面向对象程序设计
420420

派生类的构造和析构函数

- 1、派生类的构造函数

- 在定义派生类时，派生类并没有把基类的构造函数和析构函数继承下来。因此，对继承的基类成员初始化的工作要由派生类的构造函数承担，同时基类的析构函数也要被派生类的析构函数来调用。

- 1．派生类构造函数的定义
- 在执行派生类的构造函数时，使派生类的数据成员和基类的数据成员同时都被初始化。其定义形式如下：

```
派生类名(形式参数列表) : 基类名(基类构造函数实参列表),派生类初始化列表
{
        派生类初始化函数体
}
```

- “基类名(基类构造函数实参列表)”即是调用基类构造函数，而派生类新增加的数据成员可以在“派生类初始化列表”（尽量在此）初始化，也可以在“派生类初始化函数体”中初始化。

- 派生类构造函数的调用顺序是：
- ①调用基类构造函数；②执行派生类初始化列表；
- ③执行派生类初始化函数体；
- 例如：

```
class Point {        //基类
  int x,y;   //私有数据成员
  public: Point(int a,int b):x(a),y(b) { } //构造函数
};
class Rect : public Point { //派生类公有继承Point类
    int h,w;   //私有数据成员
    public: Rect(int a,int b,int c,int d):Point(a,b),h(c),w(d) { } //派生类构造函数
};
```

- 2．组合关系的派生类的构造函数
- 假定派生类A和类B的关系是组合关系，类A中有类B的子对象。如果类B有默认构造函数，或者参数全是默认参数的构造函数，或者有无参数的构造函数，那么类A的构造函数中可以不用显式初始化子对象。编译器总是会自动调用B的构造函数进行初始化。
- 可以在一个类的构造函数中显式地初始化其子对象，初始化式只能在构造函数初始化列表中，形式为：

```
类名(形式参数列表) : 子对象名(子对象构造函数实参列表),类初始化列表
{
        类初始化函数体
}
```

- 调用顺序为：
- ①调用基类构造函数；
- ②调用子对象构造函数，各个子对象时按其声明的次序先后调用；
- ③执行派生类初始化列表；
- ④执行派生类初始化函数体；

- 说明：
- （1）如果在基类和子对象所属类的定义中都没有定义带参数的构造函数，而且也不需要对派生类自己的数据成员初始化，那么可以不必显式地定义派生类构造函数。派生类会合成一个默认构造函数，并在调用派生类构造函数时，会自动先调用基类的默认构造函数和子对象所属类的默认构造函数。
- 例如：

```
class A {//类体中没有构造函数 }; //合成默认构造函数
class B {//类体中没有构造函数 }; //合成默认构造函数
class D: public B { A a; }; //派生类合成默认构造函数
```

- （2）如果在基类中没有定义构造函数，或定义了没有参数的构造函数，那么，在定义派生类构造函数时可以不显式地调用基类构造函数。在调用派生类构造函数时，系统会自动先调用基类的无参数构造函数或默认构造函数。

- 例如：

```
class  B  {  public:  B()  {  }  };  //基类B；有个无参数构造函数
class  D:  public  B  {  //派生类D；公有继承B类
       D()  {  }  //派生类构造函数不必显式调用基类构造函数
};
```

- （3）如果在基类或子对象所属类的定义中都定义了带参数的构造函数，那么就必须显式地定义派生类构造函数，并在派生类构造函数中显式地调用基类或子对象所属类的构造函数。
- 例如：

```
class A { public: A(int) { } }; //有参数构造函数
class B { public: B(int) { } }; //有参数构造函数
class D: public B {
        D(int x) : a(x),B(x) { } //显式调用子对象或基类构造函数
        A a;
};
```

- （4）如果在基类中既定义了无参数的构造函数，又定义了有参的构造函数（构造函数重载），则在定义派生类构造函数时，既可以显式调用基类构造函数，也可以不调用基类构造函数。

- 在调用派生类构造函数时，根据构造函数的内容决定调用基类的有参数的构造函数还是无参数的构造函数。

- 可以根据派生类的需要决定采用哪一种方式。

派生类的构造和析构函数

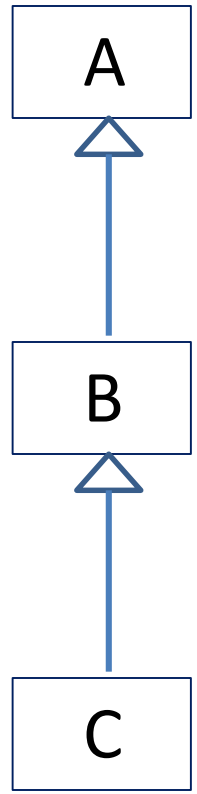
2、派生类的析构函数

- 在派生时，派生类是不能继承基类的析构函数的，也需要通过派生类的析构函数去调用基类的析构函数。
- 派生类中可以根据需要定义自己的析构函数，用来对派生类中所增加的成员进行清理工作。基类的清理工作仍然由基类的析构函数负责。
- 在执行派生类的析构函数时，系统会自动调用基类的析构函数和子对象的析构函数，对基类和子对象进行清理。
- 析构函数调用的顺序与构造函数正好相反：先执行派生类自己的析构函数，对派生类新增加的成员进行清理，然后调用子对象的析构函数，对子对象进行清理，最后调用基类的析构函数，对基类进行清理。

35.2 派生类的析构函数

【例35.1】

```
1   #include <iostream>
2   using namespace std;
3   class A{   //基类
4   public:
5           A(){cout<<“A constructor”<<endl;} //构造函数
6           ~A(){cout<<“A destructor”<<endl;} //析构函数
7   };
8   class B: public A{
9   public:
10          B(){cout<<"B constructor"<<endl;} //构造函数
11          ~B(){cout<<"B destructor"<<endl;} //析构函数
12  };
13  class C: public B{
14  public:
15          C(){cout<<"C constructor"<<endl;} //构造函数
16          ~C(){cout<<"C destructor"<<endl;} //析构函数
17  };
```



```
18      int   main()
19      {
20                C   test;
21                return   0;
22      }
```

运行结果：

```
A  constructor
B  constructor
C  constructor
C  destructor
B  destructor
A  destructor
```

【例35.2】

```
#include <iostream>
#include<string>
using namespace std;
class Undergraduate{ //基类
protected:          //保护部分
        int num;      //学号
        string name; //姓名
        char sex ;   //性别
public:             //公用部分
        Undergraduate(int n,string nam,char s ) //基类构造函数
      { num=n; name=nam; sex=s; }
        ~Undergraduate(){}                      //基类析构函数
};

```

35.2 派生类的析构函数

```
15 class graduate_stu: public Undergraduate{ //公用派生类
16 private: //派生类的私有部分
17              int age;        //年龄
18              string addr;    //地址
19 public:
20          graduate_stu(int n,string nam,char s,int a,string ad ) :
                                Undergraduate(n,nam,s) //派生类构造函数
21          { age=a; addr=ad;} //在函数体中只对派生类新增的数据成员初始化
22          void show( )
23          { cout<<"num: "<<num<<" name: "<<name<<" sex: "<<sex
                            <<" age: "<<age<<" address: "<<addr<<endl;}
24          ~graduate_stu( ) { } //派生类析构函数
26 };
```

```
27 int main( )
28 {    graduate_stu stud1(10010,"Wang-li",'f',19,
                                        "115 Beijing Road,Shanghai");
29       graduate_stu stud2(10011,"Zhang-fun",'m',21,
                                        "213 Shanghai Road,Beijing");
30        stud1.show( ); //输出第一个学生的数据
31        stud2.show( ); //输出第二个学生的数据
32        return 0;
33 }
```

运行结果：

```
num: 10010 name: Wang-li sex: f age: 19 address: 115 Beijing Road,Shanghai
num: 10011 name: Zhang-fun sex: m age: 21 address: 213 Shanghai Road,Beijing
```