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
cmake-data libjsoncpp1 librhash0
Suggested packages:
cmake-doc ninja-build
The following NEW packages will be installed:
cmake cmake-data libjsoncpp1 librhash0
0 upgraded, 4 newly installed, 0 to remove and 151 not upgraded.
Need to get 5470 kB of archives.
After this operation, 28.3 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
OUTLINE
WSL: Ubuntu
0 0
CMake: [Debug]: Ready
No Kit Selected
Build
[all]
Ln 7, Col 1
Spaces: 4
UTF-8
LF
CMake

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/CMakeDemo$ cmake .
-- The C compiler identification is GNU 9.3.0
-- The CXX compiler identification is GNU 9.3.0##
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- w
-- 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/cstudy/CMakeDemo
```

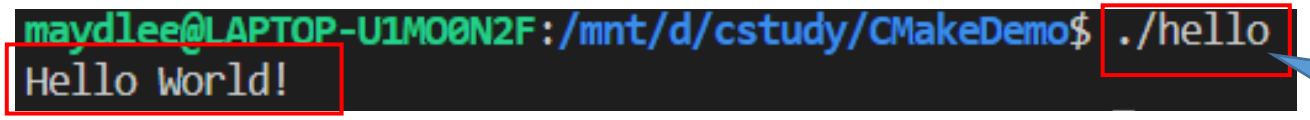
Run cmake to generate makefle, . indicates the makefile is stored in the current directory.

makefile file is created automatically after running cmake in the current directory.

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/CMakeDemo$ ls
CMakeCache.txt  CMakeFiles  CMakeLists.txt  Makefile  cmake_install.cmake  main.cpp
```



Execute make to compile the program.



Run the program

2. Multi-source files in a project

There are three files in the same directory.

```
./CmakeDemo2
    |
    +--- main.cpp
    |
    +--- function.cpp
    |
    +--- function.h
```

```
cmake_minimum_required(VERSION 3.10)

project(CmakeDemo2)

add_executable(CmakeDemo2 main.cpp function.cpp)
```

Add the function.cpp to the add_executable command.

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/CMakeDemo2$ cmake .
-- The C compiler identification is GNU 9.3.0
-- The CXX compiler identification is GNU 9.3.0
-- 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/cstudy/CMakeDemo2
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/CMakeDemo2$ make
Scanning dependencies of target CmakeDemo2
[ 33%] Building CXX object CMakeFiles/CmakeDemo2.dir/main.cpp.o
[ 66%] Building CXX object CMakeFiles/CmakeDemo2.dir/function.cpp.o
[100%] Linking CXX executable CmakeDemo2
[100%] Built target CmakeDemo2
```

2. Multi-source files in a project

If there are several files in directory, put each file into the add_executable command is not recommended. The better way is using **aux_source_directory** command.

aux_source_directory (<dir> <variable>)

The command finds all the source files in the specified directory indicated by <dir> and stores the results in the specified variable indicated by <variable>.

2. Multi-source files in a project

```
cmake_minimum_required(VERSION 3.10)

project(CmakeDemo2)

aux_source_directory(. DIR_SRCS)

add_executable(CmakeDemo2 ${DIR_SRCS})
```

Store all files in the current directory into DIR_SRCS.

Compile the source files in the variable by **${ }**
into an executable file named CmakeDemo2

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/CMakeDemo2$ cmake .
-- The C compiler identification is GNU 9.3.0
-- The CXX compiler identification is GNU 9.3.0
-- 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/cstudy/CMakeDemo2
```

3. Multi-source files in a project in different directories

./CMakeDemo3

```
|
+--- src/
|     |
|     +-- main.cpp
|     +-- function.cpp
|
+--- include/
      |
      +--- function.h
```

All .cpp files are in the src directory

Include the header file which is stored in include directory.

We write CMakeLists.txt in CmakeDemo3 folder.

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

# project information
project(CMakeDemo3)

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

# add the directory of include
include_directories(include)

# Specify the build target
add_executable(CMakeDemo3 ${DIR_SRCS})
```


```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/CMakeDemo3$ cmake .
-- The C compiler identification is GNU 9.3.0
-- The CXX compiler identification is GNU 9.3.0
-- 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/cstudy/CMakeDemo3
maydlee@LAPTOP-U1MO0N2F:/mnt/d/cstudy/CMakeDemo3$ make
Scanning dependencies of target CMakeDemo3
[ 33%] Building CXX object CMakeFiles/CMakeDemo3.dir/src/function.cpp.o
[ 66%] Building CXX object CMakeFiles/CMakeDemo3.dir/src/main.cpp.o
[100%] Linking CXX executable CMakeDemo3
[100%] Built target CMakeDemo3
```

For more about Cmake(cmake tutorial):
https://cmake.org/cmake/help/latest/guide/tutorial/index.html

https://riptutorial.com/cmake

Exercise 1

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

int main()
{
   char *pc, cc = 'A';
   int *pi, ii = 10;
   float *pf, ff = 23.4f;
   double *pd, dd = 123.78;

   pc = &cc;
   pi = &ii;
   pf = &ff;
   pd = &dd;

   cout << "sizeof(cc) = " << sizeof(cc) << ", sizeof(pc) = " << sizeof(pc)  << endl;
   cout << "sizeof(ii) = " << sizeof(ii) << ", sizeof(pi) = " << sizeof(pi) <<  endl;
   cout << "sizeof(ff) = " << sizeof(ff) << ", sizeof(pf) = " << sizeof(pf) <<  endl;
   cout << "sizeof(dd) = " << sizeof(dd) << ", sizeof(pd) = " << sizeof(pd) <<  endl;

   cout << "&pc = " << &pc << ", pc = " << static_cast <void*>(pc) << ", *pc = " << *pc << endl;
   cout << "&pi = " << &pi << ", pi = " << pi << ", *pi = " << *pi << endl;
   cout << "&pf = " << &pf << ", pf = " << pf << ", *pf = " << *pf << endl;
   cout << "&pd = " << &pd << ", pd = " << pd << ", *pd = " << *pd << endl;

   const char *msg = "C/C++ programming is fun.";
   const char *copy;

   copy = msg;

   cout << "msg = " << msg << ",its address is: " << (void*)msg << ", &msg = " << &msg << endl;
   cout << "copy= " << copy << ",its address is: " << (void*)copy << ", &copy= " << &copy << endl;

   return 0;
}
```

Run the program and explain the result to SA.

Exercise 2

```
#include<stdio.h>

int main()
{
   int a[]={2,4,6,8,10},y=1,*p;
   p=&a[1];

   printf("a = %p\np = %p\n",a, p);

   for(int i = 0; i < 3; i++)
      y += *(p+i);

   printf("y = %d\n\n",y);

   int b[5]={1,2,3,4,5};
   int *ptr=(int*)(&b+1);

   printf("b = %p\nb+4 = %p\nptr = %p\n",b,b+4,ptr);
   printf("%d,%d\n",*(b+1),*(ptr-1));

   return 0;

}
```

Run the program and explain the result to SA.

Exercise 3

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

int main()
{
   int a[][4]={1,3,5,7,9,11,13,15,17,19};
   int *p=*(a+1);
   p += 3;
   cout << "*p++ = " << *p++ << ",*p = " << *p << endl;

   const char *pc = "Welcome to programming.", *r;
   long *q = (long *)pc;
   q++;
   r = (char *)q;

   cout << r << endl;

   unsigned int m = 0x3E56AF67;
   unsigned short *pm = (unsigned short *) &m;

   cout << "*pm = " <<  hex << *pm << endl;

   return 0;
}
```

Run the program and explain the result to SA.

Exercise 4

Write a program that use ***new*** to allocate the array dynamically for five integers.

- The five values will be stored in an array using a pointer.
- Print the elements of the array in reverse order using a pointer.

Exercises 5

Declare a structure named **stuinfo** and two function prototypes below in a **stuinfo.hpp**. Implement the two functions in a **stufun.cpp**. Write a **main.cpp** which contains main() and demonstrate all the features of the prototyped functions.

Write a **MakeLists.txt** for cmake to create Makefile automatically. Run cmake and make, and then run the program at last.

```
struct stuinfo
{
    char name[20];
    double score[3];
    double ave;
};
```

Function prototypes:

- **void inputstu(stuinfo stu[] , int n),** asks the user to enter each of the preceding items of information to set the corresponding members of the structure and compute the average score for each student.
- **void showstu(stuinfo stu[] , int n)** ,displays the contents of the structure, one student one line.