

## 实验 3：类与对象的进一步讨论（1）

姓名：陈王子

班级：大数据分析 2101

学号：202103150503

➤ 上传包结构为：

压缩包命名为：实验 3 学号+姓名

包含 3 个子文件夹（分别命名为：ex3-1, ex3-2, ex3-3） + 1 个实验文档；

每个子文件夹中包含相关对应实验题的源代码

//为防止与系统名字冲突，可以将 3-1 distance 和 3-2 clock 首字母大写

### 1、位置类 position: (30 分)

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；读取位置；判断第几象限；计算到源点的距离；计算到其他点的距离；计算经过源点到这个位置的直线的斜率；计算经过这个位置到其他点的直线的斜率；按坐标轴平移位置。其他成员函数功能可以自行补充。

#### ● 实验要求：

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。并通过以下测试程序。

#### ● 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

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

class Position {
private:
    double x; // 横坐标
    double y; // 纵坐标

public:
    // 默认构造函数，将坐标设置为原点
    Position() {
        x = 0;
        y = 0;
    }
}
```

```

// 构造函数，根据给定的横、纵坐标进行初始化
Position(double x_axis, double y_axis=0) {
    x = x_axis;
    y = y_axis;
}

// 复制构造函数
Position(const Position& p) {
    x = p.x;
    y = p.y;
}

// 显示当前坐标值
void show() const {
    cout << "(" << x << ", " << y << ")" << endl;
}

// 设置坐标值
void set(double x_axis, double y_axis) {
    x = x_axis;
    y = y_axis;
}

// 设置坐标值为原点
void set() {
    x = 0;
    y = 0;
}

friend double Distance(const Position& p1, const Position&
p2);
friend double Distance(const Position& p1);

// 计算当前点与另一个点之间的斜率
double slope(const Position& p) const {
    if (x == p.x) {
        cout << "Error: Cannot calculate the slope" << endl;
        return 0;
    }
    double slope = (y - p.y) / (x - p.x);
    return slope;
}

```

```

    // 计算当前点与原点之间的斜率
    double slope() const {
        Position origin(0,0);
        return slope(origin);
    }

    // 沿 x 轴方向平移指定的距离
    void move(double dist) {
        x += dist;
    }

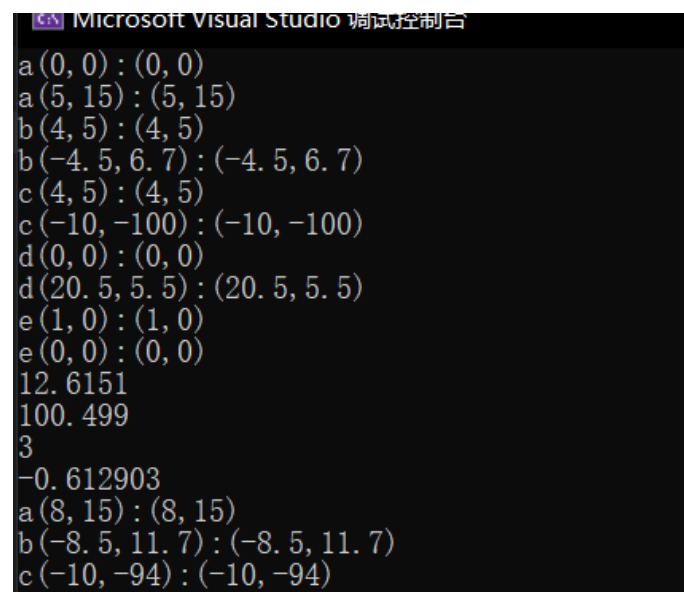
    // 沿 x 轴方向平移指定的距离，沿 y 轴方向平移指定的距离
    void move(double x_dist, double y_dist) {
        x += x_dist;
        y += y_dist;
    }
};

// 计算当前点与另一个点之间的距离
double Distance(const Position& p1, const Position& p2) {
    double dist = sqrt(pow(p2.x - p1.x, 2) + pow(p2.y - p1.y, 2));
    return dist;
}

// 计算当前点与原点之间的距离
double Distance(const Position& p1) {
    double dist = sqrt(pow(p1.x - 0, 2) + pow(p1.y - 0, 2));
    return dist;
}

```

■ 程序测试截图：



```

Microsoft Visual Studio 调试控制台
a(0,0):(0,0)
a(5,15):(5,15)
b(4,5):(4,5)
b(-4.5,6.7):(-4.5,6.7)
c(4,5):(4,5)
c(-10,-100):(-10,-100)
d(0,0):(0,0)
d(20.5,5.5):(20.5,5.5)
e(1,0):(1,0)
e(0,0):(0,0)
12.6151
100.499
3
-0.612903
a(8,15):(8,15)
b(-8.5,11.7):(-8.5,11.7)
c(-10,-94):(-10,-94)

```

## 2、对象的指针和数组、拷贝构造（40 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①按照给出例程的文件命名各个部分并完成程序的装配。

ex3\_2 project : box.hpp+clock.hpp+demo.hpp+maintest.cpp

5 分②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。

10 分③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④给出程序中所有与 cdemo 这个对象相关的输出。

5 分⑤理解静态对象和动态对象构造析构顺序的差异。

10 分⑥给出和指针 pDemo 相关的输出。

### ● 实验提交：

①给出实验的项目装配截图 贴在此处

名称	修改日期	类型	大小
box.hpp	2023/4/14 14:48	C++ Header 源...	1 KB
clock.hpp	2023/4/14 14:48	C++ Header 源...	1 KB
demo.hpp	2023/4/14 14:49	C++ Header 源...	1 KB
maintest.cpp	2023/4/14 14:49	C++ 源文件	1 KB

②你模拟的程序输出结果：

实际程序输出结果（可截图）：

```
C:\Users\princ>C:\Users\princ\source\repos\8330\Debug
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.8:0:0
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
```

知识总结:

我理解的程序运行的结果:

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//传入参数个数少于设置参数个数则从前往后覆盖

I\'m a box:1-3-4//因为 demo 的数据包含类类型因此调用了构造函数

I\'m a clock:2:3:4

Demo-Constructor1.

I\'m a box:10-10-10//

I\'m a clock:1:1:1//因为构造 demo 类 bdemo 的时候其实用的是拷贝构造函数的缺省形式, 因此 clock 和 box 使用默认参数

Demo-Copy Constructor.

I\'m a copy box:10-10:10//abox 和 aclock 都分别调用了拷贝构造函数

I\'m a copy clock:8:34:45//按照数据成员的申明顺序来构造

Demo-Constructor2.

I\'m a box:3-0-0

I\'m a clock:4:0:0//因为调用了构造函数并且传入了部分参数

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 box:20-30-10//虽然构造的顺序是 clock->box

I\'m a clock:8:34:45//但是构造的顺序由数据成员的申明顺序决定

Demo-Constructor2.

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

//因为 pclock 指针解指针的对象数据为 6, 7, 8

//将一个对象作为函数的参数按值调用方式传递给另一个对象时会生成对象副本

本

//且此对象副本是按照传入的对象经过拷贝构造函数进行创建的

A clock says: Bye-bye.0:0:0//reset 使参数都变成 0 以后自动销毁

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

I\'m a clock:1:1:1//与上面的 clock 不同的是, clock 的拷贝构造函数有赋值语句, 而 demo 里面的拷贝构造函数(传入参数为 demo)是系统内自带的缺省的拷贝构造函数, 因此 box 和 clock 都按照本身无参的构造函数来处理。

Demo-Copy Constructor.//

I\'m a copy clock:1:1:1//因为 demo 类中 getclock 函数返回值为 clock 类(即 return y), 要调用 clock 的拷贝构造函数, 且 y 即为 xdemo 的 clock 数据

A clock says: Bye-bye.0:0:0//return 完成以后马上析构, 又前面已经 reset 过, 因此析构的时候数据均为 0

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

I'm a clock:1:1:1

Demo-Copy Constructor.//return 的时候又需要调用 demo 类的拷贝构造函数,系统内自带的缺省的拷贝构造函数,因此 box 和 clock 都按照本身无参的构造函数来处理。

Demo-Destructor.//Q1: ? 为什么不是先析构 clock 和 box 再输出 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

//这里两组析构分别对应 return 时候的拷贝构造的析构和传入实参时候生成的临时副本的析构

A clock says: Bye-bye.1:1:1//对应 pclock 指针 new 出来的对象

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

//这里两组 demo 的析构对应 pdemo 指针所对应的数组

//并且先 new 的 pdemo[0]再 new 的 pdemo[1], 因此先析构[1]再析构[0]

A box says: Bye-bye.10-10-10//析构掉 pbox

Demo-Destructor.

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

A box says: Bye-bye.3-0-0//析构掉 ddemo(3, 4)

Demo-Destructor.

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

A box says: Bye-bye.10-10-10//析构掉 cdemo(abox, aclock)

Demo-Destructor.

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

A box says: Bye-bye.10-10-10//析构掉 bdemo(ademo)

Demo-Destructor.

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

A box says: Bye-bye.1-3-4//析构掉 ademo(1, 2, 3, 4)

A box says: Bye-bye.20-30-10//析构掉 bbox

A box says: Bye-bye.10-10-10//析构掉 abox

A clock says: Bye-bye.8:34:45//析构掉 bclock

A clock says: Bye-bye.8:34:45//析构掉 aclock

③答:

Demo 类型的对象数据成员 x 和 y 的构造顺序是先获得 x 的空间,再获得 y 的空间。这个顺序与初始化列表的顺序有关,初始化列表中先给出的是 x, 后给出的是 y, 所以先构造 x 的空间, 再构造 y 的空间。

④答:

相关输出:

```
I'm a clock:1:1:1
I'm a box:10-10-10
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.
A clock says: Bye-bye.0:0:0
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.
```

```
box abox, bbox(20, 30); // 定义 abox 和 bbox 两个对象
Clock aclock(8, 34, 45), bclock(aclock); // 定义 aclock 和
bclock 两个对象
Demo cdemo(abox, aclock); // 用 abox 和 aclock 对象初始化 cdemo
对象, 此处调用 Demo 的第二个构造函数
```

⑤答:

静态对象的构造在程序运行之前就完成了, 而动态对象的构造是在程序运行时进行的。因此, 静态对象的构造顺序是由它们在程序中的声明顺序决定的, 而动态对象的构造顺序是由它们的创建顺序决定的。析构顺序则恰恰相反, 动态对象的析构顺序是由它们的销毁顺序决定的, 而静态对象的析构顺序则是和构造顺序相反的。

⑥答:

2. 与指针 pDemo 相关的输出如下:

```
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 box:20-30-10
I'm a clock:8:34:45
Demo-Constructor2.
I'm a copy clock:6:7:8
A clock says: Bye-bye.0:0:0
A box says: Bye-bye.10-10-10
A clock says: Bye-bye.1:1:1
A box says: Bye-bye.10-10-10
```

A clock says: Bye-bye.1:1:1  
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.7:8:9  
A box says: Bye-bye.6-8-9  
A box says: Bye-bye.10-10-10  
Demo-Destructor.

对应的代码为:

```
pDemo = new Demo[2]{ Demo(6,7,8,9),Demo(bbox,bclock) }; //  
动态创建一个长度为 2 的 Demo 类数组，并用两个对象初始化其中的元素  
fun2(pDemo[1]); // 调用 fun2 函数，参数为 pDemo 数组中的第二个元素  
delete[]pDemo; // 释放 pDemo 数组占用的内存
```

### 3、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。对于有派生数据的类要学会自己重写可缺省部分的操作。

- **实验要求:** 实现两种不同的 student 类(见 `ex3_3_student` 文件夹中的 `Student_A, Student_B`), 并使用提供的主程进行测试。

主程见: `StudentApp.cpp`, 请装配 `Student_A` 的项目并运行观察结果;

然后完成 `student2.cpp`, 同样装配项目并观察运行结果, 两者的结果是否一致?

**结果一致!**

输出结果应该为:

```
stu1:  
stu2:Jennie Mao  
stu3:Jennie Mao  
stu4:John Smith  
stu1:John Smith  
stu1:Peter  
stu3:Tom
```



```
udentApp.cpp
main()

Student stu1, stu2("Jennie Mao"), stu3(stu2), stu4("John Smith");
cout << "stu1:" << stu1.GetName() << endl;
cout << "stu2:" << stu2.GetName() << endl;
cout << "stu3:" << stu3.GetName() << endl;
cout << "stu4:" << stu4.GetName() << endl;

stu1 = stu4;

cout << "stu1:" << stu1.GetName() << endl;

stu1.ChangeName("Peter");
cout << "stu1:" << stu1.GetName() << endl;

stu3.ChangeName("Tom");
cout << "stu3:" << stu3.GetName() << endl;

return 0;
```

标准输入:

运行结果:

**标准输出:**

```
stu1:
stu2:Jennie Mao
stu3:Jennie Mao
stu4:John Smith
stu1:John Smith
stu1:Peter
stu3:Tom
```

## ● 实验提交:

### StudentA

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

//student1.h
class Student {
public:
    Student(); //考虑这里为什么不缺省无参构造?
    Student(char* n);
    //Student(const Student & other);
    //Student& operator=(const Student& right);
    //~Student( );
    char* GetName(); //获取名字
    void ChangeName(char* n); //改名
private:
    char name[20];
};

//student11.cpp
Student::Student() {
    name[0] = '\0';
}

Student::Student(char* n)
{
    int i;
    for (i = 0; n[i] != '\0'; i++)
        name[i] = n[i];
    name[i] = '\0';
}
```

```

char* Student::GetName()
{
    return name;
}

void Student::ChangeName(char* n)
{
    int i;
    for (i = 0; n[i] != '\0'; i++)
        name[i] = n[i];
    name[i] = '\0';
}

```

## StudentB

```

#include <iostream>
#include <cstring> //需要使用到 C++ 字符串库
using namespace std;

//student2.h
class Student {
public:
    Student(); //默认构造函数
    Student(char* n); //带参数构造函数
    Student(const Student& other); //复制构造函数
    void operator=(const Student& right); //重载赋值运算符
    ~Student(); //析构函数
    char* GetName(); //获取名字
    void ChangeName(char* n); //改名
private:
    char* name;
};

//student2.cpp
Student::Student() { //默认构造函数
    name = new char[1]; //在堆上为 name 分配一个字节的空间
    *name = '\0'; //将 name 的第一个字节设为空字符
}

Student::Student(char* n) { //带参数构造函数
    int length = strlen(n); //获取字符串 n 的长度

```

```

        name = new char[length + 1]; //在堆上为 name 分配长度为
length 加 1 的空间
        strcpy(name, n); //将字符串 n 复制到 name 中
    }

Student::Student(const Student& other) { //复制构造函数
    int length = strlen(other.name); //获取 other 的名字长度
    name = new char[length + 1]; //在堆上为 name 分配长度为
length 加 1 的空间
    strcpy(name, other.name); //将 other 的名字复制到 name 中
}

void Student::operator=(const Student& right) { //重载赋值运算符
    if (this != &right) { //判断是否自己给自己赋值
        delete[] name; //先将自己原来的 name 内存释放掉
        int length = strlen(right.name); //获取 right 的名字长
度
        name = new char[length + 1]; //在堆上为 name 分配长度为
length 加 1 的空间
        strcpy(name, right.name); //将 right 的名字复制到 name
中
    }
}

Student::~Student() { //析构函数
    delete[] name; //将 name 内存释放掉
}

char* Student::GetName() { //获取名字
    return name;
}

void Student::ChangeName(char* n) { //改名
    int length = strlen(n); //获取字符串 n 的长度
    delete[] name; //将原来的 name 内存释放掉
    name = new char[length + 1]; //在堆上为 name 分配长度为
length 加 1 的空间
    strcpy(name, n); //将字符串 n 复制到 name 中
}

```