学号 专业 信息安全 姓名

实验日期 **2024/06/05** 教师签字 成绩

实验报告

【实验名称】 多态性与抽象类

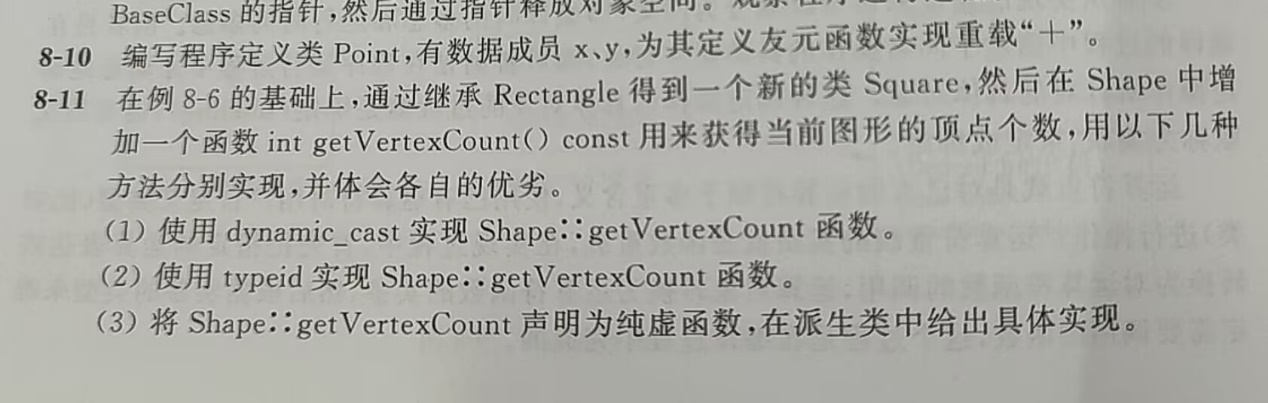
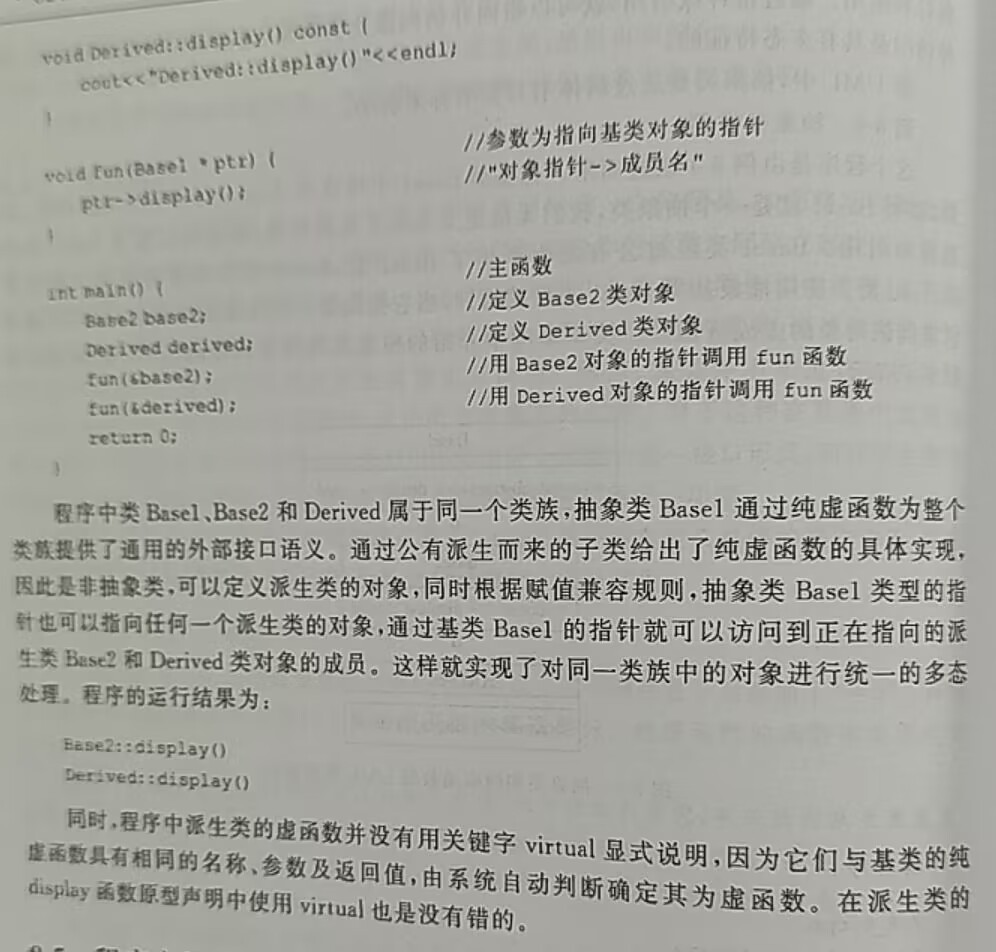
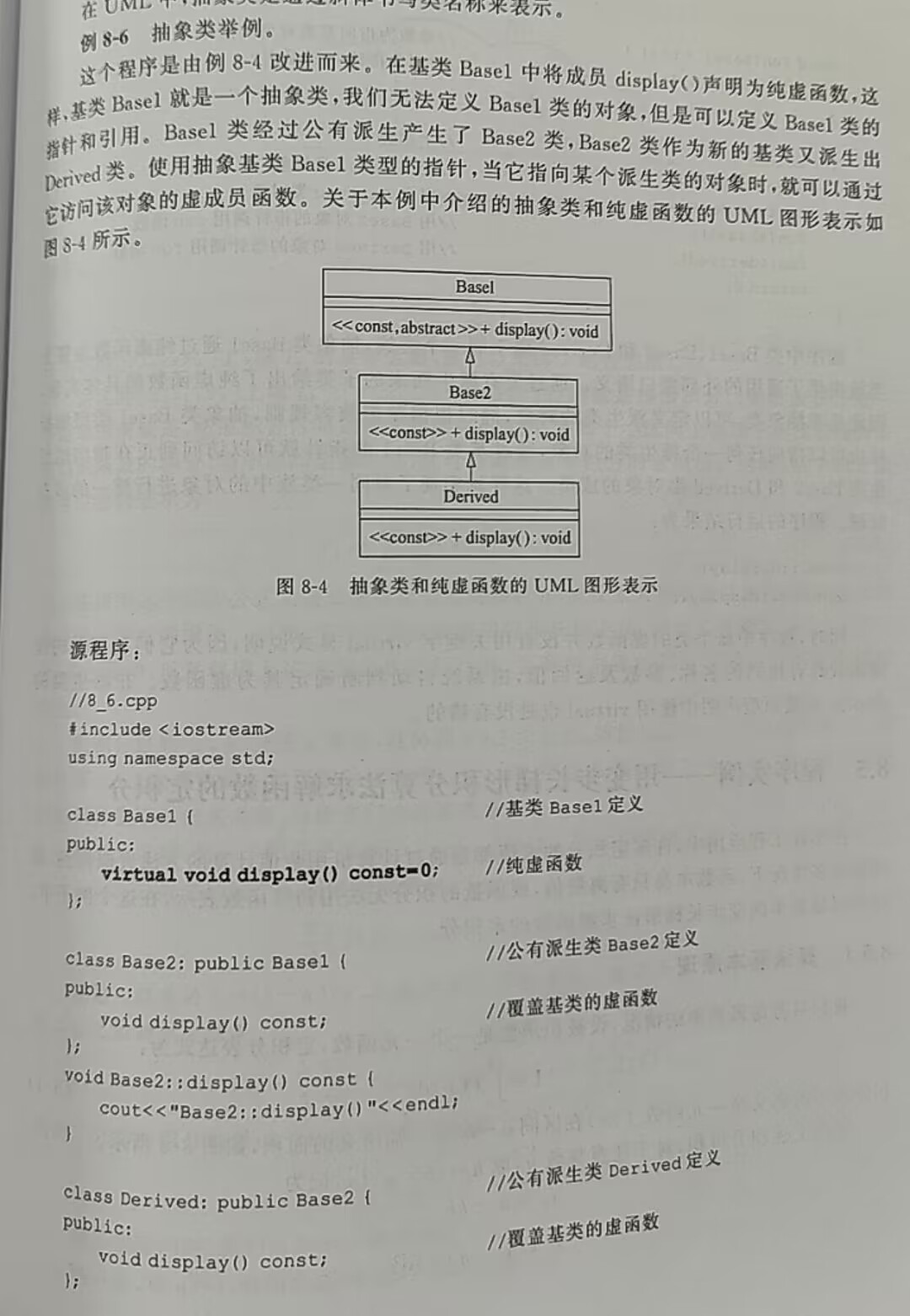
【实验目的】

**理解多态性**：通过使用基类指针或引用来调用派生类的重载函数，理解动态绑定和多态性在C++中的实现方式。

**掌握抽象类的概念**：学习如何定义和使用抽象类，理解纯虚函数的作用以及抽象类不能实例化的原因。

**实践面向对象设计原则**：通过设计和实现具有多态性的类层次结构，强化面向对象编程的基本概念和设计模式。

【题目】



8-10和8-11

【实验原理】

在实现c++多态时会用到虚函数。虚函数使用的其核心目的是通过基类访问派生类定义的函数。所谓虚函数就是在基类定义一个未实现的函数名，为了提高程序的可读性，建议后代中虚函数都加上virtual关键字。

8-10

思路是定义一个 `Point` 类，通过友元函数实现 `+` 运算符的重载，以便可以直接对两个 `Point` 对象进行相加操作。类包含私有成员变量 `x` 和 `y` 以及用于初始化这些变量的构造函数。友元函数 `operator+` 实现了将两个 `Point` 对象的对应坐标相加并返回一个新的 `Point` 对象。`main` 函数中创建了几个 `Point` 对象，演示了对象的初始化、显示和相加操作。最终，通过 `display` 成员函数输出相加后的结果。

8-11

1. 类的层次结构

程序定义了几个类，它们之间通过继承建立了层次结构：

* Shape类：作为基类，定义了纯虚函数getArea()、getPerim()和getVertexCount()，这些函数在派生类中被重写以计算不同形状（如矩形、圆、正方形）的特定属性（面积、周长、顶点个数）。
* Rectangle类、Circle类和Square类：这些类继承自Shape类，分别表示矩形、圆和正方形。它们实现了基类中定义的纯虚函数，并提供了具体的计算方法。此外，它们还包含了获取特定属性（如矩形的长和宽、圆的半径）的成员函数。

2. 多态性和虚函数的运用

* 虚函数：在Shape类中，getArea()、getPerim()和getVertexCount()函数被声明为纯虚函数，这意味着基类本身无法提供具体实现，需要在派生类中进行实现。这种设计通过虚函数实现了运行时多态性，允许通过基类指针或引用调用派生类对象的函数，根据对象的实际类型来调用相应的实现。
* 重写函数：在每个派生类中，重写了printDetails()函数，以便输出特定对象的详细信息（如矩形和正方形的边长、圆的半径）。这种方法利用了C++的动态绑定特性，确保在运行时调用正确的函数版本。

3. 运算符重载和友元函数

* 运算符重载：在Counter类中，重载了+运算符，使得两个Counter对象可以进行相加操作。通过友元函数的方式实现运算符重载，友元函数能够访问类的私有成员，从而提供了更高的灵活性。

4. 使用类型信息和动态类型转换

* typeid运算符：在getVertexCount2()函数中，使用了typeid运算符来比较对象的类型，以确定应该调用哪个派生类的成员函数。这种方法虽然比较笨拙，但展示了在缺少虚函数的情况下，如何根据对象的实际类型来调用相应的函数。

5. 设计理念总结

这个程序的设计理念主要包括：

* 继承和多态：通过继承和虚函数实现代码的复用和扩展性，允许在不同的派生类中提供不同的实现，从而符合面向对象编程的基本原则。
* 运算符重载和友元函数：运算符重载和友元函数的使用增强了代码的表达力和灵活性，使得类的接口设计更加自然和易于使用。
* 类型信息和动态类型转换：虽然推荐使用虚函数和多态来实现对象的动态行为，但typeid运算符和动态类型转换提供了在某些情况下的备选解决方案，展示了不同的设计选择和权衡。

**比较**：使用 dynamic\_cast：

* + 优点：可以在运行时确定对象的类型，灵活处理多个派生类。
  + 缺点：使用时需要检查转换是否成功，代码较为复杂，性能略有下降。

使用 typeid：

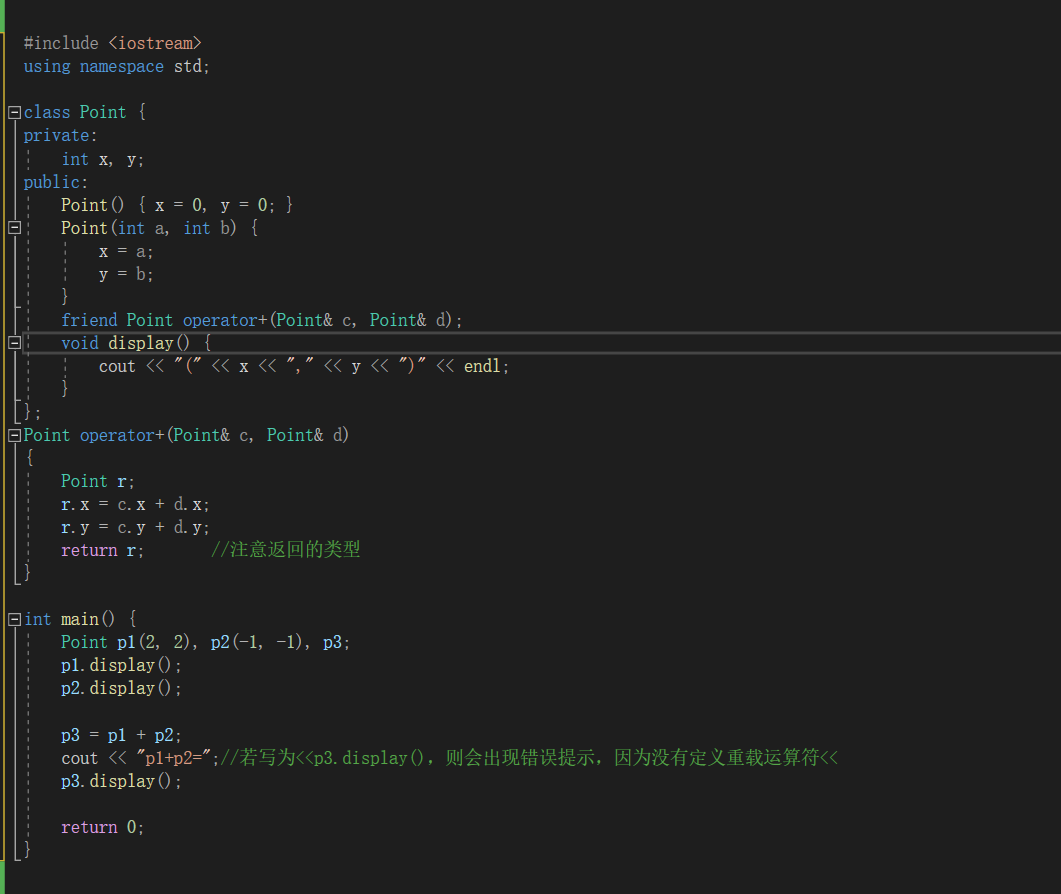
* + 优点：可以在运行时确定对象的类型，代码相对简单。
  + 缺点：需要引入 RTTI（运行时类型识别），会增加一些运行时开销，扩展性较差。

使用纯虚函数：

* + 优点：设计更加面向对象，每个派生类各自实现自己的逻辑，清晰明了，易于扩展。
  + 缺点：每个派生类需要实现该函数，增加代码量。

【源码】

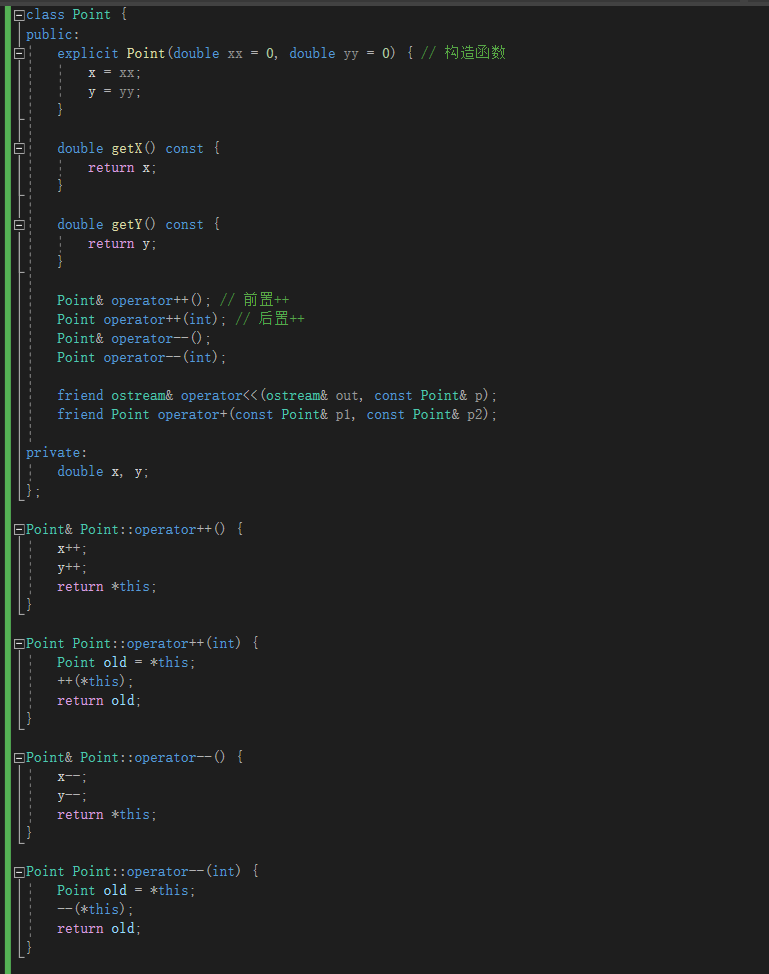
8-10

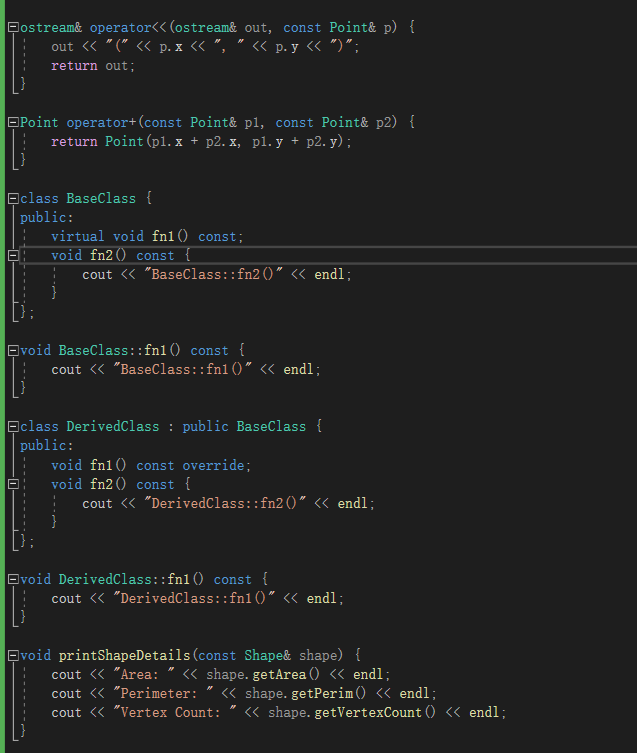


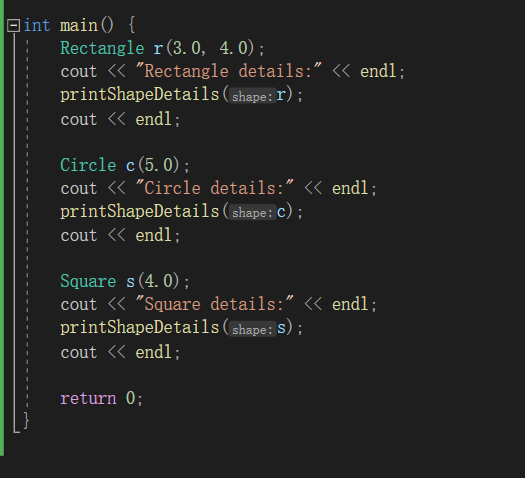
8-11



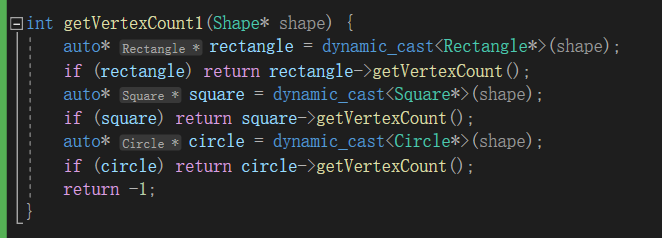




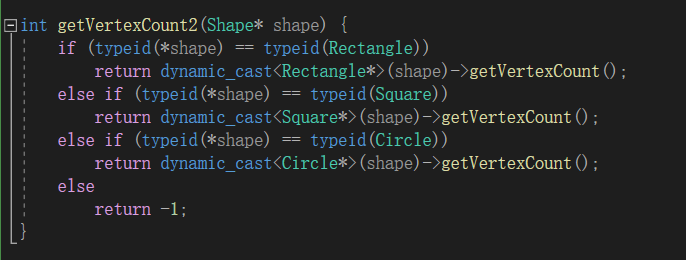




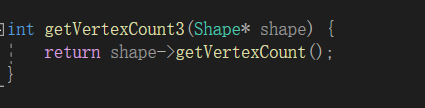
1. 使用dynamic\_cast 实现 Shape::getVertexCount函数。



(2)使用typeid 实现Shape::getVertexCount 函数。



(3)将 Shape::getVertexCount声明为纯虚函数，在派生类中给出具体实现。



【运行结果截图】

8-10



8-11



【实验总结】

本次实验涉及了C++中面向对象编程的基础知识，包括类的继承、多态性以及虚函数的运用。以下是对实验内容的总结和学习收获：

#### 1. 类的设计与继承

在本次实验中，我们定义了三个类：Shape（基类）、Rectangle（派生类）、Circle（派生类）和Square（派生类）。主要的设计思路包括：

* **Shape类**：作为抽象基类，定义了纯虚函数getArea()、getPerim()和getVertexCount()，并提供了虚析构函数。这些函数用于计算图形的面积、周长和顶点个数，子类需要实现这些函数。
* **Rectangle类和Circle类**：继承自Shape类，分别实现了具体的计算面积、周长和顶点个数的方法。此外，Rectangle类还提供了获取长和宽的方法，Circle类提供了获取半径的方法。
* **Square类**：继承自Rectangle类，重写了printDetails()方法来输出正方形的边长。

#### 2. 多态性和虚函数

本次实验中展示了多态性的使用：

* **虚函数**：在基类Shape中，定义了纯虚函数和虚析构函数，允许派生类重写这些函数以实现特定的功能。这样，可以通过基类指针或引用调用派生类的函数，实现了运行时的多态性。
* **实现多态的方法**：在printShapeDetails()函数中，通过Shape类的引用参数，调用printDetails()函数，实现了对各个派生类对象的特定信息输出。

#### 3. 运算符重载与友元函数

* **运算符重载**：在Counter类中，重载了+运算符，使得两个Counter对象可以进行相加操作。这展示了C++中运算符重载的基本用法。
* **友元函数**：在Counter类中，operator+函数被声明为Counter类的友元函数，可以访问类的私有成员变量。这在需要操作类的私有数据但不希望将其作为成员函数时很有用。

#### 4. 输出对象的信息

* **通过成员函数输出对象信息**：在每个派生类中，实现了printDetails()函数，用于输出特定的对象信息（如矩形和正方形的边长、圆的半径）。

总之，本次实验帮助我进一步巩固了C++面向对象编程的基础知识，并提高了我在设计和实现类层次结构时的技能和经验。这些知识对于日后编写复杂的软件系统和提高代码的可维护性至关重要。