浙江理工大學
ZHEJIANG SCI-TECH UNIVERSITY
School of Computer Science and Technology

Lab Report of Object-Oriented Programming A

Lab 2: Inheritance Credit hour: 3

Student Name:余洋 Student ID:
2023337621052

1 Objective

1.1 To master the principle of single inheritance and multiple inheritance;
1.2 To understand the difference among public, protected and private inheritance.
1.3 To master the definition of constructor in class hierarchy and the construction and destruction order when creating object.

2 Introduction to lab principle

Realize the extension to class function by using inheritance mechanism and design the appropriate member functions and constructors for the derived class.

3 Lab requirement

3.1 Software: C++ compiler under Windows or linux
3.2 Hardware: main memory(>2GB), free secondary memory(>40G), monitor and printer.

4 Lab content

Extend the function of lab 1. AdvancedElevator class is the derived class of Elevator class and it can realize that when several person waiting for the same elevator (up or down) in different floors, it can order the stop-floor according to the requirement of the person.

Requirement:

4.1 For the convenience of realization, we assume that a group of person waiting the elevator has the same direction, that is, up or down.
4.2 Testify the AdvancedElevator class in the main function. The method to testify is that firstly the direction(up or down) for some specific time and specific group of person should be determined, and then get the

number of the person of this group and the current floor and destination floor for each person in this group as input. During the running of the object created from the AdvancedElevator class, if the direction of the elevator is up(down), it can stop according to the current floor and destination floor of the passenger from bottom to top(from top to bottom)

4.3 When test this program, please be careful of tackling the problem that when several passengers in the same floor or several passengers' destinations are the same.

Tip:

In order to describe the passenger, we can define a Person class that is used to describe the current floor and destination floor of each person. AdvancedElevator class is derived from Elevator class and it obtain the current floor and destination floor from each person ready to take the elevator at some period, and then order the floor and display the stop-floor according to the order by using the member function setFloorNumber which derived from the base class Elevator.

Thought Questions (Optional):

If passenger's weight is taken into account, how to realize that the overload information can be displayed appropriately at some specific stop-floor.

5 Code list

、、、people.h

```
#include<iostream>
#include<string>
using namespace std;
class people
{
private:
    int start_floor;
    float weight;
    int target_floor;
    string name;
    int up_down=0;
    int in_out;
public:
    people(int total_floor);
    int read_target();
    string read_rame();
    int read_up_down();
```

```
24        int read_in_out();
25        int read_start_floor();
26        float read_weight();
27        void write_in_out(int in_out);
```

、、、elevator.h

```
#pragma once
#include <iostream>
#include<vector>
#include<string>
#include <iomanip>
#include<windows.h>
#include"people.h"
#include<queue>
#include<map>
using namespace std;

class Cdate
{
private:
    int hour;
    int minute;
    int second;
public:

    Cdate() {};
    void run();
    void desplay();
    void is_timeset();
    int change();


};
struct record {
public:
    vector<people> people_date;
    Cdate time;

};

class elevator {
```

```
private:
    float max_weight;
    float current_weight;
    int total_floor;
    int current_floor;
    vector<people> people_date;
    map<int,record> history;
    vector<people>post;
    int e_up_down = 0;
    Cdate time;
public:
    elevator();
    void people_in();
    void people_out();
    int check(people &A);
    void send_post();
    void start();
    void run_elevator(Cdate& time);
    void operate();

    };
    ----------------------------------------
、、、elevator.cpp
#include <algorithm>
#include"elevator.h"
#include"people.h"
int f(int current_floor, int start_floor)
{
    if (current_floor > start_floor) return -1;
    return 1;
}
elevator::elevator()
{
    cout << "welcome to this elevator,Starting initialization now。\nPlease enter the total floor and current floor" << endl;
    cout << "total_floor: ";
    cin >> total_floor;
    cout << "current_floor:";
    cin >> current_floor;
    cout << "the total weight" << endl;
    cin >> max_weight;
```

```
    cout << "you can press 0 to insert your operate" << endl;
};
void elevator::people_in()
{
    post.erase(remove_if(post.begin(), post.end(), [this]( people& p) {
        if (check(p) == 2) {
            people_date.push_back(p);
            current_weight += p.read_weight();
            cout << "-----------------------------------------------------------" <<
endl;
            cout << "people: " << p.read_rame() << "  get in the elevator" << endl;
            cout << "weight condition:  " << current_weight << "/" << max_weight << endl;
            cout << "-----------------------------------------------------------" <<
endl;
            return true;
        }
        else if (check(p) == 1) {
            cout << "people :" << p.read_rame() << "  is overweight,fail to go in the
elevator (the post will be saved until the elevator get this floor again)" << endl;
            cout << "weight condition:  " << current_weight << "/" << max_weight << endl;
            return true;
        }
        return false;
        }), post.end());
}
void elevator::send_post()
{
    cout << endl;
    cout << "input how many people will in the elevator,      current floor--->  " <<
current_floor << endl;
    int num;
    cin >> num;
    while (num--)
    {
        cout << endl << "----here to initialize one person's information----" << endl;
        people temp(total_floor);
        post.push_back(temp);
    }
}
void elevator::people_out() {
```

```
    if (people_date.size() != 0)
    {
        oo:
        int temp = 0;
            for(auto it = people_date.begin(); it != people_date.end(); ++it)
            {
                if (current_floor == (*it).read_target())
                {
                    cout <<endl<< "-----------------------------------------------"
<<endl;
                    cout << "floor :" << current_floor << "arrived" << endl << ",people "
<< (*it).read_rame() << " go  out  the elevator" << endl;
                    cout << "weight condition:  " << current_weight << "/" << max_weight
<< endl;
                    cout << "-----------------------------------------------" <<
endl<<endl;
                    current_weight -= (*it).read_weight();
                    people_date.erase(people_date.begin() + temp);
                    goto oo;

                }
                temp++;
        }
    }
}
int elevator::check(people &A)
{
    if (A.read_start_floor() == current_floor)
    {
        if (A.read_weight() + current_weight <= max_weight)
        {
            return 2;
        }
        else return 1;
    }
    else return 0;
}


void Cdate::is_timeset()
{
```

```
    {
        SYSTEMTIME current_time;
        GetLocalTime(&current_time);
        hour = current_time.wHour; minute = current_time.wMinute; second =
current_time.wSecond;
    }
}
void Cdate::run()
{
    {
        Sleep(1000);
        if (second > 59)
        {
            minute += 1;
            second = 0;
        }
        if (minute > 59)
        {
            hour += 1;
            minute = 0;
        }
        if (hour >= 24)
        {
            hour = 0;
        }
        desplay();
        second++;
    }
};
void Cdate::desplay()
{
    if (second < 10 && minute < 10) cout << hour << ":0" << minute << ":0" << second << endl;
    else if (minute < 10) cout << hour << ":0" << minute << ":" << second << endl;
    else if (second < 10) cout << hour << ":" << minute << ":0" << second << endl;
    else cout << hour << ":" << minute << ":" << second << endl;

};
void elevator::operate()
{
    cout << "please determinate your operate:" << endl << "1 . new one get in the elevator"
<< endl << "2.  check how many people in the elevator" << endl;
```

```
    int i;
    cin >> i;
    while (i != 1 && i != 2)
    {
        cout << "illegal operate ,please input again" << endl;
        cin >> i;
    }
    if (i == 1) send_post();
    else if (i == 2)
    {
        cout <<endl<< "The following are the persons in the elevator" << endl<<endl;
        if (people_date.size() == 0) cout << "the elevator is empty" << endl;
        else
        {
            for (auto it = people_date.begin(); it != people_date.end(); ++it)
            {
                cout << "name: " << (*it).read_rame() << "   " << "   weight: " <<
(*it).read_weight() << "   target floor : " << (*it).read_target() << endl;
            }
            cout << "weight condition:  " << current_weight << "/" << max_weight << endl;
        }
    }
}
void elevator::run_elevator(Cdate& time) {
    int max1 = -100, min1 = total_floor, max2 = -100, min2 = total_floor;
    if (e_up_down == 0 && (!post.empty())) e_up_down =
f(current_floor,post[0].read_start_floor());
    while (e_up_down != 0)
    {
        cout << "current_floor: " << current_floor << endl;
        people_in();
        people_out();
        if (total_floor  == current_floor) e_up_down = -e_up_down;
        if (current_floor == -1) e_up_down = -e_up_down;
        if (people_date.size() != 0)
        {
            for (auto it = people_date.begin(); it != people_date.end(); ++it)
            {
                if ((*it).read_target() < min1) min1 = (*it).read_target();
                if ((*it).read_target() > max1) max1 = (*it).read_target();
            }
```

```
        }
        for (auto it = post.begin(); it != post.end(); ++it)
        {
            if ((*it).read_start_floor() < min2) min2 = (*it).read_start_floor();
            if ((*it).read_start_floor() > max2) max2 = (*it).read_start_floor();
        }
        //cout << "max1=" << max1 << " min1=" << min1 << "max2= " << max2 << " min2= " << min2 << endl;

        if (!(max1 == -100 && min1 == total_floor))
        {
            if ((current_floor >= max1 && e_up_down == 1 && max2 <= current_floor) || (current_floor <= min1 && e_up_down == -1 && min2 >= current_floor)) e_up_down = -e_up_down;
        }
        current_floor += e_up_down;
        for (int i = 0; i < 5; i++)
        {
            time.run();
            if (GetAsyncKeyState('0') & 0x8000) operate();
        }

        if (people_date.size() == 0 && post.size() == 0)
        {
            e_up_down = 0; cout << "The elevator is empty and will be temporarily suspended on the floor ---->" << current_floor << endl;
        }
    }

};
void elevator::start()
{
    static Cdate time;
    time.is_timeset();
    send_post();
    while (true)
    {
        //system("CLS");
        run_elevator(time);
        if (GetAsyncKeyState('0') & 0x8000) operate();
    }
}
```

、、、people.cpp

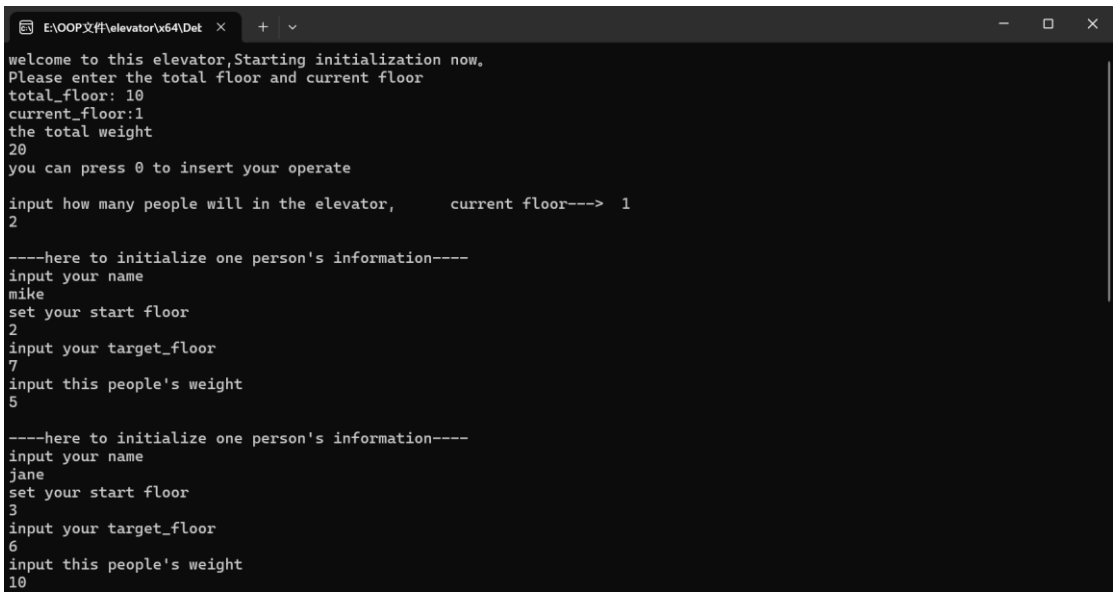
```
#pragma once
#include"people.h"
people::people(int total_floor)
{
    cout << "input your name" << endl;
    cin >> name;
    cout << "set your start floor" << endl;
    cin >> start_floor;
    while (start_floor > total_floor || start_floor <= 0)
    {
        cout << "illegal value,please input again" << endl;
        cin >> start_floor;
    }
    cout << "input your target_floor" << endl;
    cin >> target_floor;
    while (target_floor > total_floor || target_floor <= 0 || target_floor == start_floor)
    {
        cout << "illegal value,please input again" << endl;
        cout << "input your target_floor" << endl;
        cin >> target_floor;
    }
    if (target_floor > start_floor) up_down = 1;
    else up_down = -1;
    cout << "input this people's weight" << endl;
    cin >> weight;
}
string people::read_rame()
{
    return name;
}
int people::read_target()
{
    return target_floor;
}
int people::read_up_down()
{
    return up_down;
}
int people::read_in_out()
```

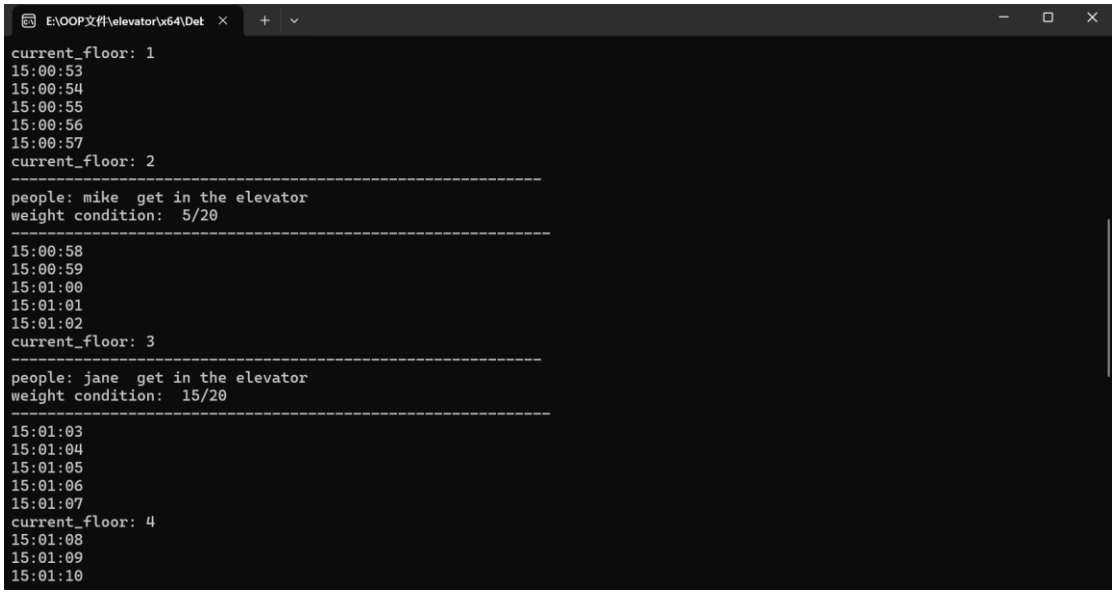
```
{
    return in_out;
}
void people::write_in_out(int in_out)
{
    this->in_out = in_out;
}
int people::read_start_floor()
{
    return start_floor;
}
float people::read_weight()
{
    return weight;
}
```

、、、main.cpp

```
#include "elevator.h"
int main()
{
    elevator A;
    A.start();
}
```

28 Output





29 Analysis and conclusions

This program mainly implements a simple elevator simulation system. The following is a list of program features:

This program mainly implements a simple elevator simulation system. The following is a list of program features:

1. Initialize elevator: Users can enter the total floor, current floor, and maximum load-bearing capacity.
2. Passengers entering the elevator: According to certain conditions, passengers can enter the elevator, including weight determination and target floor determination.
3. Passengers exiting the elevator: When the elevator reaches the target floor, passengers can exit from the elevator.
4. Send request: Users can input the number of people and information to enter the elevator, and add these people to the waiting queue.
5. Running the elevator: Simulate the operation process of the elevator, including passenger entry and exit, elevator up and down movement, floor judgment, etc.
6. Display current time: The elevator system will display the current time, updated once per second.
7. Operation selection: Users can choose to enter the elevator for new

personnel or view the current personnel information in the elevator.

8. Elevator operation control: Users can press the "0" key on the keyboard to select operations, including new personnel entering the elevator and viewing personnel information in the elevator.

9. Start running: After the program starts, initialize the time and send an initial request, and then enter the elevator operation cycle, continuously simulating the elevator operation process.