Lab 1 Report

Zi Shengbo 23020036099 Lab group: 32.

Lab date: 6th November 2025. Report date: 9th November 2025.

Summary

The summary provides an overview of this Lab 1 report. It aims to concisely outline the experiment's objectives, key procedures, and main conclusions, serving as a stand-alone section for readers to grasp the report's core content without repeating the introduction. This Lab 1 focuses on Linux system operations, pro- gramming tools, and C++ basics, with the goal of establishing a solid foundation for future technical learning. The report details the experimental process, indi- vidual work, discussions, and conclusions, ensuring clarity and completeness in conveying the learning outcomes.

1. Lab Introduction and Team Information

Lab 1 is an introductory practical experiment centered around Linux system operations, programming tools, and basic programming languages. Its core objective is to assist us in setting up a professional development environment, mastering the usage methods of key tools, and establishing a basic understanding of C++ programming, thereby laying a solid foundation for more complex technical learning in the future.

In terms of specific content, it is required to master the installation method of the Ubuntu 20.04 system, be able to select the appropriate installation medium according to the scenario requirements (selecting a virtual machine for classroom learning and a physical disk for the production environment), and complete the system deployment. Moreover, proficiently use basic Shell operations, including basic commands such as pwd, ls, cd, and practical operations such as wget and tar. Master the usage of pipes (such as "cat 1.txt — sort") and output redirection (such as "ls > output.txt"), and achieve efficient control of the Linux system. Another key point is to hope that we can utilize resources such as the Git official website, CMake tutorials, etc., to learn the core usage of Git version control and CMake project building tools, and possess the ability of code management and project building; relying on resources such as "C++ Primer", related tutorials, and W3School, understand the basic syntax and programming logic of C++, and complete the beginner's C++ programming.

The group participating in Lab 1 consists of two members, namely myself and Jiang Yuchen. In terms of executing the experimental tasks, we adopted a division of labor model where each of us independently completed all the contents of Lab 1. This divi- sion method ensures that both of us can comprehensively engage with and master all

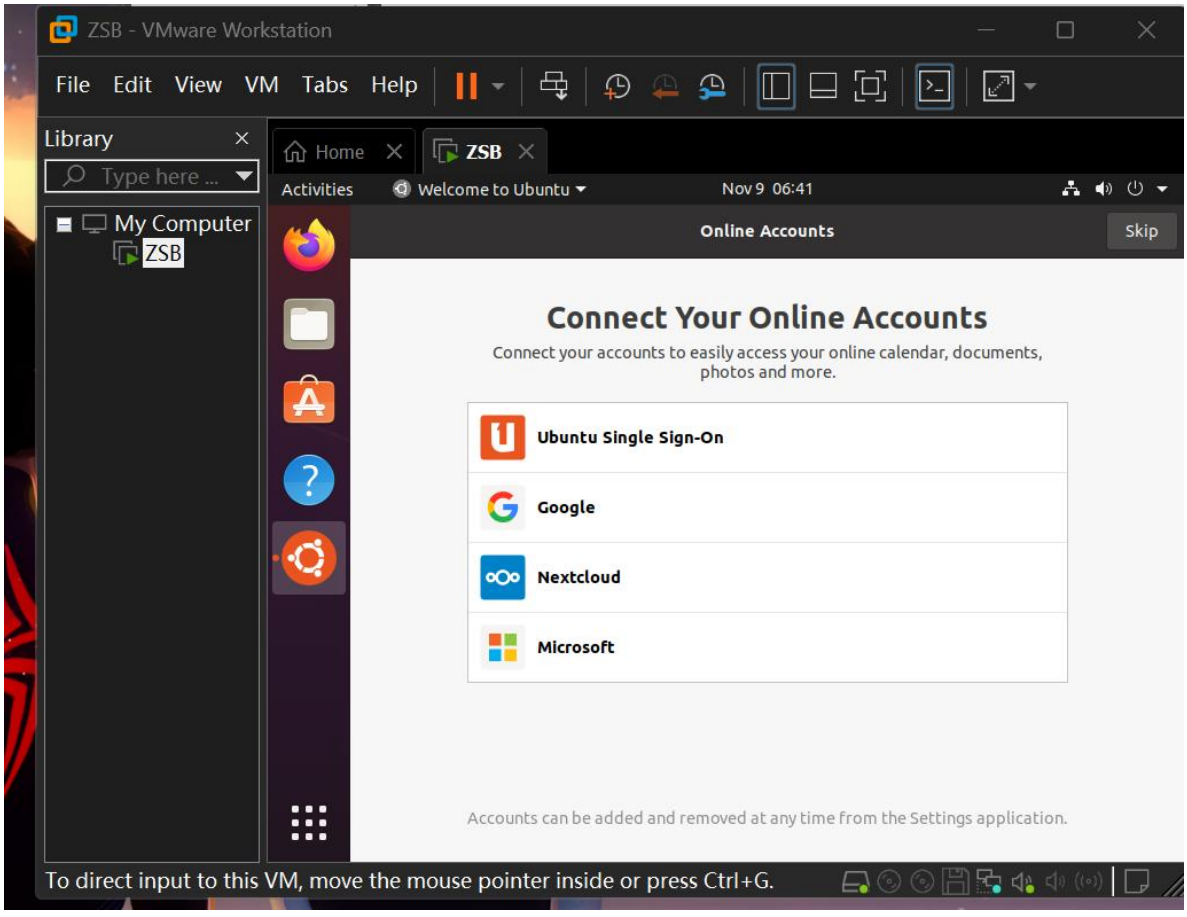
the knowledge points involved in the Ubuntu 20.04 installation, Shell basics, Git opera- tions, CMake usage, and C++ fundamentals. Through independent practice, we deepen our understanding of each operation step and theoretical concepts. Moreover, after com- pleting the subsequent experiments, we can share our operational experiences, encounter problems, and solutions with each other, achieving experience sharing and identifying and rectifying deficiencies, thereby further consolidating the learning effect of the experiment and ensuring that both of us can achieve the learning goals set for Lab 1.

2. Individual Work

PART1

The first part requires us to install Ubuntu 20.04. Using a virtual machine for installation is more suitable for the experimental scenario. After researching, the Windows system is more suitable for downloading Ubuntu using VMware Workstation Pro. I first downloaded the Ubuntu image from Tsinghua Mirror at https://mirrors.tuna.tsinghua.edu.cn/ubuntu-releases/focal/ubuntu-20.04.6-desktop-amd64.iso. After the download is complete, VMware Workstation Pro can be directly used. Next, we need to download the official VMware Workstation Pro from the official website: https://www.vmware.com/products/desktop-hypervisor/workstation-and-fusion.

After setting up the platform and Ubuntu, I referred to the relevant tutorial https://blog.csdn.net/qq_62888264/article/details/145102532 to learn how to download Ubuntu in VMware Workstation Pro. After following the steps, I successfully downloaded Ubuntu 20.04 on the virtual machine.



After this, it is necessary to switch the Ubuntu 20.04 software to the Tsinghua repository. First, switch to the administrator mode and back up the original sources.list. The following commands will be used: “sudo -s” and “cp /etc/apt/sources.list /etc/apt/sources.list.bak”

```
zsb@ubuntu:~/Desktop$ sudo -s
[sudo] password for zsb:
root@ubuntu:/home/zsb/Desktop# cp /etc/apt/sources.list /etc/apt/sources.list.bak
root@ubuntu:/home/zsb/Desktop# nano /etc/apt/sources.list
root@ubuntu:/home/zsb/Desktop# nano /etc/apt/sources.list
root@ubuntu:/home/zsb/Desktop# apt update
Get:1 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal InRelease [265 kB]
Get:2 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates InRelease [128 kB]
Get:3 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-backports InRelease [128 kB]
Get:4 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/main i386 Packages [718 kB]
Hit:5 http://security.ubuntu.com/ubuntu focal-security InRelease
Get:6 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/main amd64 Packages [970 kB]
Get:7 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/main Translation-en [506 kB]
Get:8 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/main amd64 DEP-11 Metadata [494 kB]
Get:9 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/main DEP-11 48x48 Icons [98.4 kB]
Get:10 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/main DEP-11 64x64 Icons [163 kB]
Get:11 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/main DEP-11 64x64@2 Icons [15.8 kB]
Get:12 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/main amd64 c-n-f Metadata [29.5 kB]
Get:13 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/restricted i386 Packages [8,112 B]
Get:14 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/restricted amd64 Packages [22.0 kB]
Get:15 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/restricted Translation-en [6,212 B]
Get:16 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/restricted amd64 c-n-f Metadata [392 B]
Get:17 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/universe i386 Packages [4,642 kB]
```

Then, on GitHub, search for the source address to be replaced from the Tsinghua repository at https://mirror.tuna.tsinghua.edu.cn/help/ubuntu. Open and edit the /etc/apt/sources.list file using the nano command, and replace the content in /etc/apt/sources.list with the source address provided by the Tsinghua repository.

```
  GNU nano 4.8
默认注释了源码镜像以提高 apt update 速度，如有需要可自行取消注释
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-updates main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-updates main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-backports main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-backports main restricted universe multiverse

deb http://security.ubuntu.com/ubuntu/ focal-security main restricted universe multiverse
# deb-src http://security.ubuntu.com/ubuntu/ focal-security main restricted universe multiverse

# 预发布软件源，不建议启用
# deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-proposed main restricted universe multiverse
# # deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-proposed main restricted universe multiverse
# deb-src http://us.archive.ubuntu.com/ubuntu/ focal-backports main restricted universe multiverse

## Uncomment the following two lines to add software from Canonical's
## 'partner' repository.
## This software is not part of Ubuntu, but is offered by Canonical and the
## respective vendors as a service to Ubuntu users.
# deb http://archive.canonical.com/ubuntu focal partner
# deb-src http://archive.canonical.com/ubuntu focal partner
```

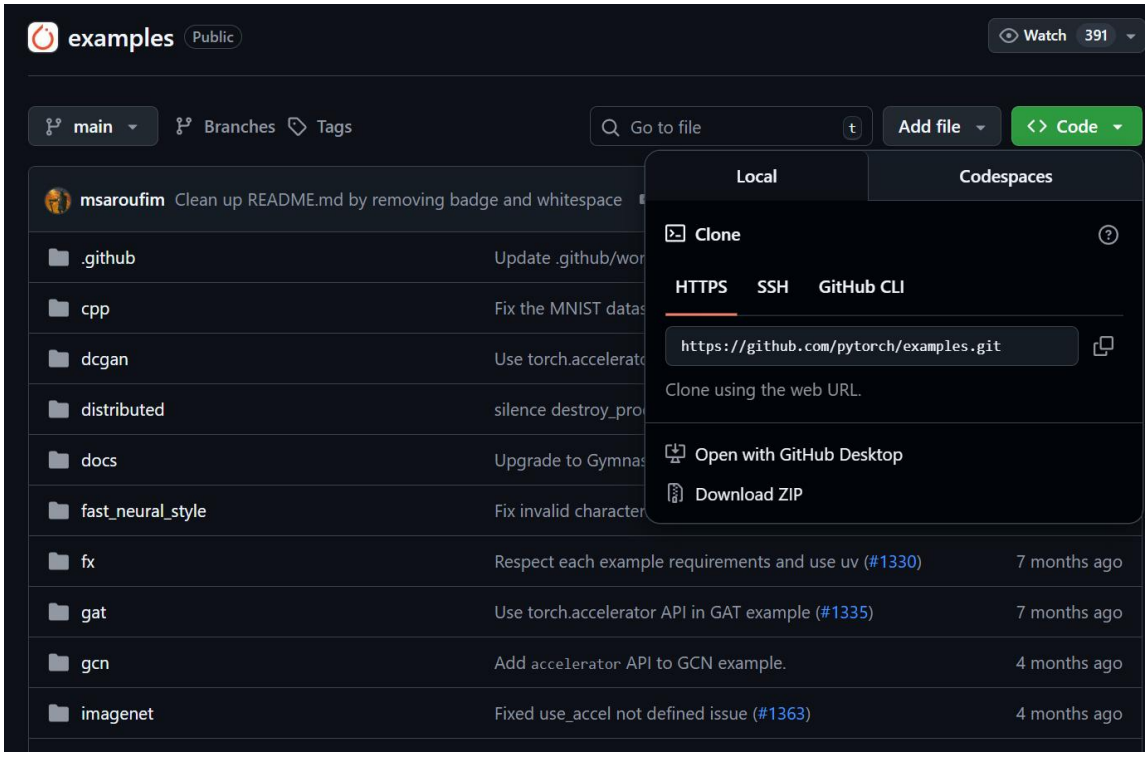
PART2

By executing the command "sudo apt install build-essential cmake git", you can directly download and install the necessary tools: Git, build-essential.

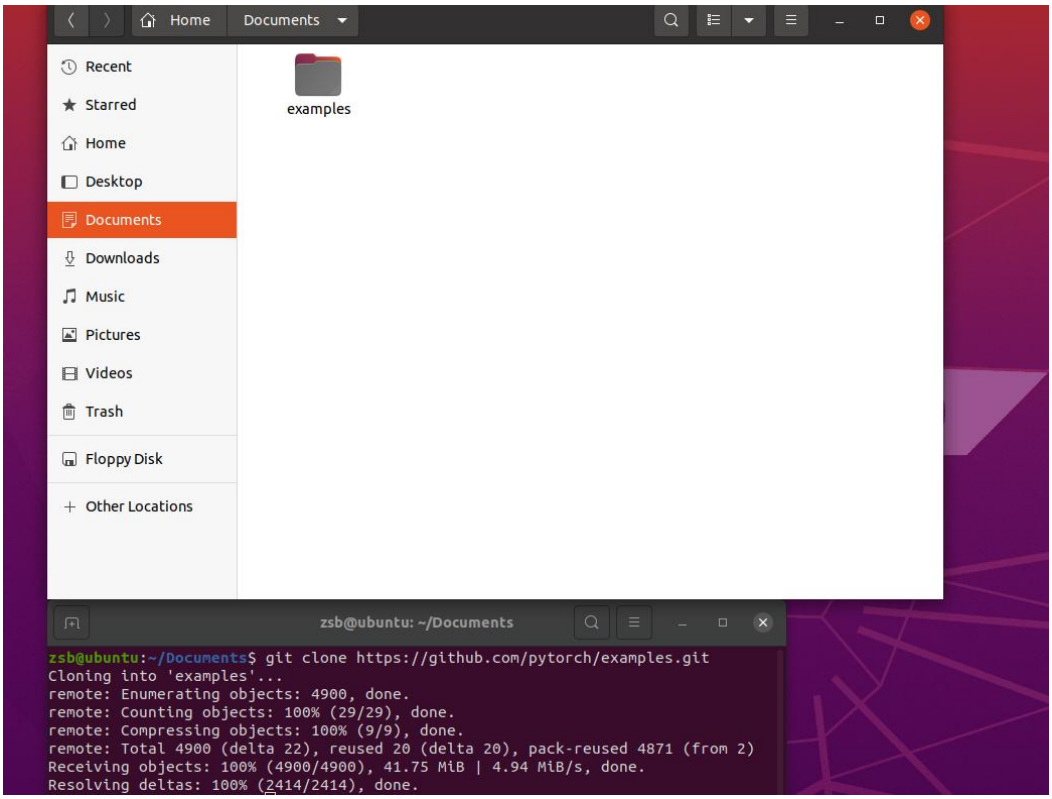
```
root@ubuntu:/home/zsb/Desktop# sudo apt install build-essential cmake git
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu cmake-data cpp-9 dpkg-dev fakeroot g++ g++-9 gcc gcc-10-base gcc-9 gcc-9-base git-man libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libasan5 libatomic1 libbinutils
  libc-dev-bin libc6 libc6-dbg libc6-dev libcc1-0 libcrypt-dev libctf-nobfd0 libctf0 libcurl4 liberror-perl libfakeroot libgcc-9-dev libgcc-s1 libgomp1 libitm1 libjsoncpp1 liblsan0 libquadmath0 librhash0 libstdc++-9-dev libstdc++6 libtsan0 libubsan1
  linux-libc-dev make manpages-dev
Suggested packages:
  binutils-doc cmake-doc ninja-build gcc-9-locales debian-keyring g++-multilib g++-9-multilib gcc-9-doc gcc-multilib autoconf automake libtool flex bison gcc-doc gcc-9-multilib git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui
  gitk gitweb git-cvs git-mediawiki git-svn glibc-doc libstdc++-9-doc make-doc
The following NEW packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu build-essential cmake cmake-data dpkg-dev fakeroot g++ g++-9 gcc gcc-9 git git-man libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libasan5 libbinutils libc-dev-bin
  libc6-dev libcrypt-dev libctf-nobfd0 libctf0 libcurl4 liberror-perl libfakeroot libgcc-9-dev libitm1 libjsoncpp1 liblsan0 libquadmath0 librhash0 libstdc++-9-dev libtsan0 libubsan1 linux-libc-dev make manpages-dev
The following packages will be upgraded:
  cpp-9 gcc-10-base gcc-9-base libatomic1 libc6 libc6-dbg libcc1-0 libgcc-s1 libgomp1 libstdc++6
10 upgraded, 39 newly installed, 0 to remove and 327 not upgraded.
Need to get 79.3 MB of archives.
After this operation, 238 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libc6-dbg amd64 2.31-0ubuntu9.18 [20.2 MB]
Get:2 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libatomic1 amd64 10.5.0-1ubuntu1~20.04 [9,284 B]
Get:3 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 gcc-10-base amd64 10.5.0-1ubuntu1~20.04 [20.8 kB]
Get:4 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libstdc++6 amd64 10.5.0-1ubuntu1~20.04 [501 kB]
Get:5 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libgomp1 amd64 10.5.0-1ubuntu1~20.04 [102 kB]
Get:6 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libcc1-0 amd64 10.5.0-1ubuntu1~20.04 [48.8 kB]
Get:7 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libgcc-s1 amd64 10.5.0-1ubuntu1~20.04 [41.8 kB]
Get:8 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libc6 amd64 2.31-0ubuntu9.18 [2,720 kB]
Get:9 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 binutils-common amd64 2.34-6ubuntu1.11 [208 kB]
Get:10 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libbinutils amd64 2.34-6ubuntu1.11 [475 kB]
Get:11 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libctf-nobfd0 amd64 2.34-6ubuntu1.11 [48.2 kB]
Get:12 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libctf0 amd64 2.34-6ubuntu1.11 [46.4 kB]
Get:13 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 binutils-x86-64-linux-gnu amd64 2.34-6ubuntu1.11 [1,612 kB]
Get:14 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 binutils amd64 2.34-6ubuntu1.11 [3,380 B]
Get:15 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libc-dev-bin amd64 2.31-0ubuntu9.18 [71.7 kB]
Get:16 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 linux-libc-dev amd64 5.4.0-216.236 [1,111 kB]
Get:17 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/main amd64 libcrypt-dev amd64 1:4.4.10-10ubuntu4 [104 kB]
Get:18 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libc6-dev amd64 2.31-0ubuntu9.18 [2,520 kB]
Get:19 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 cpp-9 amd64 9.4.0-1ubuntu1~20.04.2 [7,502 kB]
Get:20 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 gcc-9-base amd64 9.4.0-1ubuntu1~20.04.2 [18.9 kB]
Get:21 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libitm1 amd64 10.5.0-1ubuntu1~20.04 [26.2 kB]
Get:22 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libasan5 amd64 9.4.0-1ubuntu1~20.04.2 [2,752 kB]
Get:23 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 liblsan0 amd64 10.5.0-1ubuntu1~20.04 [835 kB]
Get:24 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libtsan0 amd64 10.5.0-1ubuntu1~20.04 [2,016 kB]
Get:25 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libubsan1 amd64 10.5.0-1ubuntu1~20.04 [785 kB]
Get:26 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates/main amd64 libquadmath0 amd64 10.5.0-1ubuntu1~20.04 [146 kB]
```

PART3

First, I need to attempt to clone a project from platforms like GitHub or GitLab to our local machine. I logged into GitHub and randomly selected an "examples" project. I chose to clone it to my localmachine.



Then I opened the Documents folder of Ubuntu 20.04, opened the terminal, and created a folder with the same name as the git repository in the current Documents directory. Using the command "git clone https://github.com/pytorch/examples.git", I cloned this project to my local machine.



Next, I need to create a basic C++ program to understand the meaning of the program's syntax, learn how to compile and run the project. Here, many functions and formatting issues of the Linux

language will also be introduced.The project structure requirements are as follows. In each sub-file, the corresponding content needs to be compiled.

```
- lab1_demo
  - CMakeLists.txt
  - main.cpp
  - my_lib.hpp
  - my_lib.cpp
```

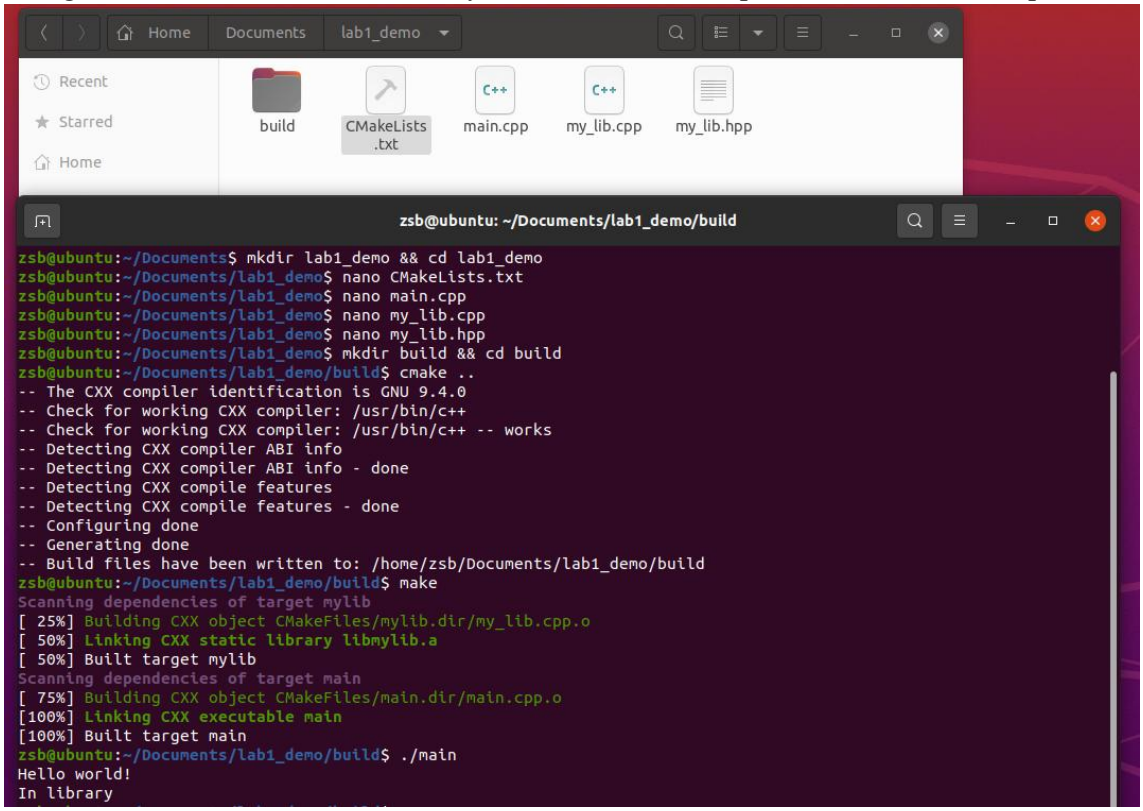
```
cmake_minimum_required(VERSION 3.10)

project(
  hello_world
  VERSION 1.0
  LANGUAGES CXX)

add_library(mylib my_lib.cpp my_lib.hpp)
add_executable(main main.cpp)
target_link_libraries(main PRIVATE mylib)
```

```
#include <iostream>
#include "my_lib.hpp"

int main() {

    std::cout << "Hello world!" << std::endl;
    std::cout << my_lib_function() << std::endl;

    return 0;
}
```

```
#include <string>
#include "my_lib.hpp"

std::string my_lib_function() {
    return "In library";
}
```

```
#pragma once

#include <string>

std::string my_lib_function();
```

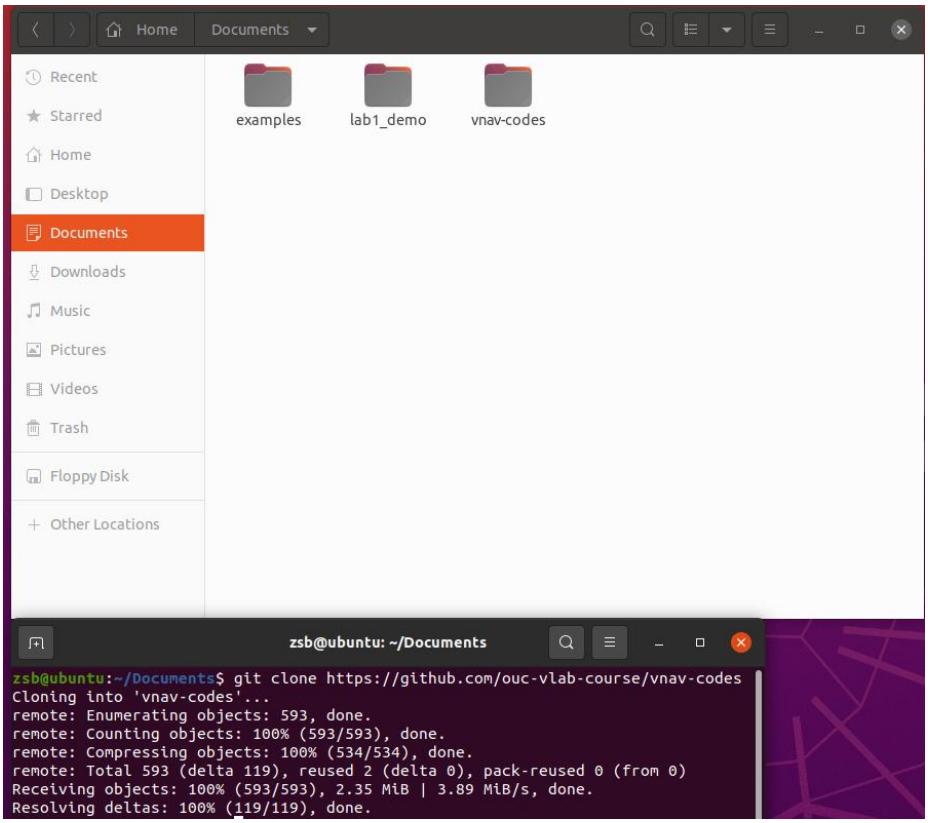
I want to first create the "lab1_demo" directory and navigate to this interface. Then, I will proceed with the subsequent project within this interface. Next, I will add and edit four files as per the specified requirements. How to compile this project requires us to create a "build" folder within the project folder and use the "cd" command to enter this folder (note that the execution path of the command should be within the project folder). So, I created "build" and navigated to it. This folder is used to record the data and content during the code execution process. In the "build" folder, use "cmake .." 。 ".." Perform the CMake operation on the parent directory, namely lab1_demo. Then use "make" to generate the executable file. Finally, use "./main" for compilation to obtain the output result.



PART4

Ⅰ.

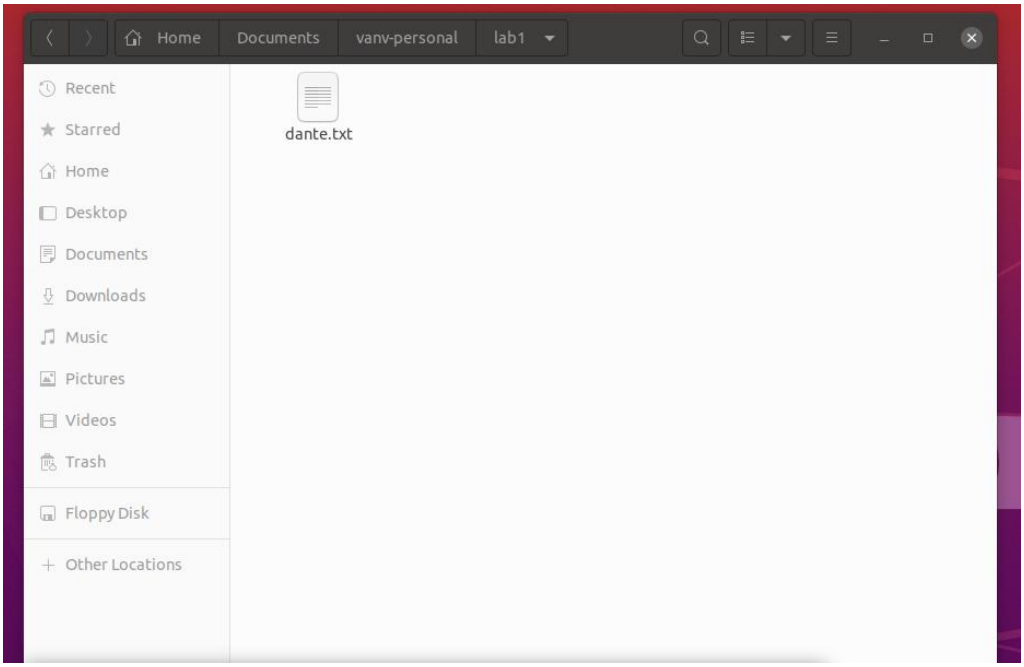
Just like in PART3, clone the project from the provided GitHub link and save the "vnav_codes" file to your local machine.

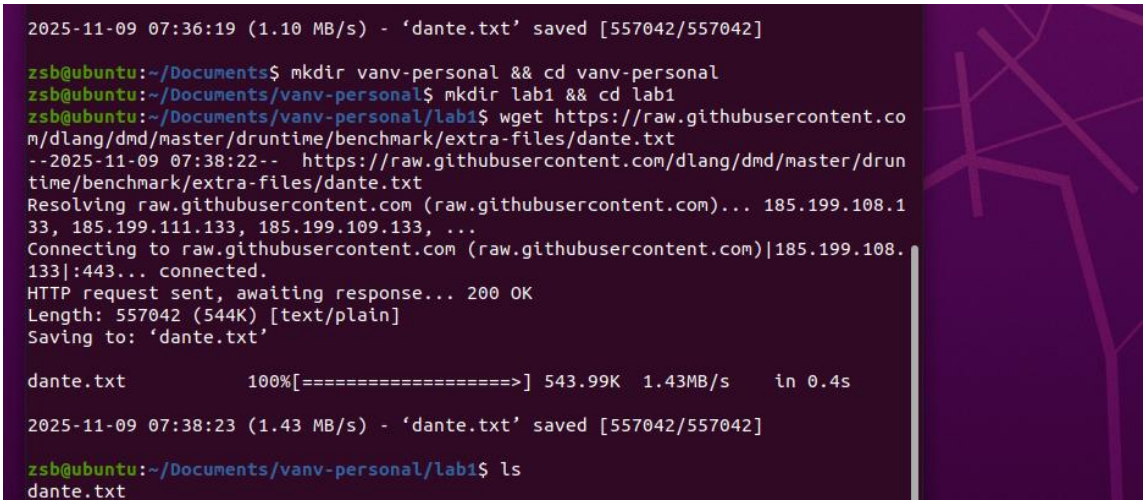


Ⅱ.

This part requires us to be proficient in using the Linux language, and to understand Shell and related formats and contents.

Considering that exercise2 needs to be operated in the vnan_personal/lab1 directory, I first created and navigated to the vnan_personal directory, and then created and navigated to the lab1 directory. Next, use the wget command to download the target file from the website specified in the command. After the download was completed, to verify that the dante.txt file was successfully saved in the vnan_personal/lab1 directory, I used the ls command to check all the files in that directory. I found that dante.txt appeared in the correct location.

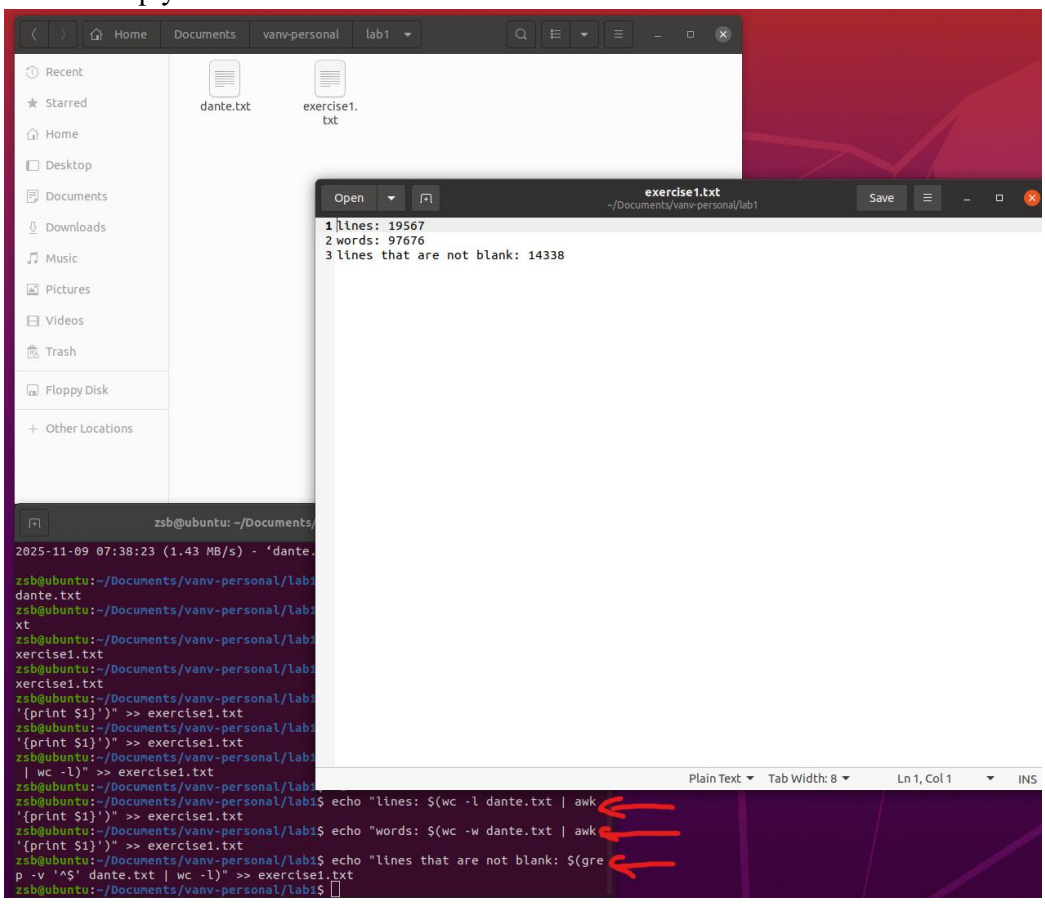




Next, we need to perform :

Exercise 1 (use command wc): create file exercise1.txt, and calculate these pieces of information: ① How many lines does this file contain? ② How many words does this file contain? ③ How many lines are not blank?

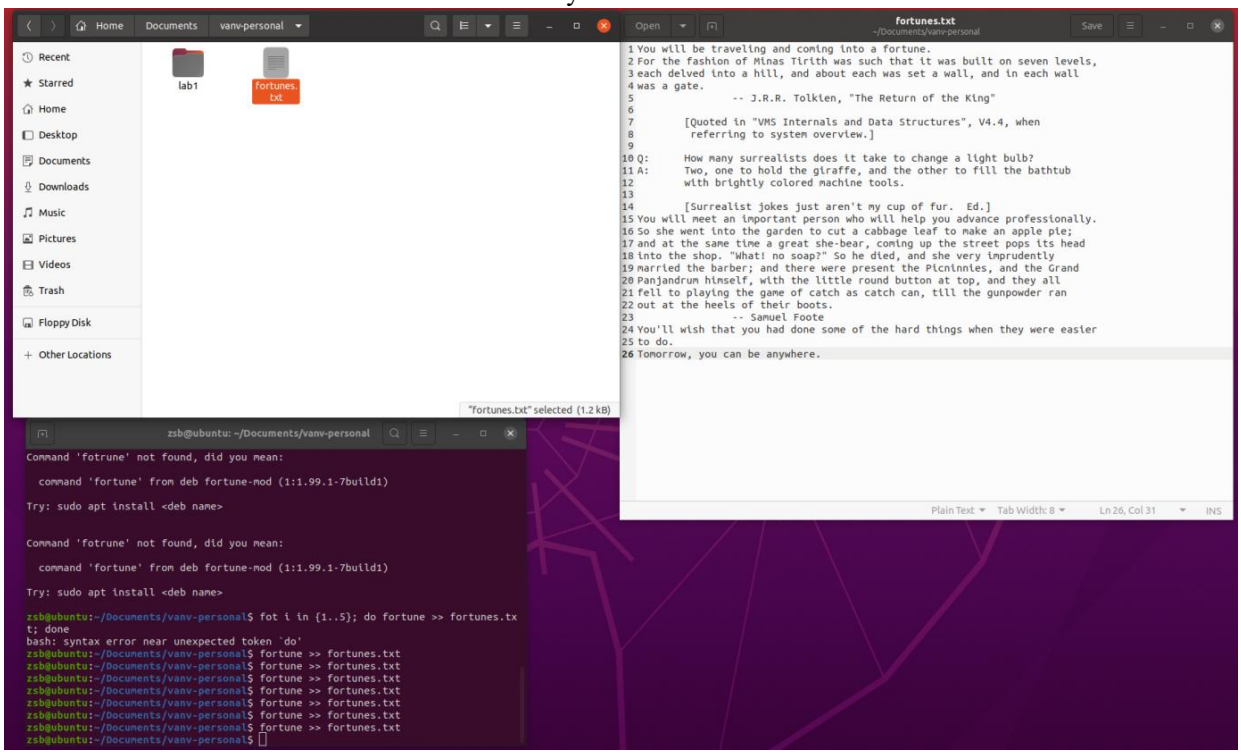
I completed this exercise using the instruction indicated by the red arrow in the picture. First, execute the command in parentheses. wc stands for word count and is used for counting. wc -l dante.txt counts the lines of dante.txt and the output is 19567 dante.txt. I only want to keep the preceding numbers, so I considered using the awk command. First, let the output of the previous part serve as the input for the following part. awk '{print $1}' makes the previous output a table, where $1 represents the content of the first column, which is exactly what I want to output only the numbers. The same applies to the subsequent steps. The third part is to find non-empty lines. I choose to select the empty lines and then reverse the selection. Empty lines can be represented as '^$' which means the beginning and the end are adjacent. Searching for text that does not meet this format is the text I want after deleting the empty lines. Then, use this text as input and perform wc -l, and the result will be the number of non-empty lines I want.



For all the results, I redirected them using ">>" to append the obtained output at the end of exercise1.txt, resulting in the final exercise1.txt. The three lines respectively counted the number of lines, the number of characters, and the number of non-empty lines in the dante.txt file.

Exercise 2: using apt to install fortune-mod, Run fortune 5 more times and each time redirect the output to a file called fortunes.txt in ~/vnav-personal/lab1(Hint: do not recreate the file 5 times - each time a new proverb should be added to the end of fortunes.txt)

I directly followed the instructions and ran the "fortune" command.It will randomly generate a proverb. The output of the "fortune" command was redirected and appended to the end of the "fortunes.txt" file. >> The file "forturns.txt" will be automatically created as "forturns.txt".



III.

Operators

1. i = 2, j = 1. Since the initial value of i is 0, when executing j = ++i, i first increments by 1 and then assigns the value to j. Therefore, at this point, j = 1 and i = 1. When executing "j = i++", first the value of i is assigned to j, which is 1. Then i is incremented by 1, so now i = 2. Therefore, in the end, i equals 2 and j equals 1.
2. b,and the cursor is at the beginning of the next line.Because this is a ternary operation, the condition is first evaluated to determine whether it is true or false. Since 42 < 42 is false, therefore the output is equal to: the subsequent value, which is b. After outputting the data, endl is used to add a newline character at the end and then the buffer is refreshed, causing the cursor to land at the beginning of the next line.

References and Pointers

1. 10 10.First, an integer variable i was declared, followed by an integer reference ri, both pointing to the same memory address. When executing i = 5, the value of i becomes 5, and the value accessed through ri at this time is also 5. Then, executing ri = 10, since ri is a reference to i, this assignment operation actually modifies the memory content corresponding to i, updating the value of i to 10. Finally, when using std::cout to output i and ri, it displays as 10 10 and the cursor lands at the beginning of the next line.
2. 1764. First, an integer variable i was initialized with a value of 42. Then, a pointer j was defined, which pointed to the memory address of variable i through the & operator. At this point, j stores the address of i, and *j represents accessing the value stored at that address, which is the current value of i, 42. *j = *j * *j; 。 This statement multiplies *j (which is the value of i, 42) by itself and writes the result, 1764, back to the variable i through the pointer j. Thus, the value of i is updated to 1764. Finally, in the output statement, *j is used again to access the current value of the variable i it points to, which

is 1764, and a newline character is added through endl to move the cursor to the beginning of the next line.
3. 0.First, an array named "i" containing four integer elements was declared and initialized, with the initial values being {42, 24, 42, 24}. *(i + 2) = *(i + 1) - i[3] employs two equivalent ways of accessing array elements. *(i + 2) represents the third element of the array, *(i + 1) represents the second element of the array, and i[3] represents the fourth element of the array. The value of the second element is subtracted from the value of the fourth element and assigned to the third element. At this point, *(i + 2) = 24 - 24 = 0. Finally, when accessing the third element of the array in the output statement, the value of this element has been updated to 0, so the output is 0, and endl is used to move the cursor to the beginning of the next line.
4. 0.First, a function named "reset" was defined, with an integer reference parameter "& i" as its argument. Inside the function, the variable represented by the reference parameter "& i" was assigned the value 0. Then, in the main program, an integer variable "j" was declared, with an initial value of 42. When "reset(j)" was called, "j" was bound to the parameter "& i", which meant that "i" became an alias for "j". Therefore, the execution of "i = 0" inside the function actually directly modified the value of "j", and the output value of "j" became 0, and the cursor was moved to the beginning of the next line.

Numbers
1. The main differences lie in the amount of memory they occupy and the range of values they can represent. Generally speaking, short occupies the least memory, usually 2 bytes, and has the smallest range; int has a size usually equal to the machine word length, usually 4 bytes, and has a range wider than short type; long is at least as large as int, usually 4 or 8 bytes; and long long is the largest among these types, usually 8 bytes, and can represent the widest range.
2. The main differences between float and double lie in precision, size and range. Float is a single-precision floating-point number, usually occupying 4 bytes and providing approximately 6-7 significant digits; while double is a double-precision floating-point number, usually occupying 8 bytes and providing approximately 15-16 significant digits. Therefore, it consumes more memory. The final value of i is: 3. When a floating-point number 3.14 is assigned to an integer variable i, a type conversion occurs. The decimal part is directly discarded, only the integer part 3 is retained, and then this integer value is assigned to the variable i.
3. For signed type,the first bit is used as the sign bit, where 0 represents a positive number and 1 represents a negative number. While unsigned types can only represent non-negative numbers, all bits are used to represent the magnitude of the number. c=255.Because the binary complement representation of -1 is 11111111. Since c is of unsigned type, this bit pattern is directly interpreted as an unsigned integer. 11111111 represents 255 in an unsigned 8-bit integer. So in the end, c was assigned the value of 255.
4. 0.First, the integer variable i was initialized and given an initial value of 42. Then the program entered the conditional judgment statement if (i). In C++, for the potential rule of the if condition expression, any non-zero value will be converted to the boolean value true, while zero value will be converted to false. Since the current value of i is 42, which is a non-zero value, the condition judgment result is true, and the program enters the if branch to execute the assignment statement i = 0.

Ⅳ.

The main task of this experiment is to create a class named RandomVector for generating and processing random vector data. Specifically, the relevant code from Lab1 needs to be implemented in the RandomVector folder. The class definition is located in the header file random_vector.h, while the specific member function implementations need to complete all the TODO sections in the file random_vector.cpp. The core functions of this class include: initializing a double-precision floating-point vector of a specified size with random numbers within the given maximum range for its elements; calculating and returning the average, maximum, and minimum values of the vector's elements; printing all the elements of the vector; and generating and printing the histogram of the data based on the specified number of bins. It is particularly important to note that the <algorithm> header file functions cannot be used during the implementation process to enhance the ability of low-level programming. After completing the code writing, the specific compilation command “g++ -std=c++11

-Wall -pedantic -o random_vector main.cpp random_vector.cpp" should be used for compilation to ensure that the code complies with the C++11 standard and can pass the strict warning checks. Finally, the correctly implemented program will be able to output the statistical information of the vector and provide an intuitive histogram display.

*random_vector.cpp

```cpp
#include "random_vector.h"
// TODO: add any include you might require
#include <cstdlib>
#include <iostream>
#include <cmath>
RandomVector::RandomVector(int size, double max_val) {
  // TODO: Write your code here
  vect.resize(size);
  for (int i = 0; i < size; i++) {
      vect[i] = (static_cast<double>(std::rand()) / RAND_MAX) * max_val;
  }
}
void RandomVector::print(){
  // TODO: Write your code here
  for (size_t i = 0; i < vect.size(); i++) {
        std::cout << vect[i];
        if (i < vect.size() - 1) {
            std::cout << " ";
        }
    }
    std::cout << std::endl;
}

double RandomVector::mean(){
  // TODO: Write your code here
   if (vect.empty()) return 0.0;

    double total = 0.0;
    for (size_t i = 0; i < vect.size(); i++) {
        total += vect[i];
    }
    return total / vect.size();
}
double RandomVector::max(){
  // TODO: Write your code here
   if (vect.empty()) return 0.0;

    double current_max = vect[0];
    for (size_t i = 1; i < vect.size(); i++) {
        if (vect[i] > current_max) {
            current_max = vect[i];
        }
    }
    return current_max;
}
double RandomVector::min(){
  //TODO:  Write your code here
   if (vect.empty()) return 0.0;

    double current_min = vect[0];
    for (size_t i = 1; i < vect.size(); i++) {
        if (vect[i] < current_min) {
            current_min = vect[i];
        }
    }
    return current_min;
}
void RandomVector::printHistogram(int bins){
  // TODO: Write your code here
    if (bins <= 0 || vect.empty()) return;

    double min_val = min();
    double max_val = max();
    double bin_width = (max_val - min_val) / bins;

    // Count values in each bin
    std::vector<int> counts(bins, 0);
    for (size_t i = 0; i < vect.size(); i++) {
        int bin_index = static_cast<int>((vect[i] - min_val) / bin_width);
        if (bin_index == bins) bin_index = bins - 1;
        counts[bin_index]++;
    }

    int max_count = 0;
    for (int i = 0; i < bins; i++) {
        if (counts[i] > max_count) {
            max_count = counts[i];
        }
    }

    for (int row = max_count; row > 0; row--) {
        for (int col = 0; col < bins; col++) {
            if (counts[col] >= row) {
                std::cout << "***";
            } else {
                std::cout << "   ";
            }
            std::cout << " ";
        }
        std::cout << std::endl;
    }
}
```

Below is a detailed explanation of my code.

```cpp
#include "random_vector.h"
// TODO: add any include you might require
#include <cstdlib>
#include <iostream>
#include <cmath>
```

#include "random_vector.h": This line includes the custom header file, which should declare the structure of the RandomVector class (member variables, method declarations, etc.). For example, it may include a vector of double-precision floating-point numbers: std::vector<double> vect; 。

#include <cstdlib>: Provides the random number generation functions std::rand() and the macro RAND_MAX, which are used to generate random numbers.

#include <iostream>: Offers standard input and output functionality, which is used in the print() method to print the contents of the vector.

#include <cmath>: Although the current code does not directly use mathematical functions, it might be a dependency reserved for future extensions (such as calculating the standard deviation, variance, etc.).

```cpp
RandomVector::RandomVector(int size, double max_val) {
  // TODO: Write your code here
  vect.resize(size);
  for (int i = 0; i < size; i++) {
      vect[i] = (static_cast<double>(std::rand()) / RAND_MAX) * max_val;
  }
}
```

When creating a RandomVector object, an array of length size (20) with random numbers in the range [0, max_val) is initialized. std::rand() generates integers in the range [0, RAND_MAX], which is converted to a double precision floating-point number using static_cast<double>, then divided by RAND_MAX to obtain a decimal number in the range [0, 1), and then multiplied by max_val to obtain a random number in the range [0, max_val). Finally, this number is stored in vect[i]. This effectively prevents the generated numbers from being too concentrated, so normalization processing is performed on all data to make the distribution as dispersed and uniform as possible.

```cpp
void RandomVector::print(){
  // TODO: Write your code here
  for (size_t i = 0; i < vect.size(); i++) {
        std::cout << vect[i];
        if (i < vect.size() - 1) {
            std::cout << " ";
        }
    }
    std::cout << std::endl;
}
```

Print all elements in the vector, with each element separated by a space, and end the line with a new character. `size_t` is an unsigned integer type, used to match the return type of `vect.size()` (ensuring that the loop variable does not overflow).After printing each element, except for the last one, a space is added, and a newline character `std::endl` is output at the end.

```cpp
double RandomVector::mean(){
  // TODO: Write your code here
   if (vect.empty()) return 0.0;

    double total = 0.0;
    for (size_t i = 0; i < vect.size(); i++) {
        total += vect[i];
    }
    return total / vect.size();
}
```

Calculate the arithmetic mean of all elements in the vector. If the vector is empty, return 0.0. Set a variable for calculating the overall sum, as the mean requires the total sum divided by the number of items. The final total is calculated by using a for loop to iterate through each element, and then adding each element to this variable. Finally, the totals are summed up to obtain the average.

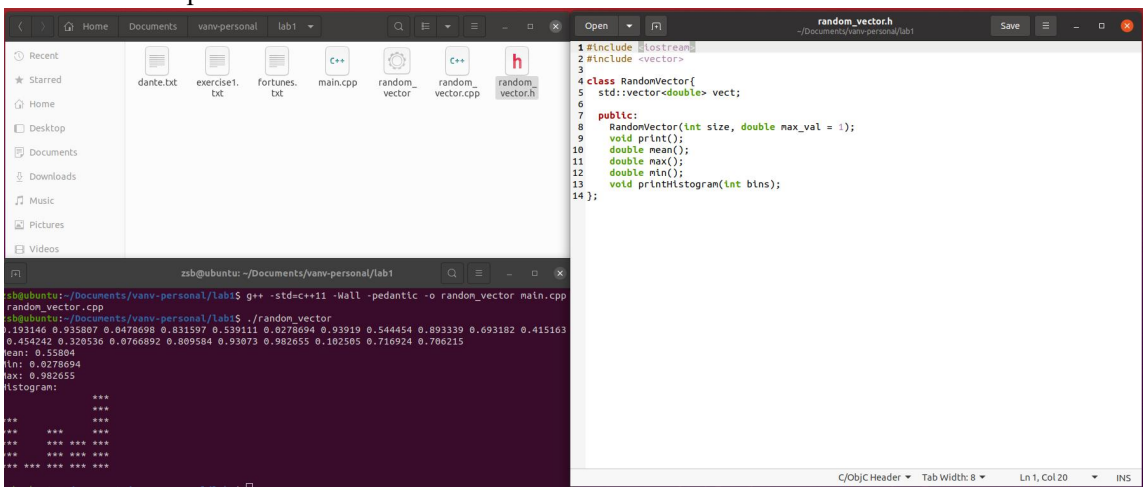
```cpp
double RandomVector::max(){
  // TODO: Write your code here
   if (vect.empty()) return 0.0;

    double current_max = vect[0];
    for (size_t i = 1; i < vect.size(); i++) {
        if (vect[i] > current_max) {
            current_max = vect[i];
        }
    }
    return current_max;
}
```

Find the maximum value in the vector. If the vector is empty, return 0.0. First, assume that the first element is the maximum value, current_max. Then, start traversing from the second element. If a larger element is encountered, update current_max; otherwise, do not update it. The final returned current_max will be the maximum value among all the numbers. The calculation for the minimum value is the same.

```cpp
double RandomVector::min(){
  //TODO:  Write your code here
   if (vect.empty()) return 0.0;

    double current_min = vect[0];
    for (size_t i = 1; i < vect.size(); i++) {
        if (vect[i] < current_min) {
            current_min = vect[i];
        }
    }
    return current_min;
}
void RandomVector::printHistogram(int bins){
  // TODO: Write your code here
    if (bins <= 0 || vect.empty()) return;

    double min_val = min();
    double max_val = max();
    double bin_width = (max_val - min_val) / bins;

    // Count values in each bin
    std::vector<int> counts(bins, 0);
    for (size_t i = 0; i < vect.size(); i++) {
        int bin_index = static_cast<int>((vect[i] - min_val) / bin_width);
        if (bin_index == bins) bin_index = bins - 1;
        counts[bin_index]++;
    }

    int max_count = 0;
    for (int i = 0; i < bins; i++) {
        if (counts[i] > max_count) {
            max_count = counts[i];
        }
    }

    for (int row = max_count; row > 0; row--) {
        for (int col = 0; col < bins; col++) {
            if (counts[col] >= row) {
                std::cout << "***";
            } else {
                std::cout << "   ";
            }
            std::cout << " ";
        }
        std::cout << std::endl;
    }
}
```

Key point：The calculation method for generating the distribution histogram is as follows: First, subtract the maximum value from the minimum value using the previous method, then divide the result by the number of intervals of the target. This will give the length of each interval. Since normalization processing was carried out in advance, this length is appropriate. Then, all the elements are traversed, and they are grouped into the corresponding intervals. For each interval, a variable used for counting is set up and used to perform the counting operation. From this, I obtained the widths of five intervals and the number of each interval. Finally, the plotting is carried out. The logic is as follows: for a two-dimensional image, two for loops are required, one to control the width and the other to control the height. First, use the largest number as the maximum height. Then, in the order from the top left to the top right, and from the top layer to the bottom layer, if the number is greater than or equal to the number of layers, output "***", otherwise generate "   ". After each layer finishes outputting, an endl is used and then it moves to the next layer.

Below is the output result.



3.Summary and Reflections

This Lab 1 serves as the core practical exercise for the introduction of computer professional skills. It is divided into three modules: Linux system operation, application of programming tools, and basic C++ development. This design forms a complete learning path from environment setup to function implementation. During the deployment process, I chose VMware Workstation Pro to set up a virtual environment for the experimental scenario. I downloaded the Ubuntu 20.04 image from the Tsinghua mirror source (https://mirrors.tuna.tsinghua.edu.cn/ubuntu-releases/focal/) and completed the installation. Subsequently, I switched to the administrator mode with `sudo -s`, backed up the original source list with `cp /etc/apt/sources.list /etc/apt/sources.list.bak`, and then used the `nano` editor to replace it with the Tsinghua source address and executed `apt update`, successfully resolving the problem of slow default source download speed and mastering the underlying logic of Linux system source configuration.

At the tool application level, after installing the development toolkit set via `sudo apt install build-essential cmake git`, I practiced the core operations of Git - cloning the pytorch/examples project and vnav-codes project from GitHub, understanding the logical connection between "remote repository" and "local repository" in version control; when using CMake to complete the build of the C++ project, by creating the `lab1_demo` directory, writing the `CMakeLists.txt` configuration file, executing `cmake ..` and `make` commands in the `build` directory, I clearly mastered the complete process of "cross-platform project build - compiling and generating executable files", solving the complex problem of dependency management in traditional compilation methods.

The programming practice section is the key part of this experiment. In the basic Shell command practice, I downloaded the `dante.txt` file in the `vnav-personal/lab1` directory using `wget`, then used the `wc` command combined with pipes (`|`) and the `awk` tool to complete the count of lines and words, filtered out non-empty lines using `grep -v '^$'`, and finally redirected the result to `exercise1.txt`, deepening the Shell programming thinking of "command combination - data processing - result output"; The C++ practice focused on the design and implementation of the `RandomVector` class, from the declaration of the class structure in `random_vector.h` (including the `std::vector<double> vect` member variable and member functions such as `mean()`, `max()`, `printHistogram()`), to the implementation of random vector generation (using `std::rand()` and normalization processing to generate data in the [0, max_val) interval), statistical analysis (iterating through the vector to calculate the mean, maximum value), and histogram drawing (dividing the data by intervals, and printing `***` frequency counts layer by layer), throughout avoiding the `<algorithm>` library to enhance the implementation of the underlying logic, and finally ensuring the code compliance through the `g++ -std=c++11 -Wall -pedantic` compilation command, successfully outputting vector statistics information and visualizing the histogram.

The entire experimental process covers the entire process of development environment setup, tool chain application, and programming language practice, not only achieving the experimental goal of "mastering Linux basic operations, programming tool usage, and C++ basics", but also solving multiple practical pain points (such as source configuration, project building, data statistics), laying a solid foundation for subsequent embedded development, large project development, and other complex technical learning.

As a computer science student, this experiment pushed me out of the comfort zone of "theoretical cognition" and allowed me to truly grasp the profound meaning of "engineering practice is the core of technology implementation". During the operation of the Linux system, initially, due to my unfamiliarity with the save and exit shortcut keys of the `nano` editor (saving with `Ctrl+O` and exiting with `Ctrl+X`), the source configuration failed. After repeated attempts, I realized that the core of Linux command-line operation is not only remembering the commands, but also understanding the underlying significance of each step (such as backing up the source list is to avoid the system being unable to recover after configuration errors). This kind of "knowing both the what and the why" thinking is precisely what distinguishes computer science students from ordinary users.

The process of learning tools enabled me to understand the "efficiency logic of engineering development". Previously, I manually compiled multiple C++ files, and the compilation failed due to incorrect dependency order. However, CMake automatically manages dependencies through instructions such as `add_library` and `target_link_libraries`, and Git avoids the problem of "difficulty in code overwriting and backtracking" through version control. The application of these tools made me realize that in actual development, "choosing the appropriate tool to solve a specific problem" is more important than "repeatedly writing basic code", and mastering tools mainly involves understanding their design concepts (such as Git's "distributed version control" and CMake's "cross-platform compatibility").

The practice of C++ programming is a profound training in combining "object-oriented thinking with underlying logic". When implementing the `printHistogram()` function of the `RandomVector` class, I initially caused the program to crash due to not considering the "data boundary issue" (such as when an element exactly equals the maximum value, the `bin_index` exceeds the range). Later, I solved this problem through the judgment statement `if (bin_index == bins) bin_index = bins - 1`. This made me realize that excellent code not only needs to achieve functionality but also needs to consider abnormal scenarios and robustness. When manually calculating the mean and maximum value, by traversing the vector instead of relying on the `<algorithm>` library, I more intuitively understood the underlying logic of "loop structure - data access - result calculation". This "leaving the library functions behind and returning to basic syntax" practice is crucial for consolidating the foundation of the C++ language.

Furthermore, the cultivation of "problem-solving ability" in the experiment has also benefited me a lot. For instance, when cloning a GitHub project due to network issues, the cloning was interrupted. I solved the problem by using `git clone --depth 1` for a shallow clone to reduce data transmission volume; when compiling the `RandomVector` class and encountering an error due to incompatible C++ standard versions, I adapted to the requirements by adding the `--std=c++11` compilation option. These experiences have taught me that in the field of computer technology, learning always involves the cycle of "encountering problems - analyzing causes - solving problems", and actively referring to official documentation (such as Git and CMake official tutorials) and searching for solutions on technical communities (such as CSDN and Stack Overflow) are core skills that professional students must possess.