**实验3：类与对象的进一步讨论**

**姓名\_\_李飞飞\_\_\_班级\_\_软工2206\_学号\_202105710309\_**

**1、对象的指针和数组、拷贝构造（30分）**

* **实验要求：**

**理解和掌握类类型的对象数组和对象指针的使用。理解拷贝构造函数应用的不同场景。**

**5分❶按照给出例程的文件命名各个部分并完成程序的装配。**

**ex3\_1 project : box.hpp+clock.hpp+demo.hpp+maintest.cpp**

**5分❷先阅读程序, 先写出你理解的程序运行结果。**

**再运行程序，验证你的结论。**

**结合程序实际执行理解输出结果，注意总结自己理解不到位的知识点并及时巩固。能够分辨拷贝构造函数应用的三种场景：声明、函数参数值传递、函数返回值传递。 可以使用注释部分代码的方法，帮助自己理解程序的工作结果。**

**5分❸对于Demo类型的对象，其数据成员x,y 的构造顺序是怎样的？（先获得x的空间还是先获得y的空间？）数据成员的这个顺序是由什么决定的？注意观察初始化列表的顺序，和它有关系吗？请给出你的结论，并想办法证实它。**

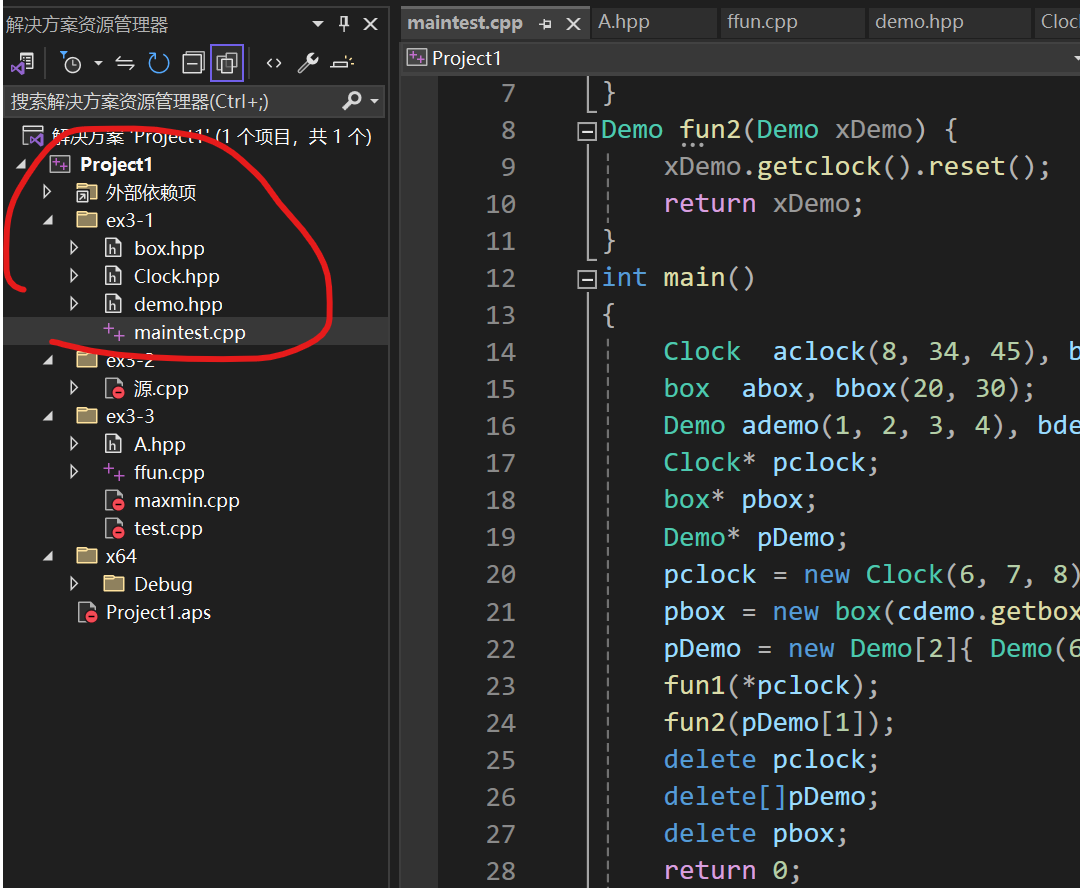
**5分❹给出程序中所有与 cdemo 这个对象相关的输出。**

**5分❺ 理解静态对象和动态对象构造析构顺序的差异。**

**5分❻ 给出和指针pDemo相关的输出。**

* **实验提交：**

**❶给出实验的项目装配截图 贴在此处**

****

**❷你模拟的程序输出结果：**

I'm a clock:8:34:45

I'm a copy clock:8:34:45

I'm a box:10-10-10

I'm a box:20-30-10

I'm a box:1-3-4

I'm a clock:2:3:4

Demo-Constructor1.

I'm a box:10-10-10

I'm a clock:1:1:1

Demo-Copy Constructor.

I'm a copy box:10-10:10

I'm a copy clock:8:34:45

Demo-Constructor2.

I'm a box:3-0-0

I'm a clock:4:0:0

Demo-Constructor1.

I'm a clock:6:7:8

I'm a copy box:10-10:10

I'm a box:6-8-9

I'm a clock:7:8:9

Demo-Constructor1.

I'm a copy box:20-30:10

I'm a copy clock:8:34:45

Demo-Constructor2.

I'm a copy clock:6:7:8

A clock says: Bye-bye.0:0:0

I'm a box:10-10-10

I'm a clock:1:1:1

Demo-Copy Constructor.

I'm a copy clock:1:1:1

A clock says: Bye-bye.0:0:0

I'm a box:10-10-10

I'm a clock:1:1:1

Demo-Copy Constructor.

Demo-Destructor.

A clock says: Bye-bye.1:1:1

A box says: Bye-bye.10-10-10

Demo-Destructor.

A clock says: Bye-bye.1:1:1

A box says: Bye-bye.10-10-10

A clock says: Bye-bye.6:7:8

Demo-Destructor.

A clock says: Bye-bye.8:34:45

A box says: Bye-bye.20-30-10

Demo-Destructor.

A clock says: Bye-bye.7:8:9

A box says: Bye-bye.6-8-9

A box says: Bye-bye.10-10-10

Demo-Destructor.

A clock says: Bye-bye.4:0:0

A box says: Bye-bye.3-0-0

Demo-Destructor.

A clock says: Bye-bye.8:34:45

A box says: Bye-bye.10-10-10

Demo-Destructor.

A clock says: Bye-bye.1:1:1

A box says: Bye-bye.10-10-10

Demo-Destructor.

A clock says: Bye-bye.2:3:4

A box says: Bye-bye.1-3-4

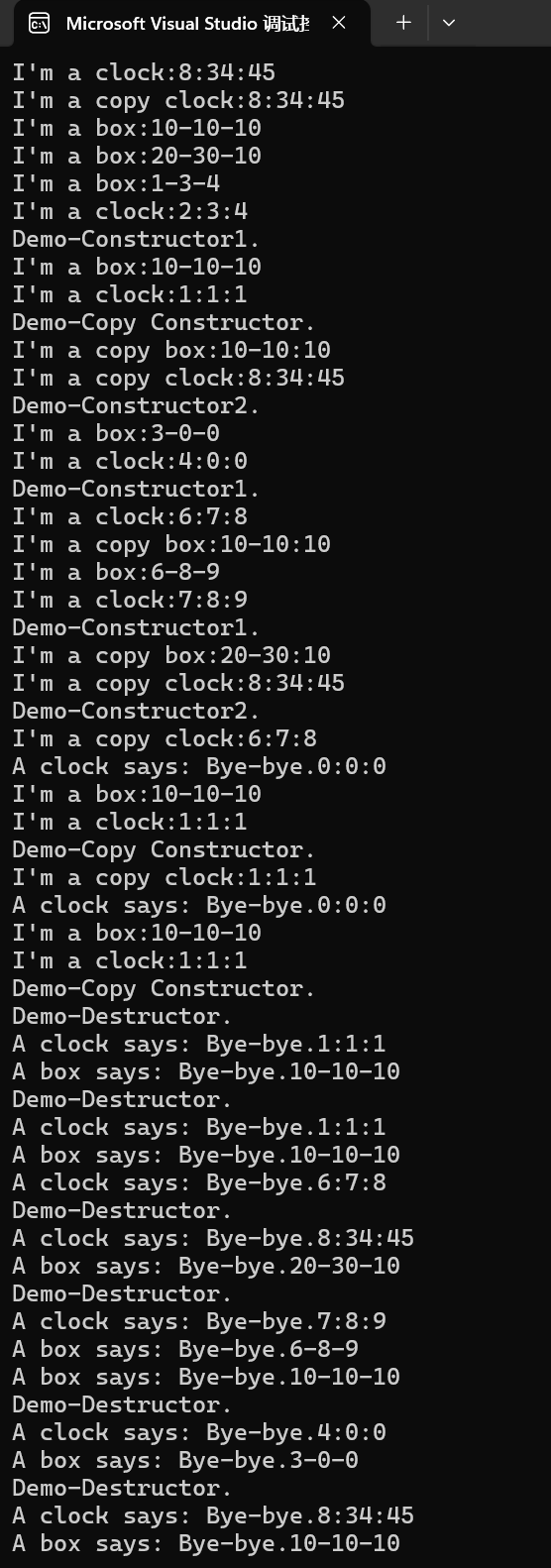
A box says: Bye-bye.20-30-10

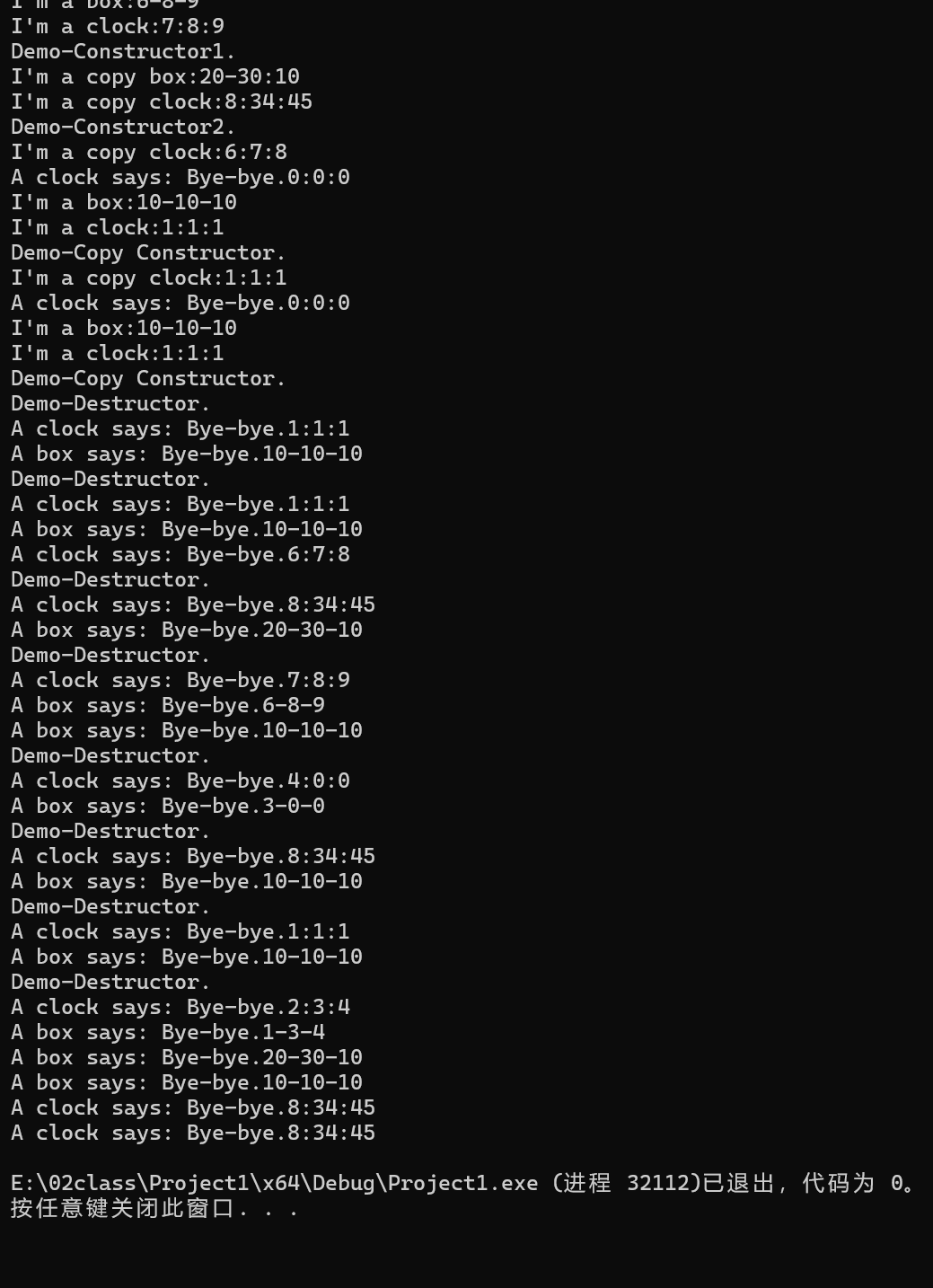
A box says: Bye-bye.10-10-10

A clock says: Bye-bye.8:34:45

A clock says: Bye-bye.8:34:45

**实际程序输出结果（可截图）：**

**、**

****

**知识总结：**

执行bdemo(ademo)时，调用的是拷贝构造函数Demo(const Demo& other);由于没有指定 box x;

Clock y;所以在初始化时这两个字段会调用自己的默认构造函数来初始化，分别为10，-10，10和1：1：1

fun1(\*pclock); fun2(pDemo[1]); 前面是把指针解引用传过去了，后面是直接传引用。由于两个函数是值传递，所以首先会调用拷贝构造来初始化函数参数。在fun2中还会先初始化参数的两个字段，即先构造clock和box,函数结束后自动调用析构函数销毁。Fun2中还调用了getclock,由于deom中的getclock是返回对象而不是引用，所以会再次调用拷贝构造函数来生成一个clock,这个clock又有没有被变量接收所以在执行rteset后会被销毁。Fun2返回一个由参数xDemo拷贝生成后的demo后结束函数，同时销毁参数xDemo，返回的结果由于没有被接收所以也立即销毁，至此两个函数在主程序中全部结束

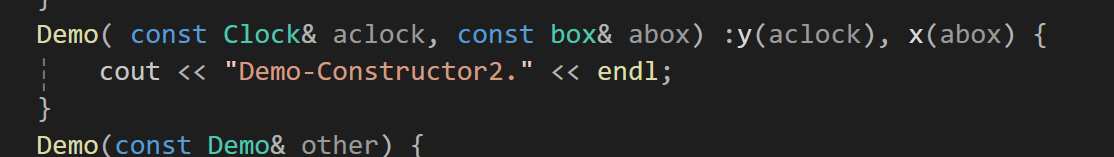
函数调用是复制了一个clock对象，这个对象在函数作用域结束后被释放，同理，但是Demo中

**❸答：**

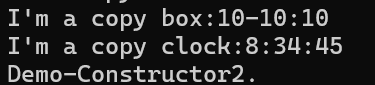
Demo类型的对象，其数据成员x,y 的构造顺序是先获得x的空间后获得y的空间。这个顺序由类申明顺序决定，而不是构造函数的参数顺序，也不是初始化列表的传参顺序。

证明：

1. 不由构造函数参数顺序决定：

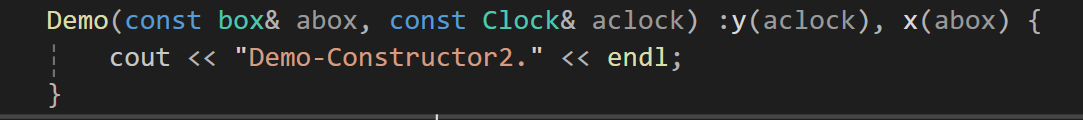
交换了顺序

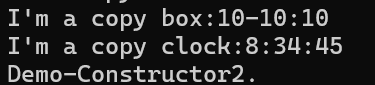
结果：



1. 不由初始化列表的传参顺序决定

交换初始化列表参数顺序：





**❹答：**

与cdemo(abox, aclock)相关：

I'm a copy box:10-10:10

I'm a copy clock:8:34:45

Demo-Constructor2.

与pbox = new box(cdemo.getbox());相关：

I'm a copy box:10-10:10

**❺答：**

静态对象的生命周期是整个程序的执行期间，而动态对象的生命周期则取决于何时进行内存分配和释放。

构造顺序：静态对象的构造顺序与声明顺序有关，而动态对象的构造顺序则完全取决于代码中调用new操作符的顺序。

析构顺序：静态对象的析构顺序是确定的，与构造顺序相反；而动态对象的析构顺序则取决于何时调用delete操作符。

**❻答：**

**最开始的声明没有输出**

与pDemo = new Demo[2]{ Demo(6,7,8,9),Demo(bbox,bclock) };有关：

I'm a box:6-8-9

I'm a clock:7:8:9

Demo-Constructor1.

I'm a copy box:20-30:10

I'm a copy clock:8:34:45

Demo-Constructor2.

与fun2(pDemo[1]);相关：

I'm a box:10-10-10

I'm a clock:1:1:1

Demo-Copy Constructor.

I'm a copy clock:1:1:1

A clock says: Bye-bye.0:0:0

I'm a box:10-10-10

I'm a clock:1:1:1

Demo-Copy Constructor.

Demo-Destructor.

A clock says: Bye-bye.1:1:1

A box says: Bye-bye.10-10-10

Demo-Destructor.

A clock says: Bye-bye.1:1:1

A box says: Bye-bye.10-10-10

与delete[]pDemo;有关：

Demo-Destructor.

A clock says: Bye-bye.8:34:45

A box says: Bye-bye.20-30-10

Demo-Destructor.

A clock says: Bye-bye.7:8:9

A box says: Bye-bye.6-8-9

**2、类的可缺省成员：Student（30分）**

**要求理解类的可缺省部分，以及缺省部分的适用情况。对于有额外申请动态空间的类要学会自己重写可缺省部分的操作。**

* **实验要求：**

**（15分）❶实现两种不同的student类(见ex3\_2\_student文件夹中的Student\_A,Student\_B), 并使用提供的主程进行测试。**

**（15分）**❷**思考总结：**类的四个可缺省成员原型、相应的缺省功能是什么？四个可缺省成员适用Student\_A类吗？四个可缺省成员适用Student\_B吗？总结适用类的特点，总结需要重写可缺省成员的类的特点。

* **实验提交：**

❶**给出完善后的Student\_A 和 Student\_B**

1. **Student\_A**

#include "Student1.hpp"

Student::Student() {

name[0] = '\0';

}

Student::Student(const char\* n){

int i;

for (i = 0; n[i] != '\0'; i++)

name[i] = n[i];

name[i] = '\0';

}

const char\* Student::GetName(){ //返回名字

return name;

}

void Student::ChangeName(const char\* n){

//改名

int i;

for (i = 0; n[i] != '\0'; i++)

name[i] = n[i];

name[i] = '\0';

}

****

1. **Student\_B**

#include"student2.hpp"

#include<iostream>

#include<cstring>

Student::Student(){

name = new char[1];

name[0] = '\0';

}

Student::Student(const char\* n){

int i = 0;

for (; n[i] != '\0'; i++){

}

name = new char[i + 1];

for (int j = 0; j < i; j++){

name[j] = n[j];

}

name[i] = '\0';

}

Student::Student(const Student& other){

int i = 0;

for (; other.name[i] != '\0'; i++){

}

name = new char[i + 1];

for (int j = 0; j < i; j++){

name[j] = other.name[j];

}

name[i] = '\0';

}

Student::~Student(){

delete[]name;

}

void Student::operator=(const Student& right){

int i = 0;

for (; right.name[i] != '\0'; i++){

}

name = new char[i + 1];

for (int j = 0; j < i; j++){

name[j] = right.name[j];

}

name[i] = '\0';

}

void Student::ChangeName(const char\* n){

delete[]name;

int i = 0;

for (; n[i] != '\0'; i++){

}

name = new char[i + 1];

for (int j = 0; j < i; j++){

name[j] = n[j];

}

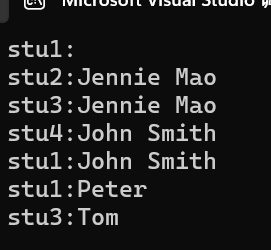
name[i] = '\0';

}

char\* Student::GetName(){

return name;

}

****

❷**思考总结回答：**

**四个可缺省成员的原型和功能：**

**Student();//不会执行任何操作，仅用于对象的初始化。**

**Student(const Student & other);// 会执行成员变量的浅拷贝**

**Student& operator=(const Student& right); //如果类中没有定义赋值操作符，编译器会生成一个浅拷贝赋值操作符**

**~Student( );// 不会执行任何特定的清理操作，用new声明的变量无法删除**

**3、static在类中的使用：静态数据成员和静态成员函数。(40分)**

* **实验要求：**

**1）（6分）运行文件夹“3-3 static”中的两个程序，体会static的不同作用。**

**答：staic修饰普通函数作用是：**

* + - **staic修饰普通函数作用是：将函数的作用范围限定于定义它的文件内**

**static修饰类的数据成员和成员函数的作用是：**

* + - **修饰类的数据成员属于类本身,所有对象实例共享同一个static数据成员。当static修饰类的成员函数时，该函数也是属于类本身，而不是类的对象实例，不能在static成员函数中访问类的非静态成员。**

**2）（14分）有基本的学生类型定义如下：**

**现要完成程序实现程序中学生的平均成绩的计算工作。可能的设计如下：**

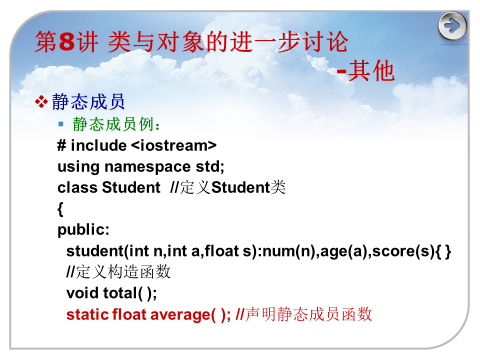
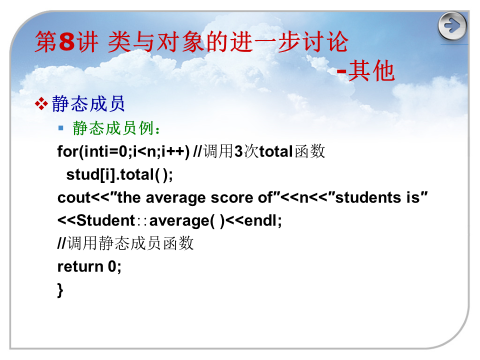
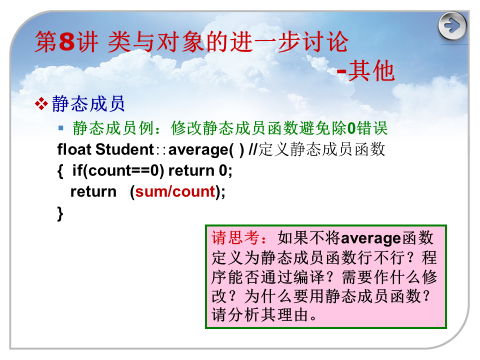
**❶定义全局变量来统计成绩总和和记录学生人数。设计函数来完成平均分的计算。**

**❷定义类内的静态数据成员来统计成绩总和和记录学生人数。（课堂例程）**

**装配并运行课程ppt上的代码，并思考：如果不将average函数定义为静态成员函数行不行？程序能否通过编译？需要作什么修改？为什么要用静态成员函数？请分析其理由。**

**❸思考总结：通过两种设计方案的比较，总结使用静态数据成员和成员函数来完成对应工作的优点？**

* **ppt附图：**

********

**❶ 答：按照第一种设计方案，基本的学生类需要做什么样的调整或者补充？请给出最终支持你实现的学生类和完成的主程。**

#include<iostream>

using namespace std;

float score\_sum = 0.0f;

int stu\_sum = 0;

class student {

public:

student(int n, int a, float s) :num(n), age(a), score(s)

{ //每创建一个对象加上总分和人数

score\_sum += s;

stu\_sum++;

}

private:

int num;

int age;

float score;

};

float AverageScore(){

if (stu\_sum == 0) return 0;

return score\_sum / stu\_sum;

}

int main(){

student s1(1, 20, 83.5), s2(2, 20, 90), s3(3, 20, 100);

cout << AverageScore() << endl;

}

**❷答：按照第二种设计方案。基本学生类需要做的调整或者补充可参考ppt。若你的设计实现与ppt不同，则给出对应的学生类。**

**思考：示范程序中如果不将average函数定义为静态成员函数行不行？程序能否通过编译？需要作什么修改？为什么要用静态成员函数？请分析其理由。**

**答：不将average函数定义为静态成员函数不行，程序不能通过编译，要将主程序中Student::average()的调用方式改成用单个Student对象来调用。**

**用静态成员函数可以通过类直接调用，不需要依赖具体的对象，静态成员函数只能访问静态数据成员，因为它们不与任何特定对象关联。**

**其它实现：**

//Student.hpp

class Student {

public:

Student(int n, int a, float s) :num(n), age(a), score(s) { }

void total();

static float average();

private:

int num;

int age;

float score;

static float sum;//总分

static int count;//人数

};

void Student::total(){

sum += score;

count++;

}

float Student::average(){

return (sum / count);

}

//Student.cpp

#include<iostream>

#include"Student.hpp"

using namespace std;

float Student::sum = 0;

int Student::count = 0;

int main(){

Student stud[3] = { Student(1001,18,70), Student(1002,19,78), Student(1003,18,98) };

for (int i = 0; i < 3; i++){

stud[i].total();

}

cout << "average = " << Student::average() << endl;

}

**3）位置类position再探索（20分）**

**实验2工作回顾：设计并实现一个平面坐标系内的位置类position。包含的基本数据成员有：横坐标，纵坐标；包含的基本成员函数有：设置位置set；读取位置show；判断第几象限quadrant；计算到原点的距离distance；计算到其他点的距离distance；计算经过原点到这个位置的直线的斜率slope；计算经过这个位置到其他点的直线的斜率slope；按坐标轴平移位置move。其他成员函数功能可以自行补充。**

**按照描述完成position类的基本的设计和实现。将数据成员设计为私有(private)成员；将成员函数设计为公有(public)成员。并通过给定测试程序。**

**现有新的工作需求：❶增加位置实例新的初始化可能；❷能完成平面坐标系内位置点密集度K的评估。密度K=平面坐标系中总的位置实例数/所有位置所在的区域面积。区域面积定义为由坐标系位置中最左、最右位置的x坐标垂直线，最上、最下位置的y坐标垂直线合围而成的矩形面积。（假设坐标系内总有最左、最右、最上、最下的位置存在。）**

**在1）2）实验工作积累的经验下，补充原有的position类的设计并实现它，使其可以完成以下主函数的测试工作。（提示：补充静态数据成员：left,right,up,down来记录位置实例的最左最右最上最下对应的坐标值；补充静态数据成员count来累计位置数；补充静态成员函数K()来完成位置点密度的计算。）**

#include "position.hpp

#include <iostream>

using namespace std;

int main() {

position a; //默认坐标位置为原点

position b(4,5),c(b),d(-2,-7),e(1); //b的位置为x轴值4，y轴值5

position \*px;

cout<<"K1 = "<<position::K()<<endl;

px=new position(-3.5, 9.8);

(\*px).show(); //在屏幕上输出 (-3.5,9.8)

cout<<"K2 = "<<a.K()<<endl;

px->set(99,-48); //重新设置点

cout<<"K3 = "<< position::K()<<endl;

delete px;

cout<<"K4 = "<< b.K()<<endl;

px=new position[3]{position(5,15),position(-4.5,16.7),position(-10.8)};

cout<<"b:"; b.show();

cout<<"c:"; c.show();

c.set(-10, -100);

cout<<"c:"; c.show();

e.set();//默认为原点

cout<<"e:"; e.show();

cout<<a.distance()<<endl;

cout<<c.distance()<<endl;//默认求与原点的距离

cout<<a.slope()<<endl; //与原点构成直线的斜率

cout<<a.slope(d)<<endl; //与d构成直线的斜率

a.move(3);//沿x轴平移

cout<<"a:"; a.show();

b.move(-4,5);

cout<<"b:"; b.show();

c.move(0,6.3);//沿y轴平移

cout<<"c:"; c.show();

cout<<"K5 = "<<position::K()<<endl;

delete []px;

cout<<"K6 = "<<position::K()<<endl;

return 0;

}

* **实验提交：**

**将完整的源代码和测试截图 粘贴在下面。**

* **源代码粘贴处：将Position类的声明和定义实现附在此处即可**

#pragma once

#include<cstring>

#include<iostream>

#include<cmath>

using namespace std;

class position {

public:

position(double a = 0, double b = 0) :x(a), y(b) {

count++;

update(\*this);

};

position(const position& other) {

this->x = other.x;

this->y = other.y;

count++;

update(\*this);

}

~position();

void set(double x = 0, double y = 0);

void show()const;

//判断第几象限

string quadrant()const;

double distance()const;

double distance(const position& other)const;

double slope()const;

double slope(const position& other)const;

void move(double distX, double distY = 0);

//补充的函数

static double K();

private:

static int count;

static double up, down, right, left;

double x;

double y;

static void update(position &now);

};

double distance(const position& other) {

return other.distance();

}

double distance(const position& a, const position& b) {

return a.distance(b);

}

//初始化

int position::count = 0;

double position::up = -1e9;

double position::down = 1e9;

double position::right = -1e9;

double position::left = 1e9;

void position::update(position& now) {

//cout << "JJJ\n";

up = max(up, now.y);

down = min(down,now.y);

left = min(left, now.x);

right = max(right, now.x);

}

double position::K() {

if (!count)return 0;

if (right < left || up < down)return 0;

double width = right - left;

double height = up - down;

if (width < 1e-5 || height < 1e-5)return 0;

return count / (width \* height);

}

position::~position() {

this->count--;

}

inline void position::set(double x, double y) {

this->x = x; this->y = y;

update(\*this);

}

inline void position::show()const {

cout << "横坐标：" << x << " 纵坐标：" << y << endl;

}

string position::quadrant()const {

if (x == 0 && y == 0)return "原点";

if (x == 0)return "y轴上";

if (y == 0)return "x轴上";

if (x > 0 && y > 0) return "一象限";

if (x < 0 && y < 0) return "三象限";

if (x > 0 && y < 0) return "四象限";

if (x < 0 && y > 0) return "二象限";

}

inline double position::distance()const {

return sqrt(x \* x + y \* y);

}

inline double position::distance(const position& other)const {

return sqrt((x - other.x) \* (x - other.x) + (y - other.y) \* (y - other.y));

}

inline double position::slope()const {

return y / x;

}

inline double position::slope(const position& other)const {

return (other.y - y) / (other.x - x);

}

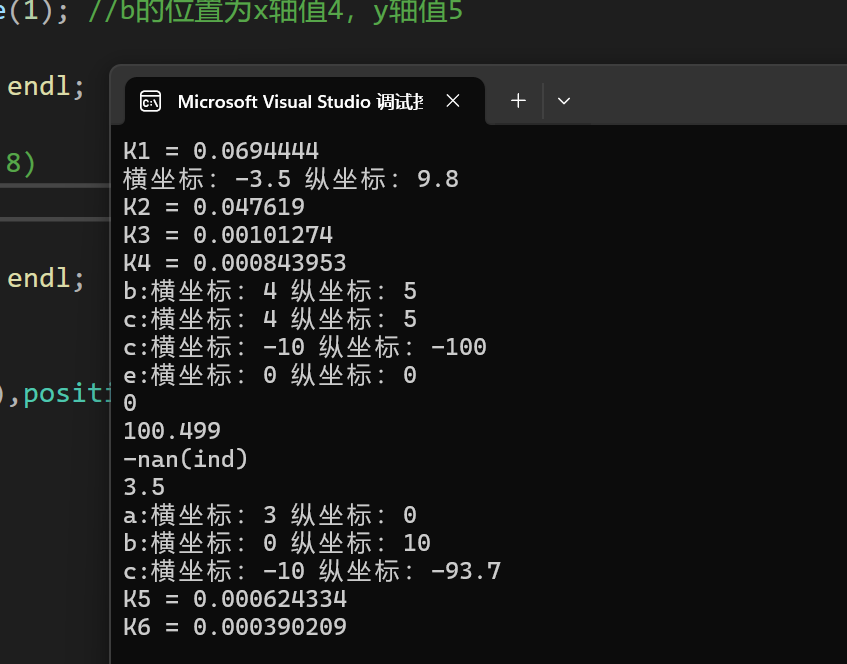
inline void position::move(double distX, double distY) {

x += distX; y += distY;

update(\*this);

}

* **程序测试截图：**

****