

计算机学院高级语言程序设计课程实验报告

实验题目：实验七、多态性		学号：202300130183
日期：2024 年 3 月 4 日	班级：2023 级智能班	姓名：宋浩宇
Email：202300130183@mail.sdu.edu.cn		
<p>实验目的：</p> <ol style="list-style-type: none">1. 掌握运算符重载的方法。2. 学习使用虚函数实现动态多态性。		
<p>实验软件和硬件环境：</p> <p>实验软件：Windows 11 家庭中文版(x64) Visual Studio 2022</p> <p>硬件环境：处理器：13th Gen Intel(R) Core(TM) i9-13980HX 2.20 GHz RAM 32.0 GB (31.6 GB 可用)</p>		
<p>实验步骤与内容：</p> <p>第八章 1. (1)</p> <div><pre>_x=1 _y=1 _x=1 _y=1 _x=2.41421 _y=2.41421 _x=1.70711 _y=1.70711 _x=1.70711 _y=1.70711 F:\Homework\高程作业\实验7\code\x64\Debug\code.exe (进程 32512)已退出，代码为 0。 按任意键关闭此窗口。 . . .</pre></div>		
<p>结果解释为：初始的 p 的_x, _y 被初始化为 1，然后对 p 进行后置自增运算，此时 show() 的是_x, _y 自增之前的值即为 1，而经过后置自增之后此时_x 和_y 为 1.70711；在经过前置自增运算，此时_x, _y 等于 2.41421，此时 show() 的就是自增之后的_x, _y 的值；在对 p 进行前置自减，此时_x, _y 等于 1.70711，因此</p>		

show() 的就是自减之后的值；再对 p 进行后置自减运算，此时 show() 的是_x, _y 自减之前的值即为 1.70711，在经过以上所有运算后此时的 p 的_x, _y 变回 1.
第八章 1. (2)



```
Microsoft Visual Studio  X + - □ X

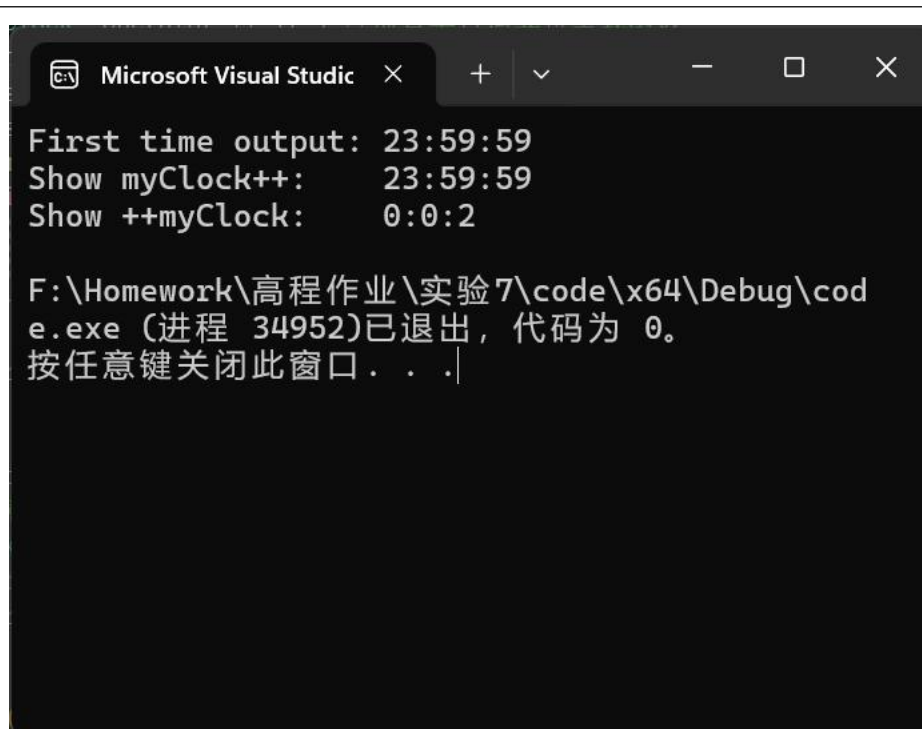
run a motorcycle
stop a motorcycle
run a bicycle
stop a bicycle
run a motorcar
stop a motorcar
run a vehicle
stop a vehicle

F:\Homework\高程作业\实验7\code\x64\Debug\code.exe (进程 22416)已退出，代码为 0。
按任意键关闭此窗口. . .
```

因为基类里的 Run() 和 Stop() 函数均为虚函数，因此不会被隐藏，而可以被重写。因此在派生类中尽管都声明了新的 Run() 和 Stop() 函数，原本基类中的 Run() 和 Stop() 函数依然可以使用基类名称和：：的方式来调用。

第八章 2. 出错原因为：重载的+运算符是在 Complex 类内进行重载的，因此运算需要满足 Complex 类型对象+Complex 类型对象才行，而 1 在默认情况下不会隐式地转换为 Complex 类型的对象而是按照 int 类型来处理，又因为不存在 int 类型+Complex 类型的运算符重载，所以 1+c2 无法进行计算；如果把形参中的 const 去掉，main 函数里用 c3=c1+1 也不可以，因为在这个计算中 1 会被隐式地转换为一个 Complex 对象，并且作为右值来进行计算，而重载的+中需要的是一个 Complex 对象的左值引用，因此也没有能够对应的上的+运算符的重载。

第八章 3. 更改后的代码输出为：



```
Microsoft Visual Studio × + - □ ×  
First time output: 23:59:59  
Show myClock++: 23:59:59  
Show ++myClock: 0:0:2  
  
F:\Homework\高程作业\实验7\code\x64\Debug\code.exe (进程 34952)已退出, 代码为 0。  
按任意键关闭此窗口. . .
```

从结果来看, 前置运算符成功实现连续加 1, 但后置运算符没有实现连续加 1; 根据代码所写, 在重载后置++运算符时返回的是 old 这个临时的 Clock 对象, 因此计算时在对 myClock 进行第一次后置自增之后此时再进行的是 (old++). showTime(), 即下一步是对这个临时的 Clock 对象进行后置自增运算, 所以连续的后++++不能实现连续自增, 而前置的自增因为返回的是*this, 即 myClock 自己, 所以在进行了一次前置自增之后在进行的是 (++myClock). showTime(), 因此前置自增可以实现连续自增, 所以在输出结果中经过两次后置自增之后输出的是 23:59:59, 此时如果再调用 myClock.showTime(), 将输出 0:0:0, 而经过两次自增运算符之后, myClock 的秒位加 2, 所以输出 0:0:2。

第八章 4. (1) 可以执行, 原因为: 在 Complex 类中有声明带默认参数的构造函数, 因此如果将 c1 或者 c2 改成数字, 该数字在计算时会被隐式地构造为一个 Complex 对象, 而这个数字的值会带入 Complex 类的构造函数中, 因此如果将 c1 改为 1, 则此时进行的是一个 real 为 1, imag 为 0 的 Complex 对象的右值引用和 c2 的左值引用相加, 而因为重载的+运算符的两个参数为 const Complex &类型, 这个计算的 1 和 c2 符合这个+运算符的重载的参数类型, 因此可以进行+计算。

第八章 4. (2) 实现如下:

```

class Complex { //复数类定义
public: //外部接口
    Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {} //构造函数
    double r()const;
    double i()const;
private: //私有数据成员
    double real; //复数实部
    double imag; //复数虚部
};

double Complex::i()const
{
    return this->imag;
}

double Complex::r()const
{
    return this->real;
}

Complex operator + (const Complex& cc1, const Complex& cc2)
{
    return Complex(cc1.r() + cc2.r(), cc1.i() + cc2.i());
}

Complex operator - (const Complex& cc1, const Complex& cc2)
{
    return Complex(cc1.r() - cc2.r(), cc1.i() - cc2.i());
}

ostream& operator << (ostream& out, const Complex& c)
{
    out << "(" << c.r() << ", " << c.i() << ")";
    return out;
}

```

第八章 4. (3) 重载为非成员函数这四个表达式都可以正常运行，而重载为成员函数只有 `c2+1.0` 和 `c2+2` 可以运行；如果要解决出的错误，需要在类外再重载一次 `+` 运算符，使第一个参数为 `int` 类型，第二个参数为 `Complex` 类型。

第八章 6. 它与成员函数形式实现的主要区别在于 1. 成员函数形式实现在调用时，操作符左侧的对象被隐式地传递给成员函数作为调用对象，而友元函数形式实现需要显式地传递操作符左侧的对象作为函数参数。2. 成员函数形式实现可以继承和多态，因为它们是类的成员函数，可以在派生类中进行重写。而友元函数形式实现不会继承和多态，因为它们不是类的成员函数。3. 成员函数重载双目操作符必须保证该操作符左侧为这个类的对象，而使用友元函数来实现可以自定义双目操作符两侧的对象类型；可以使用非友元函数来实现，但需要在类内定义一些 `public` 的函数来作为获取数据的接口。

第八章 7. 重载的=如下

```

SimpleCircle SimpleCircle::operator=(SimpleCircle tem)
{
    int* temp = new int(*tem.itsRadius);
    this->itsRadius = temp;
    return *this;
}

```

经过 `X=Y=2` 连续赋值后输出结果如下

```
Microsoft Visual Studio X + - □ X
CircleX: 2
CircleY: 2

F:\Homework\高程作业\实验7\code\x64\Debug\code.exe (进程 31584)已退出, 代码为 0。
按任意键关闭此窗口 . . .
```

第八章 8. 在 `fun(Base *b)` 函数中, 通过 `delete b` 释放了传递进来的指针 `b` 所指向的内存。然而, 在 `main` 函数中, 我们使用 `new` 运算符为 `Base` 类型的指针 `b` 分配了内存, 并将其指向一个 `Derived` 类型的对象。由于 `Base` 类的析构函数没有被声明为虚函数, 因此在 `delete b` 时, 只会调用 `Base` 类的析构函数, 而不会调用 `Derived` 类的析构函数。这可能导致 `Derived` 类中分配的内存没有得到正确释放, 从而造成内存泄漏。

第八章 8. (a) 输出如下

```
Microsoft Visual Studio X + - □ X
Derived destructor
Base destructor
Derived destructor
Base destructor
Base destructor

F:\Homework\高程作业\实验7\code\x64\Debug\code.exe (进程 23748)已退出, 代码为 0。
按任意键关闭此窗口 . . .
```

这五次析构函数的调用来源如下: 首先使用 `new` 来初始化一个 `Base` 类的指针 `b`, 它指向一个 `Derived` 类的对象, 而因为 `Base` 类的析构函数和 `Derived` 类的析构函数均为虚函数, 因此在执行 `delete b` 时, 程序会按照析构一个 `Derived` 类的对象的方式来析构 `b` 所指向的对象, 因此有了前两行的输出; 然后因为使用一个 `Derived` 类初始化一个 `Base` 类的对象 `b1`, 在初始化时 `Derived()` 会构造一个临时的 `Derived` 类的对象用于 `b1` 的初始化, 在初始化结束后, 这个临时的 `Derived` 类的对象会析构, 此时会分别调用派生类和基类的析构函数, 因此有了第三、四的输出, 然后因为程序执行完毕, 释放 `b1` 的内存, 而 `b1` 是一个 `Base` 类的对象, 因此会调用 `Base` 类的析构函数, 因此有了第五行的输出。

第八章 8. (b) 程序会出问题, 会在执行过程中出问题, 具体问题为: `delete` 语句是用来释放由 `new` 申请来的内存的, 而这两句试图用 `delete` 语句对编译器自动管理的非动态分配内存进行释放, 因此程序在执行到 `fun` 函数中 `delete` 语句时程序抛出了错误。

第八章 9. 重载是指在同一个作用域内，可以定义多个具有相同名称但参数列表不同的函数。通过参数的个数、类型或顺序的不同，可以区分并调用不同的重载函数。

覆盖是指在派生类中重新实现基类中的虚函数。派生类可以通过相同的函数名和参数列表来覆盖基类中的虚函数，从而提供自己的实现逻辑。在运行时，通过基类指针或引用调用虚函数时，将根据对象的动态类型来决定调用哪个派生类中的实现。

隐藏是指派生类中的函数屏蔽了基类中具有相同名称的函数。当派生类定义了与基类中同名的函数时，无论该函数是否为虚函数，基类中的函数都将被隐藏。在使用派生类对象调用同名函数时，将只调用派生类中的函数，而基类中的函数将不可见。

因此，重载是通过参数列表的不同来区分函数，覆盖是在派生类中重新实现基类的虚函数，而隐藏是指派生类屏蔽了基类中具有相同名称的函数。

第八章 10. 虚基类是用于解决多继承中的菱形继承问题。当一个派生类从多个基类继承，并且这些基类之间存在继承关系时，如果不使用虚基类，会导致派生类中存在多个基类子对象的副本，造成数据冗余和访问的二义性。而使用虚基类，派生类只包含虚基类的一个实例，避免了数据的冗余和二义性问题。虚基类通过在继承关系中添加虚基类关键字来声明，派生类通过直接或间接继承虚基类，形成虚基类子对象。

虚函数是用于实现运行时多态性的机制。虚函数是在基类中声明为虚函数的成员函数，通过使用关键字 `virtual` 进行声明。派生类可以覆盖基类中的虚函数，提供自己的实现。通过基类指针或引用调用虚函数时，根据对象的动态类型来决定实际调用的函数，实现动态绑定的特性。这使得我们可以在不知道对象的具体类型的情况下，通过基类指针或引用调用派生类的特定实现。

因此，虚基类主要解决多继承中的继承关系和数据冗余问题，而虚函数实现了运行时的多态性，允许在派生类中重新定义基类的行为。虚基类关注的是继承关系和数据共享，而虚函数关注的是多态性和动态绑定。

结论分析与体会：

掌握了运算符重载的方法，包括类内运算符的重载，类外运算符的重载，使用友元函数的运算符重载，以及了解在运算符重载中使用引用时设计的左值引用和右值引用的问题。理解了虚函数的功能和存在的意义以及其对多态的重要价值，并且掌握了虚函数的覆盖等特性。

就实验过程中遇到的问题及解决处理方法(如有)：

未遇到问题