Lab Report of Object-Oriented Programming A

Lab 2: Inheritance Credit hour: 3

Student Name: 代翔 Student ID: 2023337621159

## ￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣

1. Objective
   1. To master the principle of single inheritance and multiple inheritance;
   2. To understand the difference among public, protected and private inheritance.
   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.

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

* 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.
  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)
  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.

1. Code list

//CDate.h

#pragma once

#include <iostream>

#include<windows.h>

using namespace std;

class CDate {

friend class Advance\_Elavator;

public:

void show();

private:

std::string timeString;

};

//elevator.h

#pragma once

class elevator

{

public:

int curfloor;

int choose;

int targetfloor;

public:

elevator();

void up\_display();

void down\_display();

void slection();

void running();

};

//Advnace\_Elevator.h

#pragma once

#include"elevator.h"

#include<iostream>

using namespace std;

#include<vector>

#include<set>

#include"Cdate.h"

class person

{

public:

int cur\_floor;

int target\_floor;

public:

person(int curfloor, int targetfloor);

};

class Advance\_Elevator :public elevator

{

public:

vector<int>\* target = new vector<int>;

vector<int>\* cur = new vector<int>;

set<int>\* target\_set = new set<int>;

set<int>\* cur\_set = new set<int>;

int cur\_floor;

int max\_people;

Int cur\_people;

public:

Advance\_Elevator();

void store(person a);

void up\_display();

void down\_display();

void print();

bool up\_or\_down();

void running();

};

//CDate.cpp

#include"Cdate.h"

void CDate::show()

{

SYSTEMTIME time;

GetLocalTime(&time);

printf("%04d/%02d/%02d %02d:%02d:%02d\n", time.wYear, time.wMonth, time.wDay, time.wHour, time.wMinute, time.wSecond);

}

//elevator.cpp

#include<iostream>

#include"elevator.h"

#include <windows.h>

#include<vector>

#include<algorithm>

using namespace std;

vector<int> upline;

vector<int> downline;

elevator::elevator() : curfloor(1), targetfloor(1), choose(-1) {};

void elevator::up\_display()

{

Sleep(1000);

curfloor++;

cout << endl;

cout << "Current floor: " << curfloor << endl;

}

void elevator::down\_display()

{

Sleep(1000);

curfloor--;

cout << "Current floor: " << curfloor << endl;

}

void elevator::running()

{

sort(upline.begin(), upline.end());

sort(downline.rbegin(), downline.rend());

auto highest = upline.end() - 1;

auto lowest = downline.end() - 1;

int up\_index = 0;

int down\_index = 0;

while (curfloor < \*highest)

{

up\_display();

if (curfloor == upline[up\_index])

{

cout << "Arrived." << endl;

up\_index++;

}

}

cout << endl;

while (curfloor > \*lowest)

{

down\_display();

if (curfloor == downline[down\_index])

{

cout << "Arrived." << endl;

up\_index++;

}

}

cout << "Arrived." << endl;

}

void elevator::slection()

{

upline.clear();

downline.clear();

cout << "Please choose to set up or ser down." << endl;

while (choose != 3)

{

cout << "1:up" << endl << "2:down" << endl << "3:input over" << endl;

cout << "please input your choose:";

cin >> choose;

if (choose == 1)

{

cout << "please input the floor:";

cin >> targetfloor;

cout << endl;

if (targetfloor < curfloor)

{

cout << "Illegal input!" << endl << endl;

}

else

{

upline.push\_back(targetfloor);

}

}

if (choose == 2)

{

cout << "please input the floor:";

cin >> targetfloor;

cout << endl;

if (targetfloor > curfloor)

{

cout << "Illegal input!" << endl << endl;

}

else

{

downline.push\_back(targetfloor);

}

}

if (choose > 3)

{

cout << "Illegal input!" << endl << endl;

}

}

choose = -1;

running();

}

//Advance\_Elevator

#include"Advance\_Elevator.h"

person::person(int curfloor, int targetfloor) :cur\_floor(curfloor), target\_floor(targetfloor) {};

Advance\_Elevator::Advance\_Elevator() :cur\_floor(1) ,max\_people(3),cur\_people(0){};

void Advance\_Elevator::store(person a)

{

target->push\_back(a.target\_floor);

if (target\_set->find(a.target\_floor) == target\_set->end())

{

target\_set->emplace(a.target\_floor);

}

cur->push\_back(a.cur\_floor);

if (cur\_set->find(a.cur\_floor) == cur\_set->end())

{

cur\_set->emplace(a.cur\_floor);

}

}

void Advance\_Elevator::up\_display()

{

Sleep(1000);

cur\_floor++;

cout << "Current floor: " << cur\_floor << endl;

}

void Advance\_Elevator::down\_display()

{

Sleep(1000);

cur\_floor--;

cout << "Current floor: " << cur\_floor << endl;

}

void Advance\_Elevator::print()

{

cout << "The number of people: " << target->size() << endl;

}

bool Advance\_Elevator::up\_or\_down()

{

int flag1 = \*target\_set->begin();

int flag2 = \*cur\_set->begin();

if (flag1 > flag2)

{

return true;//往上

}

else return false;//往下

}

void Advance\_Elevator::running()

{

print();

set<int> stop\_set;

for (auto it : \*target\_set)

{

if (stop\_set.find(it) == stop\_set.end())

{

stop\_set.emplace(it);

}

}

for (auto it : \*cur\_set)

{

if (stop\_set.find(it) == stop\_set.end())

{

stop\_set.emplace(it);

}

}

if (up\_or\_down())

{

for (auto it = stop\_set.begin(); it != stop\_set.end(); it++)

{

while (\*it > cur\_floor)

{

up\_display();

}

while (\*it < cur\_floor)

{

down\_display();

}

if (\*it == cur\_floor)

{

cout << "Arrived." << endl;

for (auto it1 : \*target)

{

if (it1 == \*it)

{

cur\_people--;

}

}

for (auto it1 : \*cur)

{

if (it1 == \*it)

{

cur\_people++;

}

}

if (cur\_people > max\_people)

{

cout << "Overweight" << endl;

}

}

}

}

else

{

for (auto it = stop\_set.rbegin(); it != stop\_set.rend(); it++)

{

while (\*it < cur\_floor)

{

down\_display();

}

while (\*it > cur\_floor)

{

up\_display();

}

if (\*it == cur\_floor)

{

cout << "Arrived." << endl;

for (auto it1 : \*target)

{

if (it1 == \*it)

{

cur\_people--;

}

}

for (auto it1 : \*cur)

{

if (it1 == \*it)

{

cur\_people++;

}

}

if (cur\_people > max\_people)

{

cout << "Overweight" << endl;

}

}

}

}

}

//mian.cpp

#include"elevator.h"

#include"Cdate.h"

#include"Advance\_Elevator.h"

int main()

{

CDate a;

a.show();

person p1(4, 7);

person p2(-3, 2);

person p3(-1, 4);

person p4(6, 9);

person p5(-2, 4);

person p6(-2, 5);

person p7(-2, 4);

Advance\_Elevator test;

test.store(p1);

test.store(p2);

test.store(p3);

test.store(p4);

test.store(p5);

test.store(p6);

test.store(p7);

test.running();

}

1. Output

2024/05/08 14:28:06

The number of people: 7

Current floor: 0

Current floor: -1

Current floor: -2

Current floor: -3

Arrived.

Current floor: -2

Arrived.

Overweight!

Current floor: -1

Arrived.

Overweight!

Current floor: 0

Current floor: 1

Current floor: 2

Arrived.

Overweight!

Current floor: 3

Current floor: 4

Arrived.

Current floor: 5

Arrived.

Current floor: 6

Arrived.

Current floor: 7

Arrived.

Current floor: 8

Current floor: 9

Arrived.

1. Analysis and conclusions

In this experimental course, I continued to use the CDate. h and its cpp from the previous experiment, as well as the elevator. h and its cpp. I inherited the data and functions from the elevator. h in Advanced-Elevator. h. However, by redefining the running function in the subclass, I covered the original functions in the base class. At the same time, I added the store function, print function, and up\_or\_down function to achieve more of the functions I needed.

In Advanced-Elevators.cpp, I set up max\_ppeople to limit the maximum number of people possible, and used cur\_ppeople to record the current number of people in the elevator. At the same time, four memory blocks were opened up. One block is called target, which is a vector container used to store the destination of elevator passengers. The other block is a vector container CUR used to store the location of elevator passengers, and each set container is used to automatically arrange these floor positions from small to large.

The function print records how many people are taking the elevator through a vector and prints it. The up\_or\_down function determines whether to go upstairs or downstairs based on the data in the set. In running, target set and cur\_set are integrated into stop\_set and stopped at the corresponding number of floors to determine if the elevator is overloaded.

Through this experiment, I learned about inheritance and other related knowledge, mastered the definition of constructors at the class level, as well as the principles of single inheritance and multiple inheritance.