

第8讲

多态 ——运算符重载

主讲人：赵文彬

复习：继承与派生的概念

- **继承 (Inheritance) 机制**：在C++中可以利用已有的类来定义新的类，新类将拥有原有类的全部属性
- **原有的类被称为基类 (base class) 或父类 (super class)**，student类
- **新产生的类被称为派生类 (derived class) 或子类**，postgradudent类

复习：派生类定义的语法

```
class 派生类名: 继承方式1 基类名1, 继承方式2 基类名2,...  
{  
    private:  
        派生类的私有数据和函数  
    public:  
        派生类的公有数据和函数  
    protected:  
        派生类的保护数据和函数  
};
```

复习：继承方式

- 继承方式指定了派生类成员对于从基类继承来的成员的访问权限。
- 继承方式有三种：**public: 公有继承**；**private: 私有继承**；**protected: 保护继承**。

基类属性 继承方式	public	protected	private
public	public	protected	不可访问
protected	protected	protected	不可访问
private	private	private	不可访问

复习：公有继承

- 基类的公有成员在派生类中仍然为公有成员，可以由派生类对象和派生类成员函数直接访问。
- 基类的私有成员在派生类中，无论是派生类的成员还是派生类的对象都无法直接访问。
- 保护成员在派生类中仍是保护成员，可以通过派生类的成员函数访问，但不能由派生类的对象直接访问。

复习：多边形

```
#include <iostream>
```

```
using namespace std;
```

```
class CPolygon {
```

```
protected:
```

```
    int width, height;
```

```
public:
```

```
    void set_values (int a, int b)
```

```
    { width=a; height=b; }
```

```
};
```

```
class CRectangle: public CPolygon {
```

```
public:
```

```
    int area ()
```

```
    { return (width * height); }
```

```
};
```

```
class CTriangle: public CPolygon {
```

```
public:
```

```
    int area ()
```

```
    { return (width * height / 2); }
```

```
};
```

```
int main () {
```

```
    CRectangle rect;
```

```
    CTriangle trgl;
```

```
    rect.set_values (4,5);
```

```
    trgl.set_values (4,5);
```

```
    cout << rect.area() << endl;
```

```
    cout << trgl.area() << endl;
```

```
    return 0;
```

```
}
```

```
C:\Windows\system32\cmd.exe
```

```
20
```

```
10
```

```
Press any key to continue . . .
```

复习：说明

- 公有继承是最常用的一种继承方式，此时由于基类的私有成员不能在子类中访问，一般地将基类的私有成员设为protected类型。

派生类的构造函数

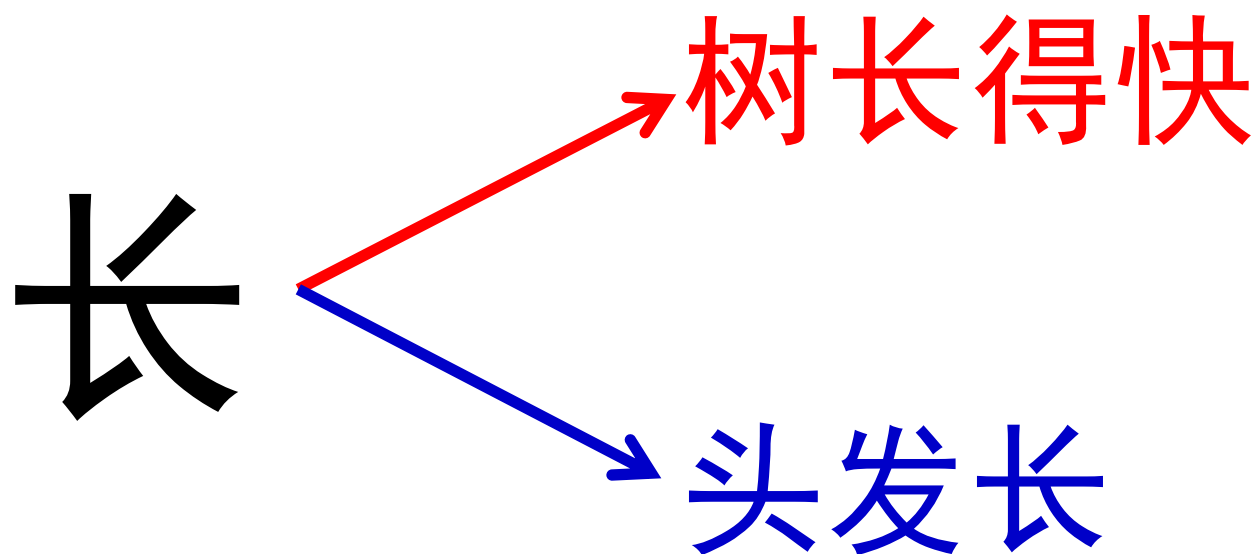
注意：这里的
参数是实参而
不是形参

派生类构造函数的一般形式：

```
派生类构造函数名(总参数表)(: 基类构造函数名(参数表))  
{  
    派生类中新增数据成员初始化  
}
```

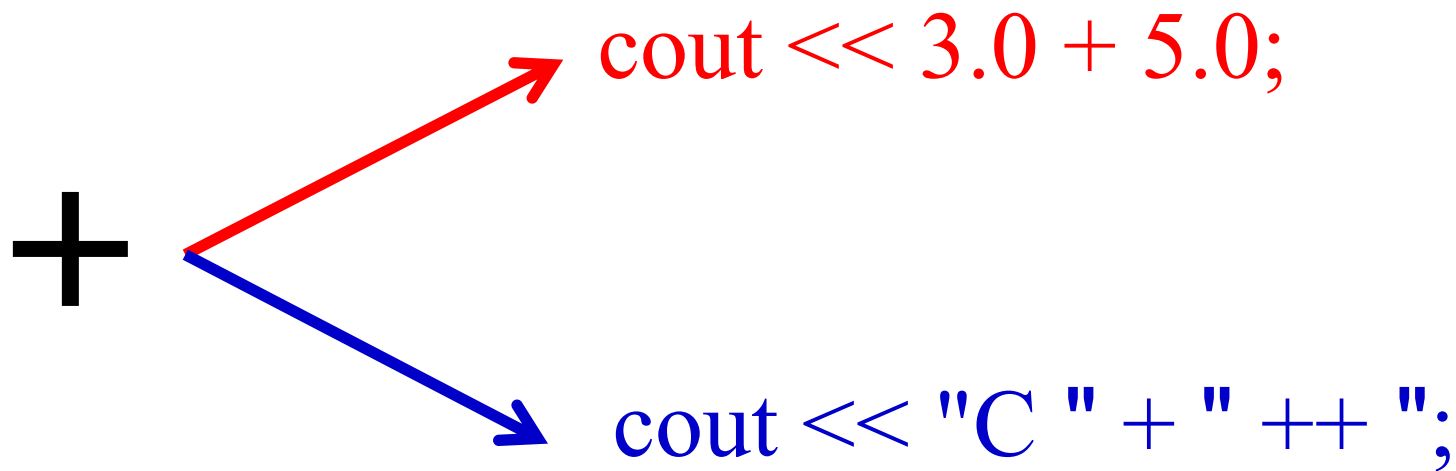

多态

➤ 一词多义



C++中的多态

➤ 我们已经用过多态：运算符



C++中的多态

➤ 我们已经用过多态：函数重载

//point.h

class point

{

public:

point(void);

point(double xx, double yy, double zz);

~point(void);

private:

double x,y,z;

};

多态

- 多态性(polymorphism)是面向对象程序设计的重要特性之一。
- 多态是指同样的消息被不同类型的对象接收时导致完全不同的行为。
- 分为静态多态（主要是函数重载和运算符重载）和动态多态（主要是虚函数）

运算符重载

复数加法

```
//Complex.h
#pragma once
#include <iostream>
using namespace std;
class Complex
{
public:
```

```
~Complex(void);
void display();
Complex operator + (Complex B);
private:
double real;
double image;
};
```

```
Complex(double real=0.0,double image=0.0)
{
    this->real=real,this->image=image;
}
```

复数加法

```
//Complex.cpp
```

```
#include "Complex.h"
```

```
Complex::~~Complex(void) {}
```

```
void Complex::display() {
```

```
    cout << real;
```

```
    if(image > 0)
```

```
        cout << " + ";
```

```
    cout << image << "i";
```

```
}
```

```
Complex Complex::operator +(Complex B) {
```

```
    return Complex(this->real + B.real, this->image + B.image);
```

```
}
```

复数加法

```
//main.cpp
```

```
#include "Complex.h"
```

```
void main()
```

```
{
```

```
    Complex A(1.0,2.0),B(3.0,4.0),C;
```

```
    cout << "Complex A: ";
```

```
    A.display();
```

```
    cout << endl;
```

```
    cout << "Complex B: ";
```

```
    B.display();
```

```
    cout << endl;
```

```
    C = A + B;
```

```
C:\Windows\system32\cmd.exe
```

```
Complex A: 1 + 2i
```

```
Complex B: 3 + 4i
```

```
Complex A + B: 4 + 6i
```

```
Press any key to continue . . .
```

```
    cout << "Complex A + B: ";
```

```
    C.display();
```

```
    cout << endl;
```

```
}
```


能重载的运算符

- 除了以下五个运算符之外，其余全部可以
 - `.` 成员选择运算符
 - `.*` 成员指针运算符
 - `::` 作用域分辨符
 - `? :` 三目选择运算符
 - `sizeof()` 计算数据大小运算符

运算符重载基本原则

- 重载后运算符的优先级与结合性不会改变
- 不能改变原运算符操作数的个数
- 不能重载C++中没有的运算符
- 不能改变运算符的原有语义

重载的两种方式

□ 重载为类的友元函数

- 双目运算符
- 具有交换性的运算符
- 输入输出
- 左边的操作数不是类的类型

□ 重载为类的成员函数

- 单目运算符
- 需要修改类的数据

复数

```
//Complex.h
```

```
#pragma once
```

```
#include <iostream>
```

```
using namespace std;
```

```
class Complex
```

```
{
```

```
public:
```

```
    Complex(double real=0.0, double  
image=0.0)
```

```
{
```

```
    this->real=real;
```

```
    this->image=image;
```

```
}
```

```
~Complex(void);
```

```
void display();
```

```
Complex Complex::operator ++()
```

```
{
```

```
    return Complex(++real,image);
```

```
}
```

```
Complex Complex::operator
```

```
++(int)
```

```
{
```

```
    return Complex(real++,image);
```

```
}
```

```
private:
```

```
    double real;
```

```
    double image;
```

```
};
```

复

```
//Complex.cpp
#include "Complex.h"
Complex::~Complex(void)
{
}
void Complex::display()
{
    cout << real;
    if(image > 0)
        cout << " + ";
    cout << image << "i";
}
```

```
//main.cpp
#include "Complex.h"
void main()
{
    Complex A(1.0,2.0);
    cout << "Complex A: ";
    A.display();
    cout << endl;
    cout << "Complex A++: ";
    (A++).display();
    cout << "Complex A: ";
    A.display();
    cout << endl << endl;
}
```

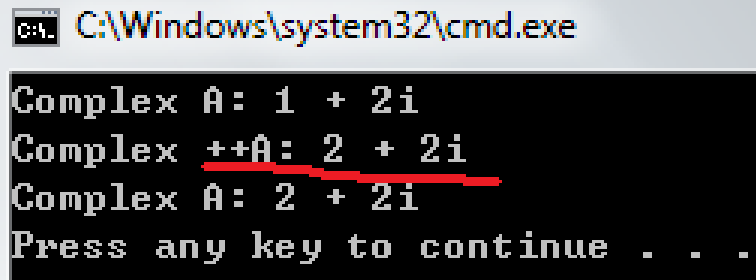
C:\Windows\system32\cmd.exe

```
Complex A: 1 + 2i
Complex A++: 1 + 2i
Complex A: 2 + 2i
Press any key to continue . . .
```

复

```
//Complex.cpp
#include "Complex.h"
Complex::~Complex(void)
{
}
void Complex::display()
{
    cout << real;
    if(image > 0)
        cout << " + ";
    cout << image << "i";
}
```

```
//main.cpp
#include "Complex.h"
void main()
{
    Complex A(1.0,2.0);
    cout << "Complex A: ";
    A.display();
    cout << endl;
    cout << "Complex ++A: ";
    (++A).display();
    cout << endl;
    cout << "Complex A: ";
    A.display();
    cout << endl << endl;
}
```



```
C:\Windows\system32\cmd.exe
Complex A: 1 + 2i
Complex ++A: 2 + 2i
Complex A: 2 + 2i
Press any key to continue . . .
```

复数减法

```
//Complex.h
#pragma once
#include <iostream>
using namespace std;
class Complex
{
public:
    Complex(double
real=0.0,double image=0.0)
    {
        this->real=real;
        this->image=image;
    }
    ~Complex(void);
    void display();
    friend Complex operator-(Complex A, Complex B);
private:
    double real;
    double image;
};
```

复数减法

```
//Complex.cpp
#include "Complex.h"
Complex::~Complex(void)
{
void Complex::display()
{
    cout << real;
    if(image > 0)
        cout << " + ";
    cout << image << "i";
}
```


复数减法

```
//main.cpp
```

```
#include "Complex.h"
```

```
Complex operator-(Complex A,
```

```
Complex B)
```

```
{
```

```
    Complex C;
```

```
    C.real = A.real - B.real;
```

```
    C.image = A.image - B.image;
```

```
    return C;
```

```
}
```

```
void main()
{
    Complex A(1.0,2.0),B(3.0,4.0),C;
    cout << "Complex A: ";
    A.display();
    cout << endl;
    cout << "Complex B: ";
    B.display();
    cout << endl;
    C = A - B;
    cout << "Complex A - B: ";
    C.display();
    cout << endl;
}
```

C:\Windows\system32\cmd.exe

```
Complex A: 1 + 2i
Complex B: 3 + 4i
Complex A - B: -2-2i
Press any key to continue . . .
```

复数输入输出

```
//Complex.h
#pragma once
#include <iostream>
using namespace std;
class Complex
{
public:
    Complex(double real=0.0,double image=0.0)
    {
        this->real=real,this->image=image;
    }
    ~Complex(void){}
```

复数输入输出

```
friend istream& operator >> (istream&, Complex&);
```

```
friend ostream& operator << (ostream&, Complex&);
```

```
private:
```

```
double real;
```

```
double image;
```

```
};
```

```
//main.cpp
```

```
#include "Complex.h"
```

```
istream& operator >> (istream& input, Complex& C)
```

```
{
```

```
input >> C.real >> C.image;
```

```
return input;
```

```
}
```

复数输入输出

```
//main.cpp
```

```
ostream& operator << (ostream& output, Complex& C)
```

```
{
```

```
    output << C.real;
```

```
    if(C.image > 0)
```

```
        output << " + ";
```

```
    output << C.image << "i";
```

```
    return output;
```

```
}
```



```
cmd C:\Windows\system32\cmd.exe
```

```
1 2  
Complex A: 1 + 2i  
Press any key to continue . . .
```

```
void main()
```

```
{
```

```
    Complex A;
```

```
    cin >> A;
```

```
    cout << "Complex A: " << A;
```

```
}
```

输入输出运算符重载

- 说明：输入输出运算符的重载必须为友元函数，并且必须有输入输出流的返回值，固定格式为：

```
friend istream& operator >> (istream&, 重载类名&);
```

```
friend ostream& operator << (ostream&, 重载类名&);
```

小结

- 运算符重载
 - 重载为类的成员函数
 - 重载为类的友元函数
 - 输入输出流运算符

课堂练习

- 设计一个三维向量类vector3D，有无参数和有参数的构造函数；重载向量的加法+、减法-、数乘*（要求乘数在前）这三个运算符，以及输入输出运算符。在主函数中对这些重载的运算符进行调用。