|  |  |
| --- | --- |
| **编号:** | **第1章第1次** |

****

信息科学与工程学院实验报告

《面向对象程序设计》

**Object-Oriented Programming**

|  |  |
| --- | --- |
| 姓名： | 朱会琛 |
| 学号： | 202311000202 |
| 班级： | 计工本2302 |
| 教师: | 张庆科 |
| 时间： | 2024年11月23日 |

**《面向对象程序设计》实验报告**

**报告要求：**实验报告包含实验目的、实验内容、实验过程（详细操作流程）、实验结果（程序运行结果高清截图）、实验分析总结五个部分。报告中若涉及代码程序，请在附录部分提供完整程序源码及源码托管地址(基于Highlight软件导入源码)。报告撰写完毕后请将PDF格式版本上传到坚果云作业提交系统。

1. **实验目的**

* 理解对象的概念及本质
* 掌握对象数组、指针和引用
* 掌握对象内成员的访问方法
* 掌握对象作为函数参数的用法
* 掌握友元函数的使用方法

1. **实验内容**

#### 任务一: 类的多种构造函数的使用

设计并实现一个学生类，其中学生的信息包括：姓名（char \*name），学号（int id），年龄（int age）, 成绩（double score）。该类对外提供的接口功能包括：

* 构造函数：初始化学生类对象（无参构造，有参构造，拷贝构造）；
* 普通函数：获取学生信息，修改学生信息，输出学生信息；
* 析构函数：释放学生类对象内存空间。//注意堆空间的释放
* 在主函数中定义多个学生类的对象，并采用不同形式实现对类对象的初始化。//采用对象数组
* 阐述普通构造函数（传值初始化新对象）、拷贝构造函数（对象初始化对象）在初始化对象过程中的联系与区别。

**任务二**: 类和对象的综合设计实验

设计一个**矩形类Rectangle**, 该类包含2个私有数据成员变量: double a, b; 对外提供的接口包括:

* 初始化类矩形类对象, 释放对象空间占用(三构一析)；
* 输出矩形的边长(get函数)、修改矩形的边长(set函数)；
* 计算并输出矩形的周长length和面积area；
* 基于该矩形类探索分析其对象作为函数void ObjectFunc(Rectangle v, Rectangle\* p, Rectangle& r)参数时的用法。

void ObjectFunc(Rectangle v, Rectangle\* p, Rectangle& r)

{

尝试修改矩形对象v中的边长a,b的数值为10,20; 在主函数中输出修改后的边长数值，分析对象传参的过程。

尝试通过指针p修改指向对象的边长a,b的数值为10, 20;在主函数中输出修改后的边长数值，分析对象指针传参的过程。

尝试修改r引用的对象中的边长a,b的数值为10,20;在主函数输出修改后的边长数值，分析对象引用传参的过程。

}

**任务三**: 类、对象及友元函数的使用

设计一个二维空间下的坐标点类 Point，该类包含 2 个私有数据成员：横坐标x 和纵坐标 y，与该类有关的函数如下所示：

* 三构造函数: Point(); Point(double a, double b);Point(const Point& r);
* 一析构函数: ∼Point();
* 提取横纵坐标: double GetX();double GetY();
* 修改横纵坐标: void SetX(double ax);void SetY(double bx)；
* 友元函数: friend double GetLength(Point& A, Point& B));//计算两点距离
* 类外全局函数: void SwapAxis(double \*xa，double \*xb); //指针法交换坐标点对象的横纵坐标值
* 类外全局函数: void SwapAxis(double& ra，double& rb);//引用法交换坐标点对象的横纵坐标值

三、 实验说明

1. 报告请勿抄袭与被抄袭，抄袭双方报告成绩均记为0分。
2. 撰写报告内容图文并茂，注意文字分析阐述要详细具体.
3. 按照课程提供的模板撰写实验报告, 采用其他模板会影响课程实验成绩。
4. 报告完成后请将**PDF**版本(不是Word版本,也不是压缩文件)  提交到坚果云。
5. **实验过程**

任务一源码：

#**include** **<**iostream**>**

#**include** **<**cstring**>**

**using** namespace std**;**

class Student **{**

private**:**

char \*name;

int id**;**

int age**;**

double score**;**

public**:**

**//** 无参构造函数

Student**() :** name**(**nullptr**),** id**(0),** age**(0),** score**(0.0) {}**

**//** 有参构造函数

Student**(**const char \*name, int id, int age, double score) {

this-**>**name **=** new char**[strlen(**name**) +** **1];**

strcpy**(**this-**>**name**,** name**);**

this-**>**id **=** id**;**

this-**>**age **=** age**;**

this-**>**score **=** score**;**

**}**

**//** 拷贝构造函数

Student**(**const Student **&**other**) {**

this-**>**name **=** new char**[strlen(**other**.**name**) +** **1];**

strcpy**(**this-**>**name**,** other**.**name**);**

this-**>**id **=** other**.**id**;**

this-**>**age **=** other**.**age**;**

this-**>**score **=** other**.**score**;**

**}**

**//** 获取学生信息

void getInfo**()** const **{**

cout **<<** "姓名**:** " **<<** name **<<** "**,** 学号**:** " **<<** id **<<** "**,** 年龄**:** " **<<** age **<<** "**,** 成绩**:** " **<<** score **<<** endl**;**

**}**

**//** 修改学生信息

void setInfo**(**const char \*name, int id, int age, double score) {

**delete[]** this-**>**name**;**

this-**>**name **=** new char**[strlen(**name**) +** **1];**

strcpy**(**this-**>**name**,** name**);**

this-**>**id **=** id**;**

this-**>**age **=** age**;**

this-**>**score **=** score**;**

**}**

**//** 析构函数

~Student**() {**

**delete[]** name**;**

**}**

**};**

int main**() {**

**//** 定义对象数组并初始化

Student students**[3] = {**

Student**(**"Alice"**,** **101,** **20,** **88.5),**

Student**(**"Bob"**,** **102,** **21,** **91.0),**

Student**(**"Charlie"**,** **103,** **22,** **85.0)**

**};**

**//** 输出学生信息

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

students**[**i**].**getInfo**();**

**}**

**//** 使用拷贝构造函数

Student copyStudent **=** students**[0];**

cout **<<** "**\n**拷贝构造函数创建的学生对象：" **<<** endl**;**

copyStudent**.**getInfo**();**

return **0;**

**}**

任务二源码：

#**include** **<**iostream**>**

**using** namespace std**;**

class Rectangle **{**

private**:**

double a**,** b**;**

public**:**

**//** 默认构造函数

Rectangle**() :** a**(0),** b**(0) {}**

Rectangle**(**double width**,** double height**) :** a**(**width**),** b**(**height**) {}**

**//** 拷贝构造函数

Rectangle**(**const Rectangle**&** rect**) :** a**(**rect**.**a**),** b**(**rect**.**b**) {}**

~Rectangle**() {}**

double getA**()** const **{** return a**; }**

double getB**()** const **{** return b**; }**

void setA**(**double width**) {** a **=** width**; }**

void setB**(**double height**) {** b **=** height**; }**

double length**()** const **{** return **2** \* (a + b); }

double area**()** const **{** return a \* b; }

**};**

void ObjectFunc**(**Rectangle v**,** Rectangle\* p, Rectangle& r) {

v**.**setA**(10);**

v**.**setB**(20);**

cout **<<** "在 ObjectFunc 函数内部 **(**按值传递**):** a **=** " **<<** v**.**getA**() <<** "**,** b **=** " **<<** v**.**getB**() <<** endl**;**

p-**>**setA**(10);**

p-**>**setB**(20);**

cout **<<** "在 ObjectFunc 函数内部 **(**指针传递**):** a **=** " **<<** p-**>**getA**() <<** "**,** b **=** " **<<** p-**>**getB**() <<** endl**;**

r**.**setA**(10);**

r**.**setB**(20);**

cout **<<** "在 ObjectFunc 函数内部 **(**引用传递**):** a **=** " **<<** r**.**getA**() <<** "**,** b **=** " **<<** r**.**getB**() <<** endl**;**

**}**

int main**() {**

Rectangle rect1**(5,** **8);**

Rectangle rect2**(4,** **6);**

Rectangle rect3**(7,** **9);**

cout **<<** "调用 ObjectFunc 之前**:**" **<<** endl**;**

cout **<<** "rect1**:** a **=** " **<<** rect1**.**getA**() <<** "**,** b **=** " **<<** rect1**.**getB**() <<** endl**;**

cout **<<** "rect2**:** a **=** " **<<** rect2**.**getA**() <<** "**,** b **=** " **<<** rect2**.**getB**() <<** endl**;**

cout **<<** "rect3**:** a **=** " **<<** rect3**.**getA**() <<** "**,** b **=** " **<<** rect3**.**getB**() <<** endl**;**

ObjectFunc**(**rect1**, &**rect2**,** rect3**);**

cout **<<** "调用 ObjectFunc 之后**:**" **<<** endl**;**

cout **<<** "rect1**:** a **=** " **<<** rect1**.**getA**() <<** "**,** b **=** " **<<** rect1**.**getB**() <<** " **(**未改变，按值传递**)**" **<<** endl**;**

cout **<<** "rect2**:** a **=** " **<<** rect2**.**getA**() <<** "**,** b **=** " **<<** rect2**.**getB**() <<** " **(**已改变，指针传递**)**" **<<** endl**;**

cout **<<** "rect3**:** a **=** " **<<** rect3**.**getA**() <<** "**,** b **=** " **<<** rect3**.**getB**() <<** " **(**已改变，引用传递**)**" **<<** endl**;**

return **0;**

**}**

任务三源码：

#**include** **<**iostream**>**

#**include** **<**cmath**>**

**using** namespace std**;**

class Point **{**

private**:**

double x**,** y**;**

public**:**

Point**() :** x**(0),** y**(0) {}**

Point**(**double a**,** double b**) :** x**(**a**),** y**(**b**) {}**

Point**(**const Point**&** r**) :** x**(**r**.**x**),** y**(**r**.**y**) {}**

**//** 析构函数

~Point**() {}**

double GetX**()** const **{** return x**; }**

double GetY**()** const **{** return y**; }**

double**&** GetX**() {** return x**; }**

double**&** GetY**() {** return y**; }**

void SetX**(**double ax**) {** x **=** ax**; }**

void SetY**(**double ay**) {** y **=** ay**; }**

**//** 声明友元函数，计算两点之间的距离

friend double GetLength**(**Point**&** A**,** Point**&** B**);**

**};**

**//** 友元函数实现，计算两点之间的欧几里得距离

double GetLength**(**Point**&** A**,** Point**&** B**) {**

double dx **=** A**.**x **-** B**.**x**;**

double dy **=** A**.**y **-** B**.**y**;**

return **sqrt(**dx \* dx + dy \* dy);

**}**

void SwapAxis**(**double\* xa, double\* xb) {

double temp **=** \*xa;

\*xa = \*xb;

\*xb = temp;

**}**

void SwapAxis**(**double**&** ra**,** double**&** rb**) {**

double temp **=** ra**;**

ra **=** rb**;**

rb **=** temp**;**

**}**

int main**() {**

Point P1**(3,** **4);**

Point P2**(7,** **1);**

cout **<<** "初始状态**:**" **<<** endl**;**

cout **<<** "P1**: (**" **<<** P1**.**GetX**() <<** "**,** " **<<** P1**.**GetY**() <<** "**)**" **<<** endl**;**

cout **<<** "P2**: (**" **<<** P2**.**GetX**() <<** "**,** " **<<** P2**.**GetY**() <<** "**)**" **<<** endl**;**

**//** 计算两点之间的距离

double distance **=** GetLength**(**P1**,** P2**);**

cout **<<** "P1 和 P2 之间的距离**:** " **<<** distance **<<** endl**;**

SwapAxis**(&**P1**.**GetX**(), &**P1**.**GetY**());**

cout **<<** "指针法交换 P1 的横纵坐标后**: (**" **<<** P1**.**GetX**() <<** "**,** " **<<** P1**.**GetY**() <<** "**)**" **<<** endl**;**

SwapAxis**(**P2**.**GetX**(),** P2**.**GetY**());**

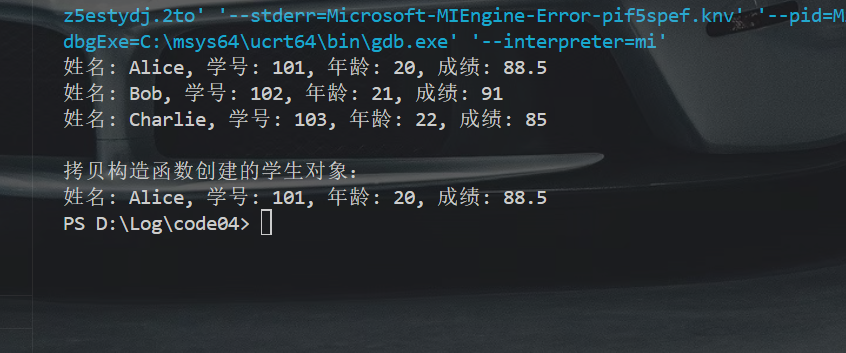
cout **<<** "引用法交换 P2 的横纵坐标后**: (**" **<<** P2**.**GetX**() <<** "**,** " **<<** P2**.**GetY**() <<** "**)**" **<<** endl**;**

return **0;**

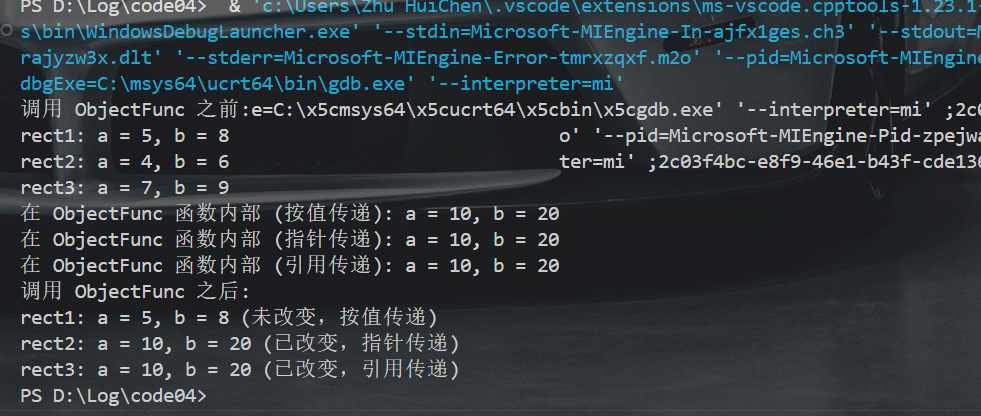
**}**

1. **实验结果**

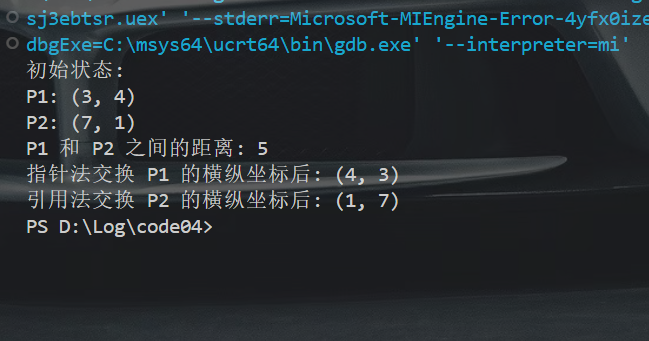
任务一运行结果：



任务二运行结果：



任务三运行结果：



1. **实验总结**

**答**：1.在本次实验中，通过实现拷贝构造函数，我们体会到它在对象复制中的重要作用。拷贝构造函数用于创建一个新对象，并用已有对象的数据初始化它，尤其在函数传参、返回值或赋值操作中至关重要。没有拷贝构造函数时，编译器会默认生成一个浅拷贝版本，但如果类中涉及动态分配内存或资源管理，就需要手动编写深拷贝版本。本实验虽然未用动态资源，但实现拷贝构造函数为后续复杂类的开发提供了基础，强化了对对象内存管理的理解。

2．我们通过设计类 Point 和实现友元函数、全局函数，深入理解了面向对象编程和 C++ 的函数调用机制。

首先，在实现 Point 类时，了解了构造函数和析构函数的重要性。三种构造函数分别用于对象的默认初始化、参数化初始化以及拷贝初始化。析构函数虽然在本实验中未涉及动态资源管理，但依然是类中不可忽略的一部分，为后续扩展做好准备。

其次，通过友元函数 GetLength 的实现，我们体会到友元的灵活性。友元函数可以直接访问类的私有成员变量，方便在类外实现功能，但同时也打破了封装性，需要合理使用，避免滥用。

全局函数 SwapAxis 的两种实现展示了指针和引用的应用场景。在指针法中，需要传递地址，容易出错，但在动态数组或需要手动管理内存时非常实用。而引用法更自然、直观，适合大多数日常使用。

实验过程中，错误提示 "lvalue required" 帮助我们意识到返回值类型对函数调用的影响。通过为 GetX 和 GetY 添加返回左值引用的重载，我们进一步理解了 lvalue 和 rvalue 的区别及其在函数调用中的作用。