



# C++程序设计精要教程

华中科技大学

# 第6章 继承与构造

## ◆6.1 单继承类

- 继承是C++类型演化的重要机制，在保留原有类的属性和行为的基础上，派生出的新类可以有某种程度的变异。
- 通过继承，新类自动具有了原有类的属性和行为，因而只需定义原有类型没有的新的数据成员和函数成员。实现了软件重用，使得类之间具备了层次性。
- 通过继承和派生形成的类簇，反映了面向对象问（主）题域、主题等概念，形成类似Java的包。
- 单继承是只有一个基类的继承方式。

# 第6章 继承与构造

## ◆6.1 单继承类

- 多继承的派生类有多于一个的基类，派生类将是所有基类行为的组合。
- 派生类与基类：接受成员的新类称为派生类，如例中的Point类；提供成员的类型称为基类，如例中的Location类。
- 基类是对若干个派生类的抽象，提取了派生类的公共特征；而派生类是基类的具体化，通过增加属性或行为变为更有用的类型。
- 派生类可以看作基类定义的延续，先定义一个抽象程度较高的基类，该基类中有些操作并未实现；然后定义更为具体的派生类，实现抽象基类中未实现的操作。

# 第6章 继承与构造

## ◆6.1 单继承类

- C++通过多种控制派生的方法获得新的派生类，可在定义派生类时：
  - 添加新的数据成员和函数成员；
  - 改变继承来的基类成员的访问权限；
  - 修改继承来的基类成员的访问权限；
  - 重新定义同名的数据和函数成员。



# 第6章 继承与构造

## ◆6.1 单继承类

### ●单继承的定义格式：

- `class <派生类名>:<继承方式><基类名>`

{

    <派生类新定义成员>

    <派生类重定义基类同名的数据和函数成员>

    <派生类声明修改基类成员访问权限>

};

- <继承方式>指明派生类采用什么继承方式从基类获得成员，分为三种：  
    ： **private**表示私有基类； **protected**表示保护基类； **public**表示公有基类。

### ●注意区别继承方式和访问权限。

# 第6章 继承与构造

【例 6.1、 6.2】 分别定义定位坐标LOCATION类和其派生的点POINT类。

```
#include <graphics.h>
class LOCATION{           //定义定位坐标类
    int x, y;
public:
    int getx( ); int gety( );    //gety()获得当前坐标y
    void moveto(int x,int y);    //定义移动坐标函数成员
    LOCATION(int x,int y);
    ~LOCATION( );
};
void LOCATION::moveto(int x,int y){
    LOCATION::x=x;
    LOCATION::y=y;
}
int LOCATION::getx( ){ return x; }
int LOCATION::gety( ){ return y; }
```

# 第6章 继承与构造

```
LOCATION::LOCATION(int x,int y){
    LOCATION::x=x;  LOCATION::y=y;
}
LOCATION::~~LOCATION(){ }
class POINT:public LOCATION{
//定义点类，从LOCATION类继承，继承方式为public
    int visible;                //新增可见属性
public:
    int isvisible(){ return visible; }    //新增函数成员
    void show(),hide();
    void moveto(int x,int y);            //重新定义与基类同名函数
    POINT(int x,int y):LOCATION(x,y){ visible=0; }
//在构造派生类对象前先构造基类对象
    ~POINT(){ hide(); }
};
void POINT::show(){
    visible=1;
```

# 第6章 继承与构造

```
    putpixel(getx( ),gety( ),getcolor( ));  
}  
void POINT::hide( ){  
    visible=0;  
    putpixel(getx( ),gety( ),getbkcolor( ));  
}  
void POINT::moveto(int x,int y){  
    int v=isvisible( );  
    if(v) hide( );  
    LOCATION::moveto(x,y);  
    if(v) show( );  
}  
void main(void){  
    POINT p(3,6);  
    p.LOCATION::moveto(7,8);  
    p.moveto(9,18);  
}
```

带类名访问基类的moveto函数，  
如果不带类名会导致无休止的递归调用。

//不能去掉LOCATION::，会自递归

问题：调用基类moveto函数会导致什么问题？

//调用基类moveto函数  
//调用派生类moveto函数



# 第6章 继承与构造

## ◆6.1 单继承类

### ●关于class、struct、union说明：

- 用class声明的类的继承方式缺省为private，因此，声明**class POINT: private LOCATION**等价于声明**class POINT: LOCATION**。
- 派生类也可以用struct声明，不同之处在于：用struct声明的继承方式和访问权限缺省为public。
- 用union声明的类既不能作派生类的基类，也不能作任何基类的派生类。
- 当基类成员被继承到派生类时，该成员在派生类中的访问权限由继承方式决定。必须慎重的选择继承方式，它是面向对象程序设计的一个非常重要的环节

# 第6章 继承与构造

## ◆6.2 继承方式

●派生类可以有三种继承方式：公有继承`public`、保护继承`protected`、私有继承`private`。基类私有成员对派生类函数是不可见的。

- ①公有继承：基类的公有成员和保护成员派生到派生类时，都保持原有的状态；
- ②保护继承：基类的公有成员和保护成员派生后都成为派生类的保护成员；
- ③私有继承：基类的公有成员和保护成员派生后都作为派生类的私有成员。

# 第6章 继承与构造

## ◆6.2 继承方式

- 基类的私有成员同样也被继承到派生类中，构成派生类的一部分，但对派生类函数成员不可见，不能被派生类函数成员访问。
  - 若派生类函数成员要访问基类的私有成员，则必须将其声明为基类的成员友元。
- 在派生类外部，对其成员访问的权限：
  - 对于新定义成员，按定义时的访问权限访问；
  - 对于继承来的基类成员，取决于这些成员在派生类中的访问权限，与其在基类中定义的访问权限无关。

# 第6章 继承与构造

## ◆6.2 继承方式

- 基类成员继承到派生类时，其访问权限的变化同继承方式有关。
  - 假定访问权限和继承方式满足 **private < protected < public**。如果基类成员的访问权限高于继承方式，则派生后基类成员在派生类中的访问权限和继承方式一样；否则，基类成员的访问权限保持不变。

继承方式 基类成员	private	protected	public
protected	private	protected	protected
public	private	protected	public

- 继承来的基类私有成员不能被派生类函数成员访问。



# 第6章 继承与构造

## ◆6.2 继承方式

基类LOCATION的成员

**private成员:**

**int x,y;**

**public成员:**

**int getx();**

**int gety();**

**void moveto( );**

**LOCATION( );**

**~LOCATION( );**

派生类POINT新增成员

**private成员:**

**int visible;**

**public成员:**

**int isvisible( );**

**void show( );**

**void hide( );**

**void moveto( );**

**POINT( );**

**~POINT( );**

继承方式为public时POINT可访问的成员

**private成员:**

**int visible;**

**public成员:**

**int isvisible( );**

**void show( );**

**void hide( );**

**void moveto( );**

**POINT( );**

**~POINT( );**

**int getx( );**

**int gety( );**

**void LOCATION::moveto( );**

**LOCATION( );**

**~LOCATION( );**

# 第6章 继承与构造

## ◆6.2 继承方式

- 若类POINT的继承方式为public，基类函数getx、gety派生后的访问权限仍为public，对类POINT来说这是合理的，因为，对类POINT来说则类需要这样的函数成员；
- 同上，若类POINT的继承方式为public，基类函数成员moveto派生后的访问权限为public，对类POINT来说则是不合理的，因为类POINT自己定义了public函数成员moveto。在第8页中，主函数还能调用基类函数LOCATION::moveto。

# 第6章 继承与构造

```
class POINT: private LOCATION{...};
```

基类LOCATION的成员

**private成员:**

```
int x,y;
```

**public成员:**

```
int getx( );
```

```
int gety( );
```

```
void moveto( );
```

```
LOCATION( );
```

```
~LOCATION( );
```

派生类POINT新增成员

**private成员:**

```
int visible;
```

**public成员:**

```
int isvisible( );
```

```
void show( );
```

```
void hide( );
```

```
void moveto( );
```

```
POINT( );
```

```
~POINT( );
```

继承方式为private时POINT可访问的成员

**private成员:**

```
int visible;
```

```
int getx( );
```

```
int gety( );
```

```
void LOCATION::moveto( );
```

```
LOCATION( );
```

```
~LOCATION( );
```

**public成员:**

```
int isvisible( );
```

```
void show( );
```

```
void hide( );
```

```
void moveto( );
```

```
POINT( );
```

```
~POINT( );
```

# 第6章 继承与构造

## ◆6.2 继承方式

- 继承方式为private时，基类成员在派生类中的访问权限变为private。不合理时可以使用“基类名::成员”或“using 基类名::成员”**修改某些成员的访问权限, 派生类不能再定义同名的成员。**

```
class POINT:private LOCATION{ //private可省略
    int visible;
public:
    LOCATION::getx; //修改权限成public
    LOCATION::gety; //修改权限成public
    int invisible() { return visible; }
    void show(), hide();
    void moveto(int x,int y);
    POINT(int x,int y):LOCATION(x,y){ visible=0; }
    ~POINT(){ hide(); }
};
```

- 需要指出的是，选用private作继承方式通常不是最好的选择。如果派生类POINT选用private作继承方式，却又未修改LOCATION::getx的访问权限，则getx在POINT类中的访问权限将变为private，从而使其它非派生类成员的函数无法访问private的POINT::getx。



# 第6章 继承与构造

## ◆6.2 继承方式

基类LOCATION的成员

**private成员:**

**int x,y;**

**public成员:**

**int getx( );**

**int gety( );**

**void moveto( );**

**LOCATION( );**

**~LOCATION( );**

派生类POINT新增成员

**private成员:**

**int visible;**

**public成员:**

**int isvisible( );**

**void show( );**

**void hide( );**

**void moveto( );**

**POINT( );**

**~POINT( );**

修改private派生的基类成员访问权限时

**private成员:**

**int visible;**

**void LOCATION::moveto( );**

**LOCATION( );**

**~LOCATION( );**

**public成员:**

**int isvisible( );**

**void show( );**

**void hide( );**

**void moveto( );**

**POINT( );**

**~POINT( );**

**int getx( );**

**int gety( );**

# 第6章 继承与构造

## ◆6.2 继承方式

- 基类成员经过继承方式被继承到派生类后，要注意访问权限的变化。
  - 按面向对象的作用域，同基类同名的派生类成员被优先访问。
  - 派生类中改写基类同名函数时，要注意区分这些同名函数，否则可能造成自递归调用。
- 标识符的作用范围可分为从小到大四种级别：①作用于函数成员内；②作用于类或者派生类内；③作用于基类内；④作用于虚基类内。
  - 标识符的作用范围越小，被访问到的优先级越高。如果希望访问作用范围更大的标识符，则可以用类名和作用域运算符进行限定。

# 第6章 继承与构造

## ◆6.3 成员访问

- 基类成员经过继承方式被继承到派生类后，要注意访问权限的变化。
  - 按面向对象的作用域，同基类同名的派生类成员被优先访问。
  - 派生类中改写基类同名函数时，要注意区分这些同名函数，否则可能造成自递归调用。
- 标识符的作用范围可分为从小到大四种级别：①作用于函数成员内；②作用于类或者派生类内；③作用于基类内；④作用于虚基类内。
  - 标识符的作用范围越小，被访问到的优先级越高。如果希望访问作用范围更大的标识符，则可以用类名和作用域运算符进行限定。

# 第6章 继承与构造

【例6.3】以链表LIST类为基类定义集合类SET。

```
class LIST{
    struct NODE{           //定义节点类
        int val;    NODE *next;
        NODE(int v, NODE *p){ val=v; next=p; }
        ~NODE(){ delete next; next=0; }
    }*head;                //定义数据成员
public:
    int insert(int), contains(int);
    LIST(){ head=0; }       //0表示空指针
    ~LIST(){ if(head){ delete head; head=0; //0表示空指针 } }
};
int LIST::contains(int v){ //搜索链表，查询是否存在该节点
    NODE *h=head;
    while((h!=0)&&(h->val!=v)) h=h->next;
    return h!=0;           //0表示空指针
}
```



# 第6章 继承与构造

```
int LIST::insert(int v){  
    head=new NODE(v,head);  
    return 1;  
}
```

//在链表中插入新增节点

```
class SET:protected LIST{  
    int used;
```

//采用保护继承方式

//集合元素的个数

```
public:  
    LIST::contains;  
    int insert(int);  
    SET(){};
```

//修改contains函数访问权限

//需要改变used值，因此改写insert函数

//等价于SET():LIST(){};

```
};  
int SET::insert(int v){  
    if(!contains(v)&&LIST::insert(v)) return ++used;  
    return 0;  
}
```

//LIST::insert中的LIST不能省略：否则自递归

```
void main(void) { SET s; s.insert(3); s.contains(3); }
```

# 第6章 继承与构造

- 派生类不能访问基类私有成员，除非将派生类的声明为基类的友元类，或者将要访问基类私有成员的派生类函数成员声明为基类的友元。

```
class B;                //前向声明类B
class A{
    int a, b;
public:
    A(int x){a=x;}
    friend B;           //声明B为A的友元类， B类成员可以访问A任何成员
};
class B:A{              //缺省为private继承， 等价于class B: private A{
    int b;
public:
    B(int x):A(x){ b=x; A::b=x; a+=3; } //可访问私有成员A::a,A::b
};
void main(void){ B x(7); }
```

# 第6章 继承与构造

## ◆6.4 构造与析构

- 单继承派生类的构造顺序比较容易确定：
  - 调用虚基类的构造函数；
  - 调用基类的构造函数；
  - 按照派生类中数据成员的声明顺序，依次调用数据成员的构造函数或初始化数据成员；
  - 最后执行派生类的构造函数构造派生类。
- 析构是构造的逆序。
- 以下情况派生类必须定义自己的构造函数：
  - 虚基类或基类只定义了带参数的构造函数；
  - 派生类自身定义了引用成员或只读成员；
  - 派生类需要使用带参数构造函数初始化的对象成员。

# 第6章 继承与构造

```
#include <iostream>
class A{
    int a;
public:
    A(int x):a(x){ std::cout<<a;}//也可在构造函数体内再次对a赋值
    ~A(){cout<<a;}
};
class B:A{ //私有继承，等价于class B: private A{
    int b,c;
    const int d;//B中定义有只读成员，故必须定义构造函数初始化
    A x,y;
public:
    B(int v):b(v),y(b+2),x(b+1),d(b),A(v){//注意构造与出现顺序无关
        c=v;    std::cout<<b<<c<<d;    std::cout<<"C";
    }
    ~B(){ std::cout<<"D";}
};
void main(void){ B z(1); }
```

//派生类成员实际构造顺序为b,d,x,y

//输出结果：123111CD321



# 第6章 继承与构造

## ◆6.4 构造与析构

- 如果虚基类和基类的构造函数是无参的，则构造派生类对象时，构造函数可以不用显式调用它们的构造函数，编译程序会自动调用虚基类或基类的无参构造函数。
- 如果引用变量r引用的是一个对象v，则对象的构造和析构由对象v完成，而不应该由引用变量r完成。如果被引用的对象是用new生成的，则引用变量r必须用delete &r析构对象，否则被引用的对象将因无法完全释放空间（为对象申请的空间）而产生内存泄漏。

# 第6章 继承与构造

【例6.6】 被引用的对象的析构。

```
#include <iostream>
using namespace std;
class A{
    int i;  int *s;
public:
    A(int x){
        s=new int[i=x];
        cout<<"(C): "<<i<<"\n";
    }
    ~A() {
        delete s;
        cout<<"(D): "<<i<<"\n";
    }
};
```

```
void sub1(void) {
    A &p=*new A(1);
} //内存泄露
void sub2(void){
    A *q=new A(2);
} //内存泄露
void sub3(void){
    A &p=*new A(3);
    delete &p;
}
void sub4(void) {
    A *q=new A(4);
    delete q;
}
void main(void){
    sub1();  sub2();
    sub3();  sub4();
}
```

输出:

(C): 1

(C): 2

(C): 3

(D): 3

(C): 4

(D): 4

# 第6章 继承与构造

## ◆6.5 父类和子类

- 如果派生类的继承方式为public，则这样的派生类称为基类的子类，而相应的基类则称为派生类的父类。
- C++允许父类指针直接指向子类对象，也允许父类引用直接引用子类对象。
- 通过父类指针调用虚函数时晚期绑定，根据对象的实际类型绑定到合适的成员函数。
- 父类指针实际指向的对象的类型不同，虚函数绑定的函数的行为就不同，从而产生多态。

# 第6章 继承与构造

## ◆6.5 父类和子类

- 编译程序只能根据类型定义静态地检查语义。由于父类指针可以直接指向子类对象，而到底是指向父类对象还是子类对象只能在运行时确定。
- 编译时，只能把父类指针指向的对象都当作父类对象。因此编译时：
- 父类指针访问对象的数据成员或函数成员时，不能超越父类为相应对象成员规定的访问权限；
- 也不能通过父类指针访问子类新增的成员，因为这些成员在父类中不存在，编译程序无法识别。



# 第6章 继承与构造

【例6.7】 定义点类，并通过点类派生出圆类。

```
#include <iostream>
using namespace std;
class POINT{
    int x,y;
public:
    int getx( ){ return x; }
    int gety( ){ return y; }
    void show( ){ cout<<"Show a point\n"; }
    POINT(int x,int y){ POINT::x=x; POINT::y=y; }
};
class CIRCLE:public POINT{    //公有继承
                               //私有成员
    int r;
public:
    int getr( ){ return r; }
    void show( ){ cout<<"Show a circle\n"; }
    CIRCLE(int x,int y,int r):POINT(x,y){ CIRCLE::r=r; }
};
```

# 第6章 继承与构造

```
void main(void){  
    CIRCLE c(3,7,8);  
    POINT *p=&c; //父类对象指针p可以直接指向子类对象，不用类型转换  
    cout<<"The circle with radius "<<c.getr();  
    //不能使用p->getr(), 因为getr()函数不是父类的函数成员  
    //编译程序无法通过检查  
    cout<<" is at ("<<p->getx()<<"<<p->gety()<<"\n";  
    p->show();  
    //p虽然指向子类对象，但调用的是父类的show函数  
}
```

输出结果：

The circle with radius 8 is at (3,7)

Show a point

# 第6章 继承与构造

- 若基类和派生类没有构成父子关系，则：
  - 普通函数定义的基类指针不能直接指向派生类对象，而必须通过强制类型转换才能指向派生类对象。
  - 普通函数定义的基类引用也不能直接引用派生类对象，而必须通过强制类型转换才能引用派生类对象。

**【例6.7】 引用父类对象的引用变量引用子类对象。**

```
#include <iostream>
using namespace std;
class A{
    int a;
public:
    int getv( ) { return a; }
    A( ) { a=0; }
    A(int x) { a=x; }
    ~A( ) { cout<<"~A\n"; }
};
```

```
class B: A{ //非父子 : private
    int b;
public:
    int getv( ) {
        return b+A::getv( );
    }
    B( ) { b=0; } //等于B( ):A( )
    B(int x):A(x) { b=x; }
    ~B( ) { cout<<"~B\n"; }
};
```

# 第6章 继承与构造

```
class C: public A{ //父子关系
    int c;          //私有成员c
public:
    int getv( ) { return c+A::getv( ); }
    C( ) { c=0; } //等价于C():A() { c=0; }
    C(int x):A(x) { c=x; }
    ~C(){ cout<<"~C\n"; }
};

void main(void){
    A &p=*new C(3);      //直接引用C类对象：A和C父子
    A &q=(A *)new B(5); //强制转换引用B类对象：A和B非父子
    cout<<"p.getv( )="<<p.getv( )<<"\n";
    cout<<"q.getv( )="<<q.getv( )<<"\n";
    delete &p;          //析构C(3)的父类A而非子类C
    delete &q;          //析构B(5)的父类A而非子类B
}
```

输出:

p.getv( )=3

q.getv( )=5

~A

~A



# 第6章 继承与构造

- 在派生类函数成员内部，定义的基类指针可以直接指向该派生类对象，即对派生类函数成员而言，基类被等地当作父类。
- 如果函数声明为派生类的友元，则该友元定义的基类指针也可以直接指向该基类的派生类对象，也不必通过强制类型转换。

**【例6.9】**定义机车类VEHICLE，并派生出汽车类CAR。

```
class VEHICLE{  
    int speed,weight,wheels;  
public:  
    VEHICLE(int spd,int wgt,int whl);  
};
```

# 第6章 继承与构造

```
VEHICLE::VEHICLE(int spd, int wgt, int whl){  
    speed=spd;  weight=wgt;  wheels=whl;  
}  
class CAR: private VEHICLE{  //非父子关系: private  
    int seats;  
public:  
    VEHICLE *who( );  
    CAR(int sd, int wt, int st);  
    friend void main( );  
};  
CAR::CAR(int sd, int wt, int st):VEHICLE(sd, wt, 4) { seats=st; }  
VEHICLE *CAR::who(){  
    VEHICLE *p=this;  //派生类内的基类指针直接指向派生类对象  
    VEHICLE &q=*this; //派生类内的基类引用直接引用派生类对象  
    return p;  
}  
//在派生类的友元main中，基类和派生类构成父子关系  
void main(void) { CAR c(1,2,3); VEHICLE *p=&c;}
```

# 第6章 继承与构造

## ◆6.6 派生类的存储空间

- 派生类的成员一部分是新定义的，另一部分是从基类派生而来的，因此，在派生类对象的存储空间中必然包含了基类的成员。
- 在构造派生类对象之前，首先构造的匿名的基类对象的存储空间，作为派生类对象存储空间的一部分。
- 在计算派生类对象存储空间时，基类和派生类的静态数据成员都不应计算在内。

# 第6章 继承与构造

【例6.10】 派生类对象存储空间的计算方法。

```
#include <iostream>
class A{
    int h, i, j;
    static int k;
};
class B: A{           //等价于class B: private A
    int m,n,p;
    static int q;
};
int A::k=0;           //静态数据成员必须初始化
int B::q=0;
void main(void){
    std::cout<<"Size of int="<<sizeof(int)<<"\n";
    std::cout<<"Size of A="<<sizeof(A)<<"\n";
    std::cout<<"Size of B="<<sizeof(B)<<"\n";
}
```

输出

Size of int=2  
Size of A=6  
Size of B=12

派生类存储空间示意图

int h;	A
int i;	
int j;	
int m;	B
int n;	
int p;	