c++中赋值运算的返回值是=左边的引用；

非引用的函数返回值不能作为赋值的左值使用；

在 C++ 中，类型的名字（包括类的名字）本身也是一种运算符，即类型强制转换运算符。类型强制转换运算符是单目运算符，也可以被重载，但只能重载为成员函数，不能重载为全局函数。经过适当重载后，(类型名)对象这个对对象进行强制类型转换的表达式就等价于对象.operator类型名()，即变成对运算符函数的调用。

详见

http://c.biancheng.net/view/244.html

将类对象转换成基本类型

注意

1转换函数必须是类方法

2转换函数不能指定返回类型

3转换函数不能有参数

在类中定义类型转换函数的形式一般为：  
　　　　operator 目标类型名();

有以下几个使用要点：  
1 转换函数必须是成员函数，不能是友元形式。  
2转换函数不能指定返回类型，但在函数体内必须用return语句以传值方式返回一个目标类型的变量。  
3 转换函数不能有参数。

## 面向对象

对象间的运算

和结构变量一样，对象之间可以用“=”进行赋值，但是不能用“==”，“！=”，“>”，“<”“）=”“<=”进行比较，除非这些运算符经过了“重载”。

### 引用的概念

>下面的写法定义了一个引用，并将其初始化为引用某个变量。

类型名&引用名=某变量名；

Int n=4；int & r=n;//r引用了n, r的类型是int &

>某个变量的引用，等价于这个变量，相当于该变量的一个别名。

>定义引用时一定要将其初始化成引用某个变量。

>初始化后，它就一直引用该变量，不会再引用别的变量了。

>引用只能引用变量，不能引用常量和表达式。

改变函数的返回值，那么必须使用引用

常引用

定义引用时，前面加const关键字，即为“常引用”

int n;const int &r=n;r的类型是const int &

不能通过常引用去修改其引用的内容：

常引用和非常引用的转换

const T&和T&是不同的类型！！！

T &类型的引用或T类型的变量可以用来初始化const T &类型的引用。

const T类型的常变量和const T &类型的引用则不能用来初始化T &类型的引用，除非进行强制类型转换。

### “const”关键字的用法

**1)**定义常量

**2)**定义常量指针

口 不可通过常量指针修改其指向的内容

口 不能把常量指针赋值给非常量指针，反过来可以

口 函数参数为常量指针时，可避免函数内部不小心改变参数指针所指地方的内容

口 不能通过常引用修改其引用的变量

### 动态内存分配

用new运算符实现动态内存分配

口 第一种用法，分配一个变量：

P=new T;

T是任意类型名，P是类型为T\*的指针。

动态分配出一片大小为sizeof（T）字节的内存空间，并且将该内存空间的起始地址赋值给P。

口 第二种用法，分配一个数组：

P=new T[N];

T：任意类型名

P：类型为T\*的指针

N:要分配的数组元素的个数，可以是整型表达式

动态分配出一片大小为sizeof（T）\*N字节的内存空间，并且将该内存空间的起始地址赋值给P。

用delete运算符释放动态分配的内存

口 用“new”动态分配的内存空间，一定要用“delete”运算符进行释放

delete指针；//该指针必须指向new出来的空间

口 用“delete”释放动态分配的数组，要加“[]”

delete[]指针；//该指针必须指向new出来的数组

内联函数

口 函数调用是有时间开销的。如果函数本身只有几条语句，执行非常快，而且函数被反复执行很多次，相比之下调用函数所产生的这个开销就会显得比较大。

口 为了减少函数调用的开销，引入了内联函数机制。编译器处理对内联函数的调用语句时，是将整个函数的代码插入到调用语句处，而不会产生调用函数的语句。

函数重载（教材P67)

口 一个或多个函数，名字相同，然而参数个数或参数类型不相同，这叫做函数的重载。

>以下三个函数是重载关系：

int Max(double f1,double f2){}

int Max(int nl,int n2){}

int Max(int nl,int n2,int n3){}

>函数重载使得函数命名变得简单。

>编译器根据调用语句的中的实参的个数和类型判断应该调用哪个函数。

函数的缺省参数

口 C++中，定义函数的时候可以让最右边的连续若干个参数有缺省值，那么调用函数的时候，若相应位置不写参数，参数就是缺省值。

void func( int x1,int x2=2,int x3=3){ }

## 类和对象

变量一通过类，可以定义变量。类定义出来的变量，也称为类的实例，就是我们所说的“对象”

C++中，类的名字就是用户自定义的类型的名字。可以象使用基本类型那样来使用它。CRectangle就是一种用户自定义的类型。

### 对象的内存分配

和结构变量一样,对象所占用的内存空间的大小，等于所有成员变量的大小之和。

对于上面的CRectangle类， sizeof (CRectangle) =8?每个对象各有自己的存储空间。一个对象的某个成员变量被改变了，不会影响到另一个对象。

### 对象间的运算

和结构样，对象之间可以用“=”进行赋值，但是不能用==”，“!=”，“>”，“<”,“>=”,“<=”进行比较，除非这些运算符经过了“重载”。

### 使用类的成员变量和成员函数用法

用法1. 对象名.成员名

用法2. 指针-〉成员名

用法3. 引用名.成员名

### 类成员的可访问范围

在类的定义中，用下列访问范围关键字来说明类成员可被访问的范围: .

- private:私有成员,只能在成员函数内访问

- public:公有成员，可以在任何地方访问

- protected:保护成员，以后再说

以上三种关键字出现的次数和先后次序都没有限制。

#### 定义一个类

class className {

private :

私有属性和函数

public:

公有属性和函数

protected:

保护属性和函数};

如果某个成员前面没有上述关键字，则缺省地被认为是私有成员。

在类的成员函数内部，能够访问:

-当前对象的全部属性、函数;

-同类其它对象的全部属性、函数。

在类的成员函数以外的地方，只能够访问该类对象的公有成员。

设置私有成员的机制，叫‘"隐藏”

"隐藏”的目的是强制对成员变量的访问一定要通过成员函数进行，那么以后成员变量的类型等属性修改后，只需要更改成员函数即可。否则，所有直接访问成员变量的语句都需要修改。

#### 用struct定义类

和用"class"的唯一-区别，就是未说明是公有还是私有的成员，就是公有

###### 成员函数的 重载及参数缺省

成员函数也可以重载 

成员函数可以带缺省参数。

#include <iostream>

using namespace std;

class Location {

private :

int x, y;

public:

void init( int x=0 , int y = 0 );

void valueX( int val ) { x = val ;}

int valueX() { return x; }

};

void Location::init( int X, int Y) {

x = X; y = Y; }

int main() {

Location A,B;

1. init(5); A.valueX(5);

cout << A.valueX(); return 0; }

输出： 5

使用缺省参数要注意避免有函数重载时的二义性

class Location {

private : int x, y;

public:

void init( int x =0, int y = 0 );

void valueX( int val = 0) { x = val; }

int valueX() { return x; }

};

Location A;

1. valueX(); //错误，编译器无法判断调用哪个valueX

### 构造函数 (constructor)

基本概念(教材P179)

 成员函数的一种

名字与类名相同，可以有参数，不能有返回值(void也不行) 

作用是对对象进行初始化，如给成员变量赋初值 

如果定义类时没写构造函数，则编译器生成一个默认的无参数的构造函数

•默认构造函数无参数，不做任何操作

 如果定义了构造函数，则编译器不生成默认的无参数的构造函数 

对象生成时构造函数自动被调用。对象一旦生成，就再也不能在 其上执行构造函数 

一个类可以有多个构造函数

为什么需要构造函数：

1) 构造函数执行必要的初始化工作，有了构造函数，就不必专门再写初始化函数，也不用担心忘记调用初始化函数。

2) 有时对象没被初始化就使用，会导致程序出错。

可以有多个构造函数，参数个数或类型不同

构造函数最好是public的，private构造函数 不能直接用来初始化对象

##### 构造函数在数组中的使用

class Test {

public:

Test( int n) { } //(1)

Test( int n, int m) { } //(2)

Test() { } //(3) };

Test array1[3] = { 1, Test(1,2) }; // 三个元素分别用(1),(2),(3)初始化

Test array2[3] = { Test(2,3), Test(1,2) , 1}; // 三个元素分别用(2),(2),(1)初始化

Test \* pArray[3] = { new Test(4), new Test(1,2) }; //两个元素分别用(1),(2) 初始化

\* pArray[3] 是个指针，创建指针，不创建对象

new Test(4) 返回值为指针

### 复制构造函数 copy constructor

只有一个参数,即对同类对象的引用。 

形如 X::X( X& )或X::X(const X &), 二者选一 后者能以常量对象作为参数 

不允许有形如 X::X( X )的构造函数。参数必须是引用不能是对象

如果没有定义复制构造函数，那么编译器生成默认复制构造函数。默认的复制构造函数完成复制功能。

如果定义了自己的复制构造函数， 则默认的复制构造函数不存在。

class Complex {

public :

double real,imag;

Complex(){ }

Complex( const Complex & c ) {

real = c.real;

imag = c.imag;

cout << “Copy Constructor called”; }

};

Complex c1;

Complex c2(c1);//调用自己定义的复制构造函数，输出 Copy Constructor called

##### 复制构造函数起作用的三种情况

1)当用一个对象去初始化同类的另一个对象时。

Complex c2(c1);

Complex c2 = c1; //初始化语句，非赋值语句，复制构造函数起作用；c2=x构造函数起作用

2)如果某函数有一个参数是类A 的对象，那么该函数被调用时，类A的复制构造函数将被调用。

class A {

public: A() { };

A( A & a) {

cout << "Copy constructor called" <<endl;

}

};

void Func(A a1){ }

int main(){

A a2;

Func(a2);

return 0;

}

程序输出结果为: Copy constructor called

1. 如果函数的返回值是类A的对象时，则函数返回时， A的复制构造函数被调用:

class A {

public:

int v;

A(int n) { v = n; };

A( const A & a) {

v = a.v;

cout << "Copy constructor called" <<endl;

}

};

A Func() {

A b(4);

return b;

}

int main() {

cout << Func().v << endl;

return 0;

}

输出结果：

Copy constructor called

4

注意：对象间赋值并不导致复制构造函数被调用

常量引用参数的使用

void fun(CMyclass obj\_ ) {

cout << "fun" << endl;

} 

这样的函数，调用时生成形参会引发复制构造函数调用，开销比较大。 

所以可以考虑使用 CMyclass & 引用类型作为参数。 

如果希望确保实参的值在函数中不应被改变，那么可以加上const 关键字：

void fun(const CMyclass & obj) {

//函数中任何试图改变 obj值的语句都将是变成非法

}

##### 类型转换构造函数

什么是类型转换构造函数

 定义转换构造函数的目的是实现类型的自动转换。 

只有一个参数，而且不是复制构造函数的构造函数，一般就可以看作是转换构造函数。 

当需要的时候，编译系统会自动调用转换构造函数，建立一个无名的临时对象(或临时变量)。

#### 析构函数 destructors

什么是析构函数

名字与类名相同，在前面加‘~’， 没有参数和返回值，一 个类最多只能有一个析构函数。

析构函数对象消亡时即自动被调用。可以定义析构函数来在 对象消亡前做善后工作，比如释放分配的空间等。

如果定义类时没写析构函数，则编译器生成缺省析构函数。 缺省析构函数什么也不做。

如果定义了析构函数，则编译器不生成缺省析构函数。

析构函数和数组

对象数组生命期结束时，对象数组的每个元素的析构函数都会被调用。

delete 运算导致析构函数调用。

若new一个对象数组，那么用delete释放时应该写 []。否则只delete一个对象(调用一次析构函数)

### this指针

this指针作用 

其作用就是指向成员函数所作用 的对象

非静态成员函数中可以直接使用this来代表指向该函数作用的对象的指针。

静态成员函数中不能使用 this 指针！因为静态成员函数并不具体作用与某个对象!因此，静态成员函数的真实的参数的个数，就是程序中写出的参数个数！

### 静态成员

静态成员：在定义前面加了static关键字的成员

普通成员变量每个对象有各自的一份，而静态成员变 量一共就一份，为所有对象共享。

sizeof 运算符不会计算静态成员变量。

普通成员函数必须具体作用于某个对象，而静态成员函数并不具体作用于某个对象。

因此静态成员不需要通过对象就能访问。

静态成员变量本质上是全局变量，哪怕一个对象都不存在，类的静态成员变量也存在。

静态成员函数本质上是全局函数。 

设置静态成员这种机制的目的是将和某些类紧密相关的全局变量和函数写到类里面，看上去像一个整体，易于维护和理解。

如何访问静态成员

1. 类名::成员名 CRectangle::PrintTotal();
2. 对象名.成员名 CRectangle r; r.PrintTotal();
3. 指针->成员名 CRectangle \* p = &r; p->PrintTotal();
4. 引用.成员名 CRectangle & ref = r; int n = ref.nTotalNumber;

注意事项  在静态成员函数中，不能访问非静态成员变量， 也不能调用非静态成员函数。

### 成员对象和封闭类（P196) 

有成员对象的类叫封闭（enclosing)类。

任何生成封闭类对象的语句，都要让编译器明白，对象中的成员对象，是如何初始化的。 具体的做法就是：通过封闭类的构造函数的初始化列表。成员对象初始化列表中的参数可以是任意复杂的表达式，可以包括函数，变量 ，只要表达式中的函数或变量有定义就行。

#### 封闭类造函数和析构函数的执行顺序

封闭类对象生成时，先执行所有对象成员的构造函数，然后才执行封闭类的构造函数。

对象成员的构造函数调用次序和对象成员在类中的说明次序一致 ，与它们在成员初始化表中出现的次序无关。 

当封闭类的对象消亡时，先执行封闭类的析构函数，然后再执行 成员对象的析构函数。次序和构造函数的调用次序相反。

封闭类的复制构造函数(P198) ：

封闭类的对象，如果是用默认复制构造函数初始化的，那么它里面包含的成员对象， 也会用复制构造函数初始化。

### 友元（friends,P199）

友元分为友元函数和友元类两种

1. 友元函数: 一个类的友元函数可以访问该类的私有成员.

可以将一个类的成员函数(包括构造、析构函数)说明为另一个类的友元。

1. 友元类: 如果A是B的友元类，那么A的成员函数可以访问B的私有成员。

友元类之间的关系不能传递，不能继承。

常量成员函数 

如果不希望某个对象的值被改变，则定义该对象的时候可以在前面加 const关键字。

在类的成员函数说明后面可以加const关键字，则该成员函数成为常量 成员函数。 

常量成员函数内部不能改变属性的值，也不能调用非常量成员函数。

class Sample {

private :

int value;

public:

void func() { };

Sample() { }

void SetValue() const {

value = 0; // wrong

func(); //wrong }

};

const Sample Obj;

Obj.SetValue (); //常量对象上可以使用常量成员函数

在定义常量成员函数和声明常量成员函数时都应该使用const 关键字。

如果一个成员函数中没有调用非常量成员函数 ，也没有修改成员变量的值，那么，最好将其写成常量成员函数。

##### 常量成员函数的重载 

两个函数，名字和参数表都一样，但是一个是const,一个不是，算重载

mutable成员变量：

可以在const成员函数中修改的成员变量

# 运算符重载

运算符重载的需求

C++预定义的运算符，只能用于基本数据类型的运算：整型、实型、字符型、逻辑型 .......

+、-、 \* 、/、%、 ^ 、&、~、!、|、 = 、<< >>、!=、……

在数学上，两个复数可以直接进行+、-等运算。但在C++中，直接将+或-用于复数对象是不允许的。

有时会希望，让对象也能通过运算符进行运算。这样代码更简洁，容易理解。

• 例如： complex\_a和complex\_b是两个复数对象； 求两个复数的和, 希望能直接写： complex\_a + complex\_b

##### 运算符重载的形式

运算符重载，就是对已有的运算符(C++中预定义的运算符)赋予多 重的含义，使同一运算符作用于不同类型的数据时导致不同类型的行为。 

运算符重载的目的是：扩展C++中提供的运算符的适用范围，使之 能作用于对象。 

同一个运算符，对不同类型的操作数，所发生的行为不同。 

complex\_a + complex\_b 生成新的复数对象 

5 + 4 = 9

运算符重载的实质是函数重载 

可以重载为普通函数，也可以重载为成员函数 

把含运算符的表达式转换成对运算符函数的调用。 

把运算符的操作数转换成运算符函数的参数。 

运算符被多次重载时，根据实参的类型决定调用哪个运算符函数。

运算符重载的形式

返回值类型 operator 运算符（形参表） { …… }

重载为成员函数时，参数个数为运算符目数减一。 重载为普通函数时，参数个数为运算符目数。

### 赋值运算符的重载

有时候希望赋值运算符两边的类型可以不匹配， 比如，把一个int类型变量赋值给一个Complex对象， 或把一个 char \* 类型的字符串赋值给一个字符串对 象,此时就需要重载赋值运算符“=”。

赋值运算符“=”只能重载为成员函数

##### 浅拷贝和深拷贝(P213)

class String {

private:

char \* str;

public:

String ():str(new char[1]) { str[0] = 0;}

const char \* c\_str() { return str; };

String & operator = (const char \* s){

delete [] str;

str = new char[strlen(s)+1];

strcpy( str, s);

return \* this;

};

~String( ) { delete [] str; }

};

如不定义自己的赋值运算符，那么S1=S2实际上导致 S1.str和 S2.str 指向同一地方。 

如果S1对象消亡，析构函数将释放 S1.str指向的空间，则S2消亡时还 要释放一次，不妥。  另外，如果执行 S1 = "other"；会导致S2.str指向的地方被delete 

因此要在 class String里添加成员函数:

String & operator = (const String & s){

if( this == & s)

return \* this;

delete [] str;

str = new char[strlen(s.str)+1];

strcpy( str,s.str);

return \* this;

}

对 operator = 返回值类型的讨论

void 好不好？

String 好不好?

为什么是 String &

对运算符进行重载的时候，好的风格是应该尽量保留运算符原本的特性

考虑： a = b = c; 和 (a=b)=c; //会修改a的值

分别等价于： a.operator=(b.operator=(c)); (a.operator=(b)).operator=(c);

### 运算符重载 为友元函数

一般情况下，将运算符重载为类的成员函数，是较好的选择。 

但有时，重载为成员函数不能满足使用要求，重载为普通函数，又不能访问类的私有成员，所以需要将运算符重载为友元。

### 流插入运算符和 流提取运算符的重载

cout 是在 iostream 中定义的，ostream 类 的对象。 

“<<” 能用在cout 上是因为，在iostream 里对 “<<” 进行了重载。 

考虑,怎么重载才能使得 cout << 5; 和 cout << “this”都能成立

ostream & operator<<( ostream & o,const CStudent & s){

o << s.nAge ;

return o;

}

类型转换运算符和 自增、自减运算符 的重载

重载类型转换运算符(P220)

#include

using namespace std;

class Complex {

double real,imag;

public:

Complex(double r=0,double i=0):real(r),imag(i) { };

operator double () { return real; } //重载强制类型转换运算符 double

};

int main() {

Complex c(1.2,3.4);

cout << (double)c << endl; //输出 1.2

double n = 2 + c; //等价于 double n=2+c.operator double()

cout << n; //输出 3.2

}

类型强制转换运算符被重载时不能写返回值类型，实际上其返回值类型就是该类型强制转换运算符代表的类型

自增，自减运算符的重载（P221)

自增运算符++、自减运算符--有前置/后置之分，为了区分所重载的是前 置运算符还是后置运算符，C++规定： 

前置运算符作为一元运算符重载

重载为成员函数：

T & operator++();

T & operator--();

重载为全局函数：

T1 & operator++(T2);

T1 & operator—(T2)

后置运算符作为二元运算符重载，多写一个没用的参数：

重载为成员函数：

T operator++(int);

T operator--(int);

重载为全局函数：

T1 operator++(T2,int );

T1 operator—( T2,int);

但是在没有后置运算符重载而有前置重载的情况下， 在vs中，obj++ 也调用前置重载，而dev则令 obj ++ 编译出错

运算符重载的注意事项

1. C++不允许定义新的运算符 ；
2. 重载后运算符的含义应该符合日常习惯； 

complex\_a + complex\_b 

word\_a > word\_b 

date\_b = date\_a + n

1. 运算符重载不改变运算符的优先级；
2. 以下运算符不能被重载：“.”、“.\*”、“::”、“?:”、sizeof；
3. 重载运算符()、[]、->或者赋值运算符=时，运算符重载函数必须声明为类的成员函数。

# 继承和派生

继承和派生的概念

继承：在定义一个新的类B时，如果该类与某个已有的类A相似(指的是B拥有A的全部特点)， 那么就可以把A作为一个基类，而把B作为基类的一个派生类(也称子类)。

派生类是通过对基类进行修改和扩充得到的。在派生类中，可以扩充新的成员变量 和成员函数。  派生类一经定义后，可以独立使用，不依赖于基类。

派生类拥有基类的全部成员函数和成员变 量，不论是private、protected、public 。 

在派生类的各个成员函数中，不能访问 基类中的private成员。

派生类的写法

class 派生类名：public 基类名 { };

## 派生类对象的内存空间

派生类对象的体积，等于基类对象的体积，再加上派生类对象自己的成员变量的体积。在派生类对象中，包含着基类对象，而且基类对象的存储位置位于派生类对 象新增的成员变量之前。

继承关系 和 复合关系

继承：“是”关系。

– 基类 A，B是基类A的派生类。

– 逻辑上要求：“一个B对象也是一个A对象”。

复合：“有”关系。

– 类C中“有”成员变量k，k是类D的对象，则C和D是复合关系

– 一般逻辑上要求：“D对象是C对象的固有属性或组成部分”。

## 派生类覆盖基类成员

覆盖 

派生类可以定义一个和基类成员同名的成员，这叫覆盖。在派生类中访问这类成员时，缺省的情况是 访问派生类中定义的成员。要在派生类中访问由基 类定义的同名成员时，要使用作用域符号::。

## 类的保护成员

另一种存取权限说明符：protected

• 基类的private成员：可以被下列函数访问

– 基类的成员函数

– 基类的友元函数

• 基类的public成员：可以被下列函数访问

– 基类的成员函数

– 基类的友元函数

– 派生类的成员函数

– 派生类的友元函数

– 其他的函数

• 基类的protected成员：可以被下列函数访问

– 基类的成员函数

– 基类的友元函数

– 派生类的成员函数可以访问当前对象的基类的保护成员

## 派生类的构造函数

在创建派生类的对象时，需要调用基类的构造函数：初始化派 生类对象中从基类继承的成员。在执行一个派生类的构造函数 之前，总是先执行基类的构造函数。

调用基类构造函数的两种方式

– 显式方式：在派生类的构造函数中，为基类的构造函数提供 参数. derived::derived(arg\_derived-list):base(arg\_base-list)

– 隐式方式：在派生类的构造函数中，省略基类构造函数时， 派生类的构造函数则自动调用基类的默认构造函数.

派生类的析构函数被执行时，执行完派生类的析构函数后，自动调用基类的析构函数。

## 包含成员对象的派生类的构造函数写法

class Skill {

public:

Skill(int n) { }

};

class FlyBug: public :Bug {

int nWings;

Skill sk1, sk2;

public:

FlyBug( int legs, int color, int wings);

};

FlyBug::FlyBug( int legs, int color, int wings):

Bug(legs,color),sk1(5),sk2(color) ,nWings(wings) { }

##### 封闭派生类对象的构造函数的执行顺序

在创建派生类的对象时:

1. 先执行基类的构造函数，用以初始化派生类对象中从基类 继承的成员；
2. 再执行成员对象类的构造函数，用以初始化派生类对象中 成员对象。
3. 最后执行派生类自己的构造函数

在派生类对象消亡时：

1. 先执行派生类自己的析构函数

2) 再依次执行各成员对象类的析构函数

3) 最后执行基类的析构函数

析构函数的调用顺序与构造函数的调用顺序相反。

## public继承的 赋值兼容规则

class base { };

class derived : public base { };

base b;

derived d;

1. 派生类的对象可以赋值给基类对象

b = d;

1. 派生类对象可以初始化基类引用

base & br = d;

1. 派生类对象的地址可以赋值给基类指针

base \* pb = & d;

如果派生方式是 private或protected，则上述三条不可行。

#### protected继承和private继承 （P240）

protected继承时，基类的public成员和protected成员成为派生类的protected成员。

private继承时，基类的public成员成为派生类的private成员，基类的protected成员成 为派生类的不可访问成员。

protected和private继承不是“是”的关系。

## 基类与派生类的指针强制转换

公有派生的情况下,派生类对象的指针可以直接赋值给基类指针

Base \* ptrBase = &objDerived;

ptrBase指向的是一个Derived类的对象；

\*ptrBase可以看作一个Base类的对象，访问它的public成员直接通过ptrBase即可 ，但不能通过ptrBase访问objDerived对象中属于Derived类而不属于Base类的 成员 

即便基类指针指向的是一个派生类的对象，也不能通过基类指针访问基类没有而派生类中有的成员。 

通过强制指针类型转换，可以把ptrBase转换成Derived类的指针

Base \* ptrBase = &objDerived;

Derived \*ptrDerived = (Derived \* ) ptrBase;

程序员要保证ptrBase指向的是一个Derived类的对象，否则很容易会出错。

直接基类与间接基类

类A派生类B，类B派生类C，类C派生类D，……

– 类A是类B的直接基类

– 类B是类C的直接基类，类A是类C的间接基类

– 类C是类D的直接基类，类A、B是类D的间接基类 A B C D

在声明派生类时，只需要列出它的直接基类

– 派生类沿着类的层次自动向上继承它的间接基类

– 派生类的成员包括

派生类自己定义的成员

直接基类中的所有成员

所有间接基类的全部成员

## 顺序容器和关联容器中都有的成员函数

begin 返回指向容器中第一个元素的迭代器

end 返回指向容器中最后一个元素后面的位置的迭代器

rbegin 返回指向容器中最后一个元素的迭代器

rend 返回指向容器中第一个元素前面的位置的迭代器

erase 从容器中删除一个或几个元素

##### 顺序容器的常用成员函数

clear 从容器中删除所有元素

front :返回容器中第一个元素的引用

back : 返回容器中最后一个元素的引用

push\_back : 在容器末尾增加新元素

pop\_back : 删除容器末尾的元素

erase :删除迭代器指向的元素(可能会使该迭代器失效），或删除一个区间，返回被删除元素后面的那个元素的迭代器

##### list 容器

在任何位置插入删除都是常数时间，不支持随机存取。 

除了具有所有顺序容器都有的成员函数以外，还支持8个成员函数：

push\_front: 在前面插入

pop\_front: 删除前面的元素

sort: 排序 ( list 不支持 STL 的算法 sort)

remove: 删除和指定值相等的所有元素

unique: 删除所有和前一个元素相同的元素(要做到元素不重复，则

unique之前还需要 sort)

merge: 合并两个链表，并清空被合并的那个

reverse: 颠倒链表

splice: 在指定位置前面插入另一链表中的一个或多个元素,并在另一链表中删除被插入的元素

## 关联容器

set, multiset, map, multimap 

内部元素有序排列，新元素插入的位置取决于它的值，查找速度快。 

除了各容器都有的函数外，还支持以下成员函数：

find: 查找等于某个值的元素(x小于y和y小于x同时不成立即为相等)

lower\_bound : 查找某个下界 upper\_bound : 查找某个上界

equal\_range : 同时查找上界和下界

count :计算等于某个值的元素个数(x小于y和y小于x同时不成立即为相等)

insert: 用以插入一个元素或一个区间

multiset的成员函数

iterator find(const T & val); 在容器中查找值为val的元素，返回其迭代器。如果找不到，返回end()。

iterator insert(const T & val); 将val插入到容器中并返回其迭代器。

void insert( iterator first,iterator last); 将区间[first,last)插入容器。

int count(const T & val); 统计有多少个元素的值和val相等。

iterator lower\_bound(const T & val); 查找一个最大的位置 it,使得[begin(),it) 中所有的元素都比 val 小。

iterator upper\_bound(const T & val); 查找一个最小的位置 it,使得[it,end()) 中所有的元素都比 val 大

pair<iterator,iterator> equal\_range(const T & val); 同时求得lower\_bound和upper\_bound。

iterator erase(iterator it); 删除it指向的元素，返回其后面的元素的迭代器(Visual studio 2010上如此，但是在 C++标准和Dev C++中，返回值不是这样)。

### 双向迭代器

若p和p1都是双向迭代器，则可对p、p1可进行以下操作：

++p, p++ 使p指向容器中下一个元素

--p, p-- 使p指向容器中上一个元素

\* p 取p指向的元素 p = p1 赋值

p == p1 , p!= p1 判断是否相等、不等

### 随机访问迭代器

若p和p1都是随机访问迭代器，则可对p、p1可进行以下操作： 

双向迭代器的所有操作 

p += i 将p向后移动i个元素 

p -= i 将p向向前移动i个元素 

p + i 值为: 指向 p 后面的第i个元素的迭代器 

p - i 值为: 指向 p 前面的第i个元素的迭代器 

p[i] 值为: p后面的第i个元素的引用 

p < p1, p <= p1, p > p1, p>= p1 

p – p1 : p1和p之间的元素个数

容器 容器上的迭代器类别

vector 随机访问

deque 随机访问

list 双向

set/multiset 双向

map/multimap 双向

stack 不支持迭代器

queue 不支持迭代器

priority\_queue 不支持迭代器