C++程序设计

主讲：王老师

尚德机构

学习是一种信仰

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

class Point
{      //定义类Point
private:
        double x,y;  //类Point的数据成员
public:
        Point( ){ };  //类Point的无参数构造函数
        Point(double a,double b) {x=a;y=b;}   //具有两个参数的构造函数
        void Setxy(double a,double b) {x=a;y=b;} //成员函数，用于重新设置数据成员
        void Display( ){cout<<x<<"\t"<<y<<endl;} //成员函数，按指定格式输出数据成员
};

void main( )
{
        Point a;  //定义类Point的对象a
        Point b(18.5,10.6);  //定义类Point的对象b并初始化
        a.Setxy(10.6,18.5);  //为对象a的数据成员赋值
        a.Display();  //显示对象a的数据成员
        b.Display();  //显示对象b的数据成员
}
```

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

void main( ){
	complex <int> num1(2,3);
	complex <float> num2(3.5,4.5);
	string str1("real is ");
	string str2="image is ";
	cout<<str1<<num1.real()<<','<<str2<<num1.imag()<<endl;
	cout<<str1<<num2.real()<<','<<str2<<num2.imag()<<endl;
}
```

```
real is 2,image is 3
real is 3.5,image is 4.5
```

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
void main( )
{
    string str1="we are here!",str2=str1;
    reverse(&str1[0],&str1[0]+12);
    copy(&str1[0],&str1[0]+12,&str2[0]);
    cout<<str1<<endl<<str2<<endl;
    reverse_copy(&str2[0],&str2[0]+12,ostream_iterator<char>(cout));
}
```

```
!ereh era ew
!ereh era ew
we are here!
```

```
#include <iostream>
#include <string>
#include <algorithm>
#include <functional>
using namespace std;

void main( )
{
    string str1="wearehere!",str2(str1);
    reverse(str1.begin( ),str1.end( ));  //str1逆向
    cout<<str1<<endl;     //输出str1="!ereheraew"
    copy(str1.begin( ),str1.end( ),str2.begin( ));
    sort (str1.begin( ),str1.end( ));  //按默认升幂排序str1
    cout<<str1<<endl;             //输出str1 = "!aeeeehrrw"
    cout<<str2<<endl;             //输出str2 = "!ereheraew"
    reverse_copy(str1.begin( ),str1.end( ),str2.begin( ));
    cout<<str2<<endl;             //输出str2 = "wrrheeeea!"
    reverse(str2.begin( )+2,str2.begin( )+8); //此时str2 = "wreeeehra!"
    copy(str2.begin( )+2,str2.begin( )+8,ostream_iterator<char>(cout));
    //输出"eeeehr"
    sort(str1.begin( ),str1.end( ),greater<char>( ));  //str1降幂排列
    cout<<str1<<endl;             //输出str1 = "wrrheeeea！ "

    str1.swap(str2);             //互换内容
    cout<<str1<<" "<<str2<<endl;
      //输出wreeeehra!(str1)  wrrheeeea!(str2)

    cout<<(*find(str1.begin( ),str1.end( ),'e')=='e')<<" "
        <<(*find(str1.begin( ),str1.end( ),'e')=='o')<<endl;
}           //输出1 0， 注意上面的find不是成员函数find
```

```
!ereheraew
!aeeeehrrw
!ereheraew
wrrheeeea!
eeeehrwrrheeeea!
wreeeehra! wrrheeeea!
1 0
```

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
void main( )
{
     string str[ ]={"we are here!","where are you?","welcome"};
     for(int i = 0;i<3;i++){
        copy(str[i].begin( ),str[i].end( ),ostream_iterator<char>(cout));
        cout<<endl;
      }    //for循环，换行分别输出we are here!  Where are you?  Welcome!
     str[0].swap(str[2]);  //互换，str[0] = "Welcome!" str[2] = "we are here!"
     str[0].swap(str[1]);  //互换，str[0] = "Where are you?" str[1] = "Welcome!"
     for(i=0;i<3;i++)
        cout<<str[i]<<endl;      //for循环，换行分别输出Where are you?
}
```

```
we are here!
where are you?
welcome
where are you?
welcome
we are here!
```

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

void main( ){
    int i(0);
    complex <int> num1(2,3);
    complex <double> num2(3.5,4.5);//用构造函数complex初始化num2并赋值
    printer(num1);
    printer(num2);
}

template <class T>
void printer(complex <T> a)
{
    string str1("real is "),str2="imag is ";
    cout<<str1<<a.real( )<<','<<str2<<
              a.imag( )<<endl;
}
```

```
real is 2,imag is 3
real is 3.5,imag is 4.5
```

```
#include <iostream>
using namespace std;
class Point {                    //使用内联函数定义类point
private:
        int x,y;                 //私有数据成员
public:
        void Setxy(int a,int b)        {x=a;y=b;}
        void Move(int a,int b)         {x=x+a;y=y+b;}
        void Display( )  {cout<<x<<","<<y<<endl;}
        int Getx( ) {return x;}
        int Gety( ) {return y;}
};    //类定义以分号结束
void print(Point *a){a->Display( );}
void print(Point&a){a.Display( );}
void main( ){
        Point A,B,*p;  //声明对象和指针
        Point &RA=A;  //声明对象RA为对象A的引用
        A.Setxy(25,55);          //使用成员函数为对象A赋值
        B=A;         //例如通过int x = 25， y = 55； 对类的私有数据赋值是错误的
        p=&B;
        p->Setxy(112,115);  //使用指针调用函数setxy重设B的值
        print (p);          //传递指针显示对象B的属性
        p->Display( );        //使用指针调用display函数显示对象B的属性
        RA.Move(-80,23);
        print(A);           //使用对象和对象指针的效果一样
        print(RA);}
```

```
112,115
112,115
-55,78
-55,78
```

```
#include <iostream>
using namespace std;
class Point {
private:
        int X,Y;
public:
        Point(int a=0,int b=0){X=a;Y=b;cout<<"初始化中"<<endl;}
     //定义有默认参数的构造函数（且为内联公有成员函数）
Point(const Point &p);      //声明复制构造函数
int GetX( ){return X;}
int GetY( ){return Y;}
void Show( ){cout<<"X="<<X<<",Y="<<Y<<endl;}
~Point( ){cout<<"删除…"<<X<<","<<Y<<endl;}
};
Point::Point(const Point &p){X=p.X;Y=p.Y;cout<<"拷贝初始化中
"<<endl;}   //定义必须使用对象的引用做形参的复制构造函数
void display(Point p){p.Show( );}    //点类对象做函数形参
void disp(Point &p){p.Show( );}      //点类对象的引用做函数形参
Point fun( ){Point A(101,202);return A;}  //函数返回值为点类对象
```

```
void main( ){
   Point A(42,35);    //定义点类对象A并赋值
   Point B(A);        //定义点类对象B，调用复制构造函数用A初始化B
   Point C(58,94);    //定义点类对象C并赋值
   cout<<"called display(B)"<<endl;
   display(B);
   cout<<"下一个…"<<endl;
   cout<<"called disp(B)"<<endl;
   disp(B);
   cout<<"call C  =  fun( )"<<endl;
   C=fun( );
   cout<<"called disp(C)"<<endl;
   disp(C);
   cout<<"out…"<<endl;
}
```

```
初始化中
拷贝初始化中
初始化中
called display(B)
拷贝初始化中
X=42,Y=35
删除…42,35
下一个…
called disp(B)
X=42,Y=35
call C  =  fun( )
初始化中
拷贝初始化中
删除…101,202
删除…101,202
called disp(C)
X=101,Y=202
out…
删除…101,202
删除…42,35
删除…42,35
```

```
#include <iostream>
using namespace std;
class Point{                    //定义点类Point
int x,y;                //没有说明的，默认性质是private
public:
void Set(int a,int b){x=a;y=b;}  //定义内联的公有成员函数
int Getx( ){return x;}          //定义内联的公有成员函数
int Gety( ){return y;}          //定义内联的公有成员函数
};
class Rectangle{                          //定义矩形类Rectangle
Point Loc;
int H,W;                        //定义矩形类的高H和宽W
public:
void Set(int x,int y,int h,int w);
        Point * GetLoc( );    //声明返回Point类指针的成员函数GetLoc
        int GetHeight( ){return H;}       //定义内联的公有成员函数
        int GetWidth( ){return W;}        //定义内联的公有成员函数
};
void Rectangle::Set(int x,int y,int h,int w){Loc.Set(x,y); H=h;W=w;}
Point * Rectangle::GetLoc( ){return &Loc;}
```

```
void main( ){
Rectangle rect;  //定义Rectangle类的对象rect
rect.Set(10,2,25,20);
cout<<rect.GetHeight( )<<","<<rect.GetWidth(
)<<",";     //输出"25,20,"
Point * p=rect.GetLoc( );
cout<<p->Getx( )<<","<<p->Gety( )<<endl;
            //输出"10,2"
}
```

25,20,10,2

```
class Test{
    static int x;            //声明静态数据成员
    int n;
public:
    Test( ){ }      //定义无参数的Test类的构造函数
    Test(int a,int b){x=a;n=b;}  //定义含两个参数的Test类的构造函数Test为内联函数
    static int func( ){return x;}  //定义静态成员函数func为内联函数
    static void sfunc(Test&r,int a){r.n=a;}  //定义静态成员函数sfunc为内联函数,函数以Test类的引用r和整形数a为参数
    int Getn( ){return n;}  //定义成员函数Getn为内联函数
};                    //类Test的声明结束
int Test::x=25;    //初始化静态数据成员
#include <iostream>
using namespace std;
void main( ){
    cout<<Test::func( );    //x在对象产生之前就存在，输出"25"
    Test b,c;       //利用无参数的构造函数产生Test类的对象b和c
    b.sfunc(b,58);  //设置对象b的数据成员n，n值为58，r为b的引用
    cout<<" "<<b.Getn( );  //输出" 58"
    cout<<" "<<b.func( );  //x属于所有对象，输出" 25"
    cout<<" "<<c.func( );  //x属于所有对象，输出" 25"
    Test a(24,56);  //利用含两个参数的构造函数产生Test类的对象a，并将x的值改为24，给a的私有数据成员n赋值56
    cout<<" "<<a.func( )<<" "<<b.func( )<<" "<<c.func( )<<endl;
}
```

```
25 58 25 25 24 24 24
```

```
#include <iostream>
#include <cmath>
using namespace std;
class Point {
private:
    double X,Y;
public:
    Point( double xi,double yi){X=xi,Y=yi;}  //类Point的构造函数
    double GetX( ){return X;}
    double GetY( ){return Y;}
    friend double distances( Point&, Point&);  //声明友元函数
};
double distances( Point& a, Point& b)  //像普通函数一样定义友元函数
{   double dx=a.X-b.X;  //因是友元函数，所以可以直接访问对象的私有数据成员
    double dy=a.Y-b.Y;  //因是友元函数，所以可以直接访问对象的私有数据成员
    return sqrt( dx*dx + dy*dy );
}
void main( ){
    Point p1(3.5,5.5),p2(4.5,6.5);
    cout<<"距离是"<<distances(p1,p2)<<endl;
}
```

```
距离是1.41421
```

```
#include <iostream>
using namespace std;
class Point{
private:
        int x,y;
public:
        Point(int a,int b){x=a;y=b;cout<<"点"<<" ";}
        void Showxy( ){cout<<"x="<<x<<",y="<<y<<" ";}
        ~Point( ){cout<<"删除点"<<" ";}
};
class Rectangle:public Point{
private:
        int H,W;
public:
        Rectangle(int a,int b,int h,int w):Point(a,b){H=h;W=w;cout<<"矩形"<<" ";} //构造函数初始化列表
        void Show( ){cout<<"H="<<H<<",W="<<W<<" ";}
        ~Rectangle( ){cout<<"删除矩形"<<" ";}
};
void main(){
        Rectangle r1(3,4,5,6);   //生成派生类对象r1，r1先后调用基类和派生类的构造函数进行初始化
        r1.Showxy( );   //派生对象调用基类的成员函数
        r1.Show( );   //派生对象调用派生类的成员函数
}
```

```
点 矩形 x=3,y=4 H=5,W=6 删除矩形 删除点
```

使用类模板的实例

```
template <class T>  //带参数T的类模板声明，可用typename代替class
class TAnyTemp{    //类声明
    T x,y;    //声明类型为T的私有数据成员
Public:
    TAnyTemp(T X,T Y):x(X),y(Y) { }  //类TAnyTemp的构造函数，实参类型为T
    T getx( ){return x;}  //返回类型为T的内联成员函数
    T gety( ){return y;}  //返回类型为T的内联成员函数
};
```

```
template <class T>
class Max4 {
          T a,b,c,d;  //四个类型为T的私有数据成员
          T Max(T a,T b){return (a>b)?a:b;}
                                          //类型为T，参数类型为T，返回a、b二者最大值的私有成员函数
        public:
           Max4(T,T,T,T);  //声明构造函数，含4个类型为T的参数
          T Max(void);     //声明返回值类型为void的公有成员函数
};
template <class T>          //定义成员函数必须再次声明类模板
Max4<T>::Max4(T x1,T x2,T x3,T x4):a(x1),b(x2),c(x3),d(x4) { }
template <class T>       //定义成员函数必须再次声明类模板
T Max4<T>::Max(void)
 {return Max(Max(a,b),Max(c,d));}
                                         //定义类Max4的成员函数Max(void)，定义时要将Max<T>看作整体
void main( ){
        Max4 <char> C('W','w','a','A');  //比较字符
        Max4 <int> A(-25,-67,-66,-256);   //比较整数
        Max4 <double> B(1.25,4.3,-8.6,3.5);   //比较双精度实数
        cout<<C.Max( )<<" "<<A.Max( )<<" "<<B.Max( )<<endl;}
```

```
w -25 4.3
```

```
#include <iostream>
using namespace std;
class Point{
            int x,y;
public:
            Point(int a,int b){x=a;y=b;}  //类Point的构造函数
            void display(){cout<<x<<","<<y<<endl;}  //类Point的公有成员函数
};
template <typename T>  //声明继承之前，需重新声明类模板
class Line:public Point{  //模板类Line公有继承非模板类Point
           T x2,y2;
public:
           Line(int a,int b,T c,T d):Point(a,b)  {x2=c;y2=d;}   //类Line的构造函数
           void display(){Point::display();  cout<<x2<<","<<y2<<endl;}};
void main(){
           Point a(3,8);
           a.display();  //输出3，8
           Line<int> ab(4,5,6,7);   //线段ab两个点的坐标均是整数
           ab.display();  //输出4，5　6，7
           Line<double> ad(4,5,6.5,7.8);   //线段ad一个点的坐标是整数，另一个是实数
           ad.display();  //输出4，5　6.5，7.8
}
```

```
3,8
4,5
6,7
4,5
6.5,7.8
```

```
#include <iostream>
using namespace std;
template <typename T>
class Point{
        T x,y;
public:
        Point(T a,T b){x=a;y=b;}  //模板类Point的构造函数
        void display(){cout<<x<<","<<y<<endl;}  //模板类Point的公有成员函数
};
template <typename T>  //声明继承之前，需重新声明类模板
class Line:public Point<T>{  //模板类Line公有继承模板类Point
        T x2,y2;
public:
        Line(T a,T b,T c,T d):Point<T>(a,b){x2=c;y2=d;}  //模板类Line的构造函数
        void display() {Point<T>::display(); cout<<x2<<","<<y2<<endl;}};
void main(){
        Point <double> a(3.5,8.8);
        a.display();  //输出3.5，8.8
        Line<int> ab(4,5,6,7);   //全部使用整数
        ab.display();  //输出4，5   6，7
        Line<double> ad(4.5,5.5,6.5,7.5);   //全部使用实数
        ad.display(); } //输出4.5，5.5   6.5，7.5
```

```
3.5,8.8
4,5
6,7
4.5,5.5
6.5,7.5
```

分别使用指针和引用的display函数

```
#include <iostream>
using namespace std;
const double PI=3.14159;
class Point {
private:
        double x,y;
public:
        Point(double i,double j) {x=i;y=j;}
        virtual double area(){return 0;}
};
class Circle:public Point {
private:
        double radius;
public:
        Circle(double a,double b,double r):Point(a,b){radius=r;}
        double area() {return PI*radius*radius;}
};
void display(Point *p){cout<<p->area()<<endl;}
void display(Point&a){cout<<a.area()<<endl;}
```

```
void main(){
    Point a(1.5,6.7);
    Circle c(1.5,6.7,2.5);
    Point *p=&c;  //派生类对象的地址赋给基类指针
    Point &rc=c;  //派生类对象初始化基类引用
    display(a);  //基类对象调用基类虚函数area，输出0
    display(p);  //指针调用派生类虚函数area，输出19.6349
    display(rc); //指针调用派生类虚函数area，输出19.6349
}
```

```
0
19.6349
19.6349
```

使用友元函数重载运算符<<和>>

```
#include <iostream.h>
class test {
private:
          int i;
          float f;
          char ch;
public:
          test(int a=0,float b=0,char c='\0') {i=a;f=b;ch=c;}
          friend ostream &operator<<(ostream &,test);
          friend istream &operator>>(istream &,test &);
};
ostream &operator<<(ostream & stream,test obj)
{
          stream<<obj.i<<",";  //stream是cout的别名
          stream<<obj.f<<",";
          stream<<obj.ch<<endl;
          return stream;
}
```

```
istream &operator>>(istream & t_stream,test&obj)
{
        t_stream>>obj.i;    //t_stream是cin的别名
        t_stream>>obj.f;
        t_stream>>obj.ch;
        return t_stream;
}
void main( ) {
        test A(45,8.5,'W');
        operator<<(cout,A);
        test B,C;
        cout<<"Input as i f ch:";
        operator>>(cin,B);  operator>>(cin,C);
        operator<<(cout,B);  operator<<(cout,C);
}
```

```
45,8.5,W
Input as i f ch:5 5.8 A 2 3.4 a
5,5.8,A
2,3.4,a
```

使用类运算符重载“++”运算符

```
#include <iostream>
using namespace std;
class number {
        int num;
public:
        number (int i) {num=i;}
        int operator++();      //前缀：++n
        int operator++(int);   //后缀：n++
        void print() {cout<<"num="<<num<<endl;}
};
int number::operator ++()   {   num++;      return num;}
int number::operator ++(int){   int i=num;          num++; return i;}  //不用给出形参名
void main() {
        number n(10);
        int i=++n;    // i=11,n=11，使用函数调用方式的语句为int i=n.operator++();
        cout<<"i="<<i<<endl;  //输出i=11
        n.print();  //输出n=11;
        i=n++;    // i=11,n=12, 使用函数调用方式的语句为i=n.operator++(0);
        cout<<"i="<<i<<endl;  //输出i=11
        n.print();  //输出n=12
}
```

```
i=11
num=11
i=11
num=12
```

使用友元运算符重载++运算符

```
#include <iostream>
using namespace std;
class number {
        int num;
public:
        number (int i) {num=i;}
        friend int operator++(number&);    //前缀：++n
        friend int operator++(number&,int);    //后缀：n++
        void print() {cout<<"num="<<num<<endl;}
};
int operator ++(number& a) { a.num++;   return a.num;}
int operator ++(number& a,int){   int i=a.num++;   return i;} //不用给出形参名
void main() {
        number n(10);
        int i=++n;    //i=11,n=11
        cout<<"i="<<i<<endl;  //输出i=11
        n.print();  //输出n=11;
        i=n++;    //i=11,n=12
        cout<<"i="<<i<<endl;  //输出i=11
        n.print();  //输出n=12
}
```

```
i=11
num=11
i=11
num=12
```

使用对象作为友元函数参数来定义运算符+的例子

```
#include <iostream.h>
class complex {
private:
        double real,imag;
public:
        complex(double r=0,double i=0) {real=r;imag=i;}  //构造函数
        friend complex operator+(complex,complex);
        void show() {cout<<real<<"+"<<imag<<"i"<<endl;}
};
complex operator+(complex a,complex b)
{
        double r=a.real+b.real;
        double i=a.imag+b.imag;
        return complex(r,i);
}
void main() {
        complex x(5,3),y;
        y=x+7;  //相当于"y=operator+(x,7);"，通过调用构造函数将x和7初始化，有y=12+3i
        y=7+y;  //相当于"y=operator+(7,y);"，通过调用构造函数将7和y初始化，有y=19+3i
        y.show();  //输出19+3i
}
```

演示文件流的概念

```
abcdefgGoodBye!
```

```
#include <iostream>
#include <fstream>        //输入输出文件流头文件
using namespace std;
void main() {
        char ch[15],*p="abcdefg";
        ofstream myFile;      //建立输出流myFile
        myFile.open("myText.txt");      //建立输出流myFile和文件myText.txt之间的关联
        myFile<<p;      //使用输出流myFile将指针p所指字符串流向文件
        myFile<<"GoodBye!";      //使用输出流myFile直接将字符串流向文件
        myFile.close();      //关闭文件myText.txt
        ifstream getText("myText.txt");   //建立输入流getText及其和文件myText.txt的关联并打开
        for(int i=0;i<strlen(p)+8;i++)  //使用输入流getText每次从文件myText.txt读入1个字符
                getText>>ch[i];      //将每次读入的1个字符赋给数组的元素ch[i]
        ch[i]='\0';      //设置结束标志
        getText.close();      //关闭文件
        cout<<ch;      //使用cout使数组元素流向屏幕，输出“abcdefgGoodBye!”
}
```

祝大家顺利通过考试！