

例：定义描述矩形Rectangle的类。

属性：长，宽

方法：求面积double area()

求周长 double perimeter()

在主函数中，

定义 Rectangle类的对象r1， Rectangle类型的
指针pr,指向r1.

用 r1调用area(), 用pr调用perimeter

8.3 构造函数

- ▶ 变量需要进行初始化

```
int i = 10; //int i(10);
```

- ▶ 对象也需要进行初始化

- 对象的数据成员需要初始化，但不能在类体内显式初始化，而是通过**构造函数**初始化

例如：定义一个日期2018年11月21的日期

```
Date mydate(2018, 11, 21);
```

8.3.1 构造函数的定义

▶ 构造函数是一个特殊的**公有成员函数**

- 在对象创建时由系统自动调用（调用哪个）；
- 在对象被创建时使用特定的值构造对象（用什么值构造）；

具有如下性质：

1. 必须与类有完全相同的名字
2. 构造函数没有类型说明，也不允许有返回值，即使是**void**也不行。
3. 构造函数可以重载，即一个类允许定义多个参数不同的构造函数。
4. **构造函数的参数可以在声明时的参数表里给予初始值。**

8.3.1 构造函数的定义

5. 构造函数可以在类中定义，也可以在类外定义
6. 每个类必须至少有一个**构造函数**，如果没有显式地为类提供任何构造函数，则**编译器提供一个缺省的构造函数**，这个构造函数是个无参函数，它只负责对象的创建，而不做任何初始化的工作。
7. 一旦一个类定义了构造函数（有参或者无参），编译器便不再提供缺省的无参构造函数。
8. 程序中不能直接调用构造函数，在创建类的对象时系统自动调用。

(1) 无参构造函数

- ▶ 构造函数的形参列表为空，不接收参数，对数据成员进行初始化时只能使用固定值。

```
class Cuboid
{
public:
    Cuboid ();
    int volume( );
private:
    int height,width,length;
};
Cuboid::Cuboid ()
{
    height=10;
    width=20;
    length=30;
}
```

无参构造函数

```
int main()
{
    Cuboid c1;
    ...
    return 0;
}
```

(1) 无参构造函数

- ▶ 构造函数的形参列表为空，不接收参数，对数据成员进行初始化时只能使用固定值。

```
class Cuboid
{
    public:
        Cuboid ();
        int volume( );
    private:
        int height,width,length;
};
```

```
Cuboid::Cuboid ()
{
    height=10;
    width=20;
    length=30;
}
```

建立对象c1，调用构造函数c1.Cuboid()

```
int main()
{
    Cuboid c1;
    ...
    return 0;
}
```

初始化列表

- ▶ 对于数据成员的初始化除构造函数的函数体内使用赋值语句完成外，还可以使用**初始化列表**。
- ▶ **在构造函数首部完成**

```
Cuboid::Cuboid( ) :  
height(12),width(12),length(12)  
{ }
```

在构造函数的头部后面，函数体的前面，以一个冒号开始，跟着是由逗号分隔开的数据成员及其初始式，

(2) 带有参数的构造函数

在函数体内定义带有参数的构造函数的一般格式为：

构造函数名(类型1 形参1,类型2 形参2,...)

{ 函数体 }

```
Cuboid::Cuboid(int h,int w,int len)
{
    heigh=h;
    width=w;
    length=len;
}
```


(2) 带有参数的构造函数

在函数体内定义带有参数的构造函数：

构造函数名{类型1 形参1,类型2 形参2,...}

{ 函数体 }

使用参数初始化列表定义带有参数的构造函数：

构造函数名（形参列表）：成员1[参数1],成员2[参数2], ...

{ }

```
Cuboid::Cuboid(int h,int w,int len): height(h),width(w),length(len)
{ }
```

8.3.1 构造函数的定义

例8-6

```
#include <iostream>
using namespace std;
class Cuboid
{
public:
    Cuboid (int,int,int);
    int volume();
private:
    int height;
    int width;
    int length;
};
```

声明带有三个参数的构造函数

```
Cuboid::Cuboid (int h,int w,int len)
{
    height=h;
    width=w;
    length=len;
}
```

8.3.1 构造函数的定义

```
int Cuboid::volume()  
{  
    return(height*width*length);  
}  
int main()  
{  
    Cuboid cuboid1(15,45,30);  
    cout<<" cuboid1的体积为: "<< cuboid1.volume()<<endl;  
    Cuboid cuboid2(10,30,22);  
    cout<<" cuboid2的体积为: "<< cuboid2.volume()<<endl;  
    return 0;  
}
```

建立对象cuboid1, cuboid2, 调用构造函数

(3) 构造函数重载

构造函数具有相同的名字，而参数的个数或参数的类型不相同，称为**构造函数的重载**。

可以为一个类声明的构造函数数量没有限制，只要每个构造函数的形参表是唯一的。

8.3.1 构造函数的定义

```
class Cuboid
{
public:
    Cuboid ( );
    Cuboid (int h,int w,int len)
    {
        height=h; width=w; length=len;
    }
    int volume ( );
private:
    int height; int width; int length;
};

Cuboid::Cuboid ( )
{
    height=15; width=15; length=15;
}

int Cuboid::volume ( )
{
    return (height*width*length);
}
```

构造函数重载

```
int main ( )
{
    Cuboid cuboid1;
    cout<<" cuboid1的体积为:
    "<< cuboid1.volume ( )<<endl;
    Cuboid cuboid2 (20,30,45);
    cout<<" cuboid2的体积为:
    "<< cuboid2.volume ( )<<endl;
    return 0;
}
```

8.3.1 构造函数的定义

```
class Cuboid
{
public:
    Cuboid ( );
    Cuboid (int h,int w,int len)
    {
        height=h; width=w; length=len;
    }
    int volume ( );
private:
    int height; int width; int length;
};
Cuboid::Cuboid ( )
{
    height=15; width=15; length=15;
}
int Cuboid::volume ( )
{
    return (height*width*length);
}
```

建立对象cuboid1，调用**无参**构造函数

```
int main ( )
{
    Cuboid cuboid1;
    cout<<" cuboid1的体积为: "
    << cuboid1.volume ( )<<endl;
    Cuboid cuboid2 (20,30,45);
    cout<<" cuboid2的体积为: "
    << cuboid2.volume ( )<<endl;
    return 0;
}
```

8.3.1 构造函数的定义

```
class Cuboid
{
public:
    Cuboid { };
    Cuboid (int h,int w,int len)
    {
        height=h; width=w; length=len;
    }
    int volume{ };
private:
    int height; int width; int length;
};
Cuboid::Cuboid { }
{
    height=15; width=15; length=15;
}
int Cuboid::volume{ }
{ return(height*width*length);}
```

建立对象cuboid2，调用有参构造函数

```
int main{ }
{
    Cuboid cuboid1;
    cout<<" cuboid1的体积为:
    "<< cuboid1.volume{ }<<endl;
    Cuboid cuboid2 (20,30,45);
    cout<<" cuboid2的体积为:
    "<< cuboid2.volume{ }<<endl;
    return 0;
}
```

8.3.1 构造函数的定义

```
cuboid1的体积为: 3375  
cuboid2的体积为: 27000  
Press any key to continue_
```

注意： 尽管在一个类中可以包含多个构造函数，但是对于任一个对象来说，建立对象时只执行其中一个构造函数，并非每个构造函数都被执行。具体执行哪一个构造函数需要由建立对象时提供的参数来决定。

(4) 使用默认值的构造函数

一般将无参或使用缺省值参数的构造函数称为默认构造函数

8.3.1 构造函数的定义

例8-7

```
class Cuboid
{
    private:
        int height;
        int width;
        int length;
    public:
        Cuboid (int h=15,int w=15,int len=15); //声明构造函数时使用默认参数
        int volume( );
};
Cuboid::Cuboid (int h,int w,int len) //定义函数时可不指定缺省值
{
    height=h;
    width=w;
    length=len;
}
int Cuboid::volume( )
{
    return (height*width*length);
}
```

18

8.3.1 构造函数的定义

```
void main()  
{  
    Cuboid cuboid1;  
    cout<<" cuboid1的体积为: "<<  
    cuboid1.volume()<<endl;  
    Cuboid cuboid2(25);  
    cout<<" cuboid2的体积为: "<<  
        cuboid2.volume()<<endl;  
    Cuboid cuboid3(25,40);  
    cout<<" cuboid3的体积为: "<<  
        cuboid3.volume()<<endl;  
    Cuboid cuboid4(25,30,40);  
    cout<<" cuboid4的体积为: "<<  
        cuboid4.volume()<<endl;  
}
```

建立对象cuboid1，没有给出实参，
height=15,
width=15,
length=15

8.3.1 构造函数的定义

```
void main{ }  
{  
    Cuboid cuboid1;  
    cout<<" cuboid1的体积为: "<<  
    cuboid1.volume{ }<<endl;  
    Cuboid cuboid2(25);  
    cout<<" cuboid2的体积为: "<<  
        cuboid2.volume{ }<<endl;  
    Cuboid cuboid3(25,40);  
    cout<<" cuboid3的体积为: "<<  
        cuboid3.volume{ }<<endl;  
    Cuboid cuboid4(25,30,40);  
    cout<<" cuboid4的体积为: "<<  
        cuboid4.volume{ }<<endl;  
}
```

建立对象cuboid2, 只给定1个实参

height=25
width=15
length=15

8.3.1 构造函数的定义

```
void main{ }  
{  
    Cuboid cuboid1;  
    cout<<" cuboid1 的体积为: "<<  
    cuboid1.volume{ }<<endl;  
    Cuboid cuboid2(25);  
    cout<<" cuboid2的体积为: "<<  
        cuboid2.volume{ }<<endl;  
    Cuboid cuboid3(25,40);  
    cout<<" cuboid3的体积为: "<<  
        cuboid3.volume{ }<<endl;  
    Cuboid cuboid4(25,30,40);  
    cout<<" cuboid4的体积为: "<<  
        cuboid4.volume{ }<<endl;  
}
```

建立对象cuboid3，只给定2个实参，
height=25
width=40
length=15

8.3.1 构造函数的定义

```
void main{ }  
{  
    Cuboid cuboid1;  
    cout<<" cuboid1的体积为: "<<  
    cuboid1.volume{ }<<endl;  
    Cuboid cuboid2(25);  
    cout<<" cuboid2的体积为: "<<  
        cuboid2.volume{ }<<endl;  
    Cuboid cuboid3(25,40);  
    cout<<" cuboid3的体积为: "<<  
        cuboid3.volume{ }<<endl;  
    Cuboid cuboid4(25,30,40);  
    cout<<" cuboid4的体积为: "<<  
        cuboid4.volume{ }<<endl;  
}
```

建立对象cuboid4，给定3个
实参

height=25

width=30

length=40

8.3.1 构造函数的定义

```
cuboid1的体积为: 3375  
cuboid2的体积为: 5625  
cuboid3的体积为: 15000  
cuboid4的体积为: 30000  
Press any key to continue
```

- 注意：在一个类中定义了全部是缺省参数的构造函数后，若再定义重载构造函数时容易引起错误。

```
Cuboid::Cuboid (int h,int w) //重载构造函数  
{  
    height=h;  
    width=w;  
    length=100;  
}
```

8.3.2 子对象与构造函数

- ▶ **子对象**：在定义一个新类时，把一个已定义类的对象作为该类的数据成员，则这个类对象被称为**子对象**。
- ▶ 产生新定义类的对象时：
 - 必须对它的子对象进行初始化
 - 通过新类的构造函数来对它的所有子对象数据成员初始化。

子对象构造函数语法

类名::类名(参数总表): 对象1(参数子表1), 对象2(参数子表2),, 对象n(参数子表n)
{ }

```
class Rectangle //定义矩形类
{
private:
    int Width,Length; //宽度、长度
public:
    Rectangle(int w,int len) //定义带参构造函数
    {
        Width=w;
        Length=len;
    }
    int Area() //计算面积
    {
        return (Width*Length);
    }
};
```

8.3.2 子对象与构造函数

```
class Cuboid //定义长方体类
```

```
{
```

```
private:
```

```
    int Height; //高度
```

```
    Rectangle r;
```

使用已定义的类Rectangle
的对象作为成员

```
public:
```

```
    Cuboid(int w,int len,int h):r(w,len) //用参数初始化表对r进行初始化
```

```
    { Height=h; }
```

```
    int Volume()
```

```
    { return (Height*r.Area()); }
```

```
};
```

```
void main()
```

```
{
```

```
    Cuboid c1(10,20,100); //定义对象c1
```

```
    cout<<"长方体c1的体积是: "<<c1.Volume()<<endl;
```

```
}
```

8.3.2 子对象与构造函数

```
class Cuboid //定义长方体类
```

```
{
```

```
private:
```

```
    int Height; //高度
```

```
    Rectangle r;
```

```
public:
```

```
    Cuboid(int w,int len,int h):r(w,len)
```

```
    { Height=h; }
```

```
    int Volume()
```

```
    { return (Height*r.Area()); }
```

```
};
```

```
void main()
```

```
{
```

```
    Cuboid c1(10,20,100); //定义对象c1
```

```
    cout<<"长方体c1的体积是: "<<c1.Volume()<<endl;
```

```
}
```

用参数初始化表对r进行初始化

8.3.2 子对象与构造函数

```
class Cuboid //定义长方体类
{
private:
    int Height; //高度
    Rectangle r;
public:
    Cuboid(int w,int len,int h):r(w,len)
    { Height=h; }
    int Volume()
    { return (Height*r.Area()); }
};

void main()
{
    Cuboid c1(10,20,100); //定义长方体c1
    cout<<"长方体c1的体积是: "<<c1.Volume()<<endl;
}
```

在定义Cuboid对象c1时自动调用其构造函数进行初始化。初始化过程分成两步进行：
(1) 实参10、20通过形参w、len赋给r(w,len)，然后调用Rectangle的构造函数完成对象r的初始化。
(2) 实参100通过形参h赋给Height，完成对类Cuboid的数据成员Height的初始化。

```
长方体c1的体积是: 20000
Press any key to continue
```

说明：

- ▶ 子对象可以有多个，它们构造函数的调用顺序取决于这些子对象成员在类中的说明顺序，与它们在成员初始化表的位置无关。
- ▶ 定义类的对象时，先调用各个子对象成员的构造函数，初始化相应的子对象成员，然后再执行类的构造函数，初始化类中其他成员。

8.3.3拷贝构造函数

► 编程中自然地存在如下的需求：

- 复制 / 克隆：用一个对象初始化另一个对象

内置类型的变量的复制很简单，例如：

```
int a = 10; int b = a; //从a复制出b
```



a的数据值：10

a的地址值：0012FF6CH



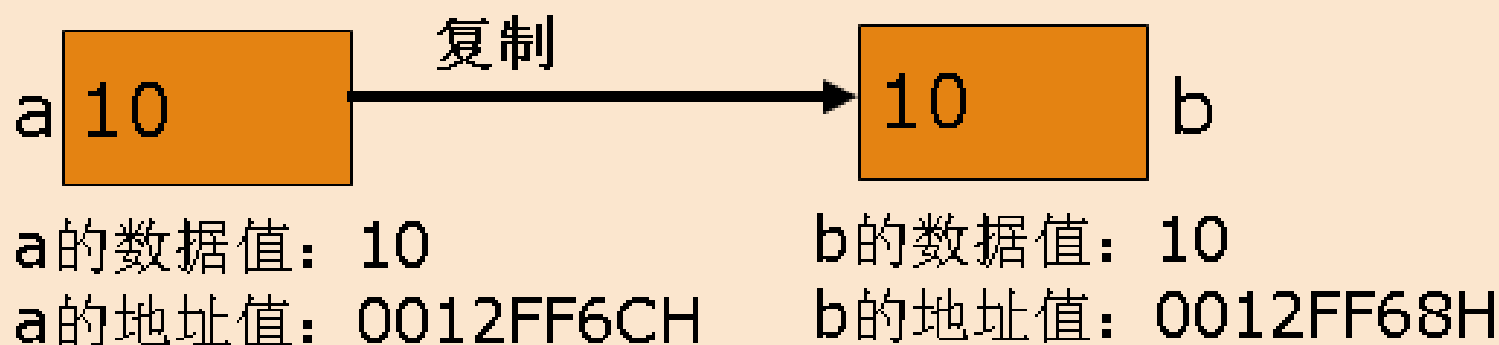
b的数据值：

b的地址值：

8.3.3拷贝构造函数

- ▶ 编程中自然地存在如下的需求：
 - 复制/克隆：用一个对象初始化另一个对象
- 内置类型的变量的复制很简单，例如：

```
int a = 10; int b = a; //从a复制出b
```



拷贝构造函数

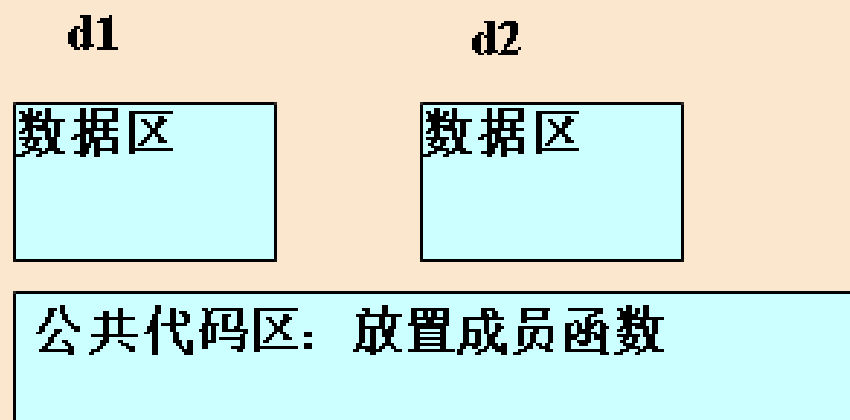
- ▶ 如何实现类对象的复制呢？

```
Date d1(2017, 11, 12);
```

```
Date d1 = d2; //用d1初始化d2
```

1. 成员函数无需复制;
2. 数据成员必须复制。

问题：但如何复制数据成员？



- ▶ 拷贝构造函数是一种构造函数
- ▶ 具有一个参数：本类对象的引用

拷贝构造函数的形式：

类名 (类名 &对象名);

- ▶ C++规定，拷贝构造函数的名称必须与类名称一致，函数的形参是本类型的一个引用变量，且必须是引用。

拷贝构造函数的注意事项

1. 当一个类没有自定义的拷贝构造函数的时候，系统会自动提供一个默认的拷贝构造函数来完成拷贝工作。
2. 一个对象向该类的另一个对象做拷贝，通过依次拷贝每个**非静态数据成员**来实现。
3. 可以通过提供特殊的拷贝构造函数来改变缺省的行为。
4. 如果定义了拷贝构造函数，则在用一个对象初始化该类另一个对象时它就会被调用。

- ▶ 默认构造函数对数据成员执行**浅拷贝**

浅拷贝：简单的位复制，等效于用源对象的每个数据成员的值，初始化目标对象的对应数据成员，又称**默认按成员初始化**。

自定义拷贝构造函数实现**深拷贝**

8.3.3拷贝构造函数

例8-10

```
class Cuboid  
{  
private:
```

```
    int heigh;  
    int width;  
    int length;
```

```
public:
```

```
    Cuboid(int,int,int);
```

```
    Cuboid(Cuboid &C)
```

```
{
```

```
    cout<<"拷贝构造函数被调用"<<endl;
```

```
    heigh=C.heigh+10;
```

```
    width=C.width+10;
```

```
    length=C.length+10;
```

```
}
```

```
    int volume();
```

```
};
```

定义Cuboid类的拷贝构造函数

8.3.3拷贝构造函数

```
void main()
{
    Cuboid cuboid1(10,10,10);
    cout<<"cuboid 1的体积为:
"<<cuboid1.volume()<<endl;
    Cuboid cuboid2(cuboid1);
    cout<<"cuboid2的体积为:
"<<cuboid2.volume()<<endl;
}
```

调用拷贝构造函数实现
初始化

- ▶ 对象cuboid2初始化的核心语句就是通过拷贝构造函数Cuboid(Cuboid &C)内的

```
heigh = C.heigh + 10;  
width = C.width + 10;  
length = C.length + 10;
```

- ▶ 在深拷贝中，需要对数据成员进行初始化，如果没有初始化，则新构造的对象的数据成员将是随机值。
- ▶ 如果去掉这些初始化代码，那么 cuboid2对象的成员将得到一个未知的随机值。

何时调用 拷贝构造函数 (1)

(1) 当用一个已经初始化过的对象去初始化同类另一个对象时，拷贝构造函数被调用。

```
Cuboid cuboid2(cuboid1);
```



```
Cuboid cuboid2=cuboid1;
```


何时调用 拷贝构造函数 (2)

(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的拷贝构造函数将被调用

```
void f( A a )
```

```
{
```

```
    a.x = 1;
```

```
};
```

```
A aObj;
```

```
f ( aObj) ;
```

// A的拷贝构造函数被调用，生成形参传入函数

8.3.3拷贝构造函数

```
int show(Cuboid c)
{
    return c.volume();
}
```

```
void main()
{
    Cuboid cuboid1(10,10,10);
    cout<<"cuboid1的体积为：
"<<cuboid1.volume()<<endl;
    cout<<"使用show函数输出长方体的体
积:"<<show(cuboid1)<<endl;
}
```

拷贝构造函数被调用

何时调用 拷贝构造函数（3）

(3) 函数的返回值是类A的对象时，则函数返回时，A的拷贝构造函数被调用

```
A f()  
{  
    A a;  
    return a; // 此时A的拷贝构造函数被调用,即调用A(a);  
}  
int main()  
{  
    A b;  
    b = f();  
    return 0;  
}
```

8.3.3拷贝构造函数

```
Cuboid callback()  
{  
    Cuboid cuboid1(10,10,10);  
    return cuboid1;  
}
```

拷贝构造函数被调用

```
void main()  
{  
    Cuboid cuboid2=callback();  
    cout<<"使用callback函数输出长方体的体  
积:"<<cuboid2.volume()<<endl;  
}
```

8.4 析构函数

8.4.1 析构函数的定义

析构(Destructor)函数是当对象脱离其作用域时（例如对象所在的函数已调用完毕），系统自动执行。

```
class 类名
{
    public:
        ~类名() //析构函数
        {
            //函数体
        }
};
```

析构函数说明 (1)

- (1) 在C++中，析构函数名应与类名相同，只是在函数名前面加一个位取反符“~”，以区别于构造函数。
- (2) 析构函数不能带任何参数，也没有返回值（void类型也不行）。一个类最多只能有一个析构函数，不能重载。
- (3) 如果用户没有编写析构函数，编译系统会自动生成一个缺省的析构函数，它也不进行任何操作。

析构函数说明 (2)

(4) 析构函数在撤销对象时由系统自动调用，其作用是在撤销对象前做好结束工作。

(5) 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。

(6) 析构函数在对象作为函数返回值返回后被调用。函数调用过程中，在临时对象生成的时候会有构造函数被调用，临时对象消亡导致析构函数调用。

8.4.1 析构函数的定义

例8-11

```
class CStudent //声明CStudent类
{
private:
int Snum;    string Sname;
public:
    CStudent(int n,string nam) //定义构造函数
    {
        Snum=n;
        Sname=nam;
        cout<<"Constructor called."<<endl; //输出有关信息
    }
    ~CStudent() //定义析构函数
    {
        cout<<Sname<<" Destructor called."<<endl;
    }
    void display() //定义成员函数,输出有关信息
    {
        cout<<"学号:"<<Snum<<endl;
        cout<<"姓名:"<<Sname<<endl<<endl;
    }
};
```


8.4.1 析构函数的定义

```
void main( )  
{  
    CStudent stud1(10010,"Zhang_San");  
    stud1.display( );           //输出stud1的数据  
    CStudent stud2(10011,"Li_Si"); //定义对象stud2  
    stud2.display( );           //输出stud2的数据  
}
```

先执行哪个对象的构造函数？

先执行哪个对象的析构函数？

```
Constructor called.  
学号: 10010  
姓名: Zhang_San  
  
Constructor called.  
学号: 10011  
姓名: Li_Si  
  
Li_Si Destructer called.  
Zhang_San Destructer called.  
Press any key to continue
```

8.4.1 析构函数的定义

例8-12对象数组析构函数

```
class CSample
{
    public:
    ~CSample()
    {
        cout<< "destructor called" << endl;
    }
};

void main ()
{
    //对象数组有两个元素，程序结束时每个对象元素都将调用析构函数
    CSample CArray[2];
    cout << "This Program End." << endl;
}
```

```
This Program End.
destructor called
destructor called
Press any key to continue_
```

8.4.2 构造函数和析构函数的调用顺序

► 在一般情况下：

- 局部定义的对象，在程序执行到定义对象的地方时调用构造函数，函数结束时调用析构函数。
- 调用析构函数的次序正好与调用构造函数的次序相反

先构造的，先析构

特殊情况:

| | 调用构造函数 | 调用析构函数 |
|----------------|----------------|-----------------------------|
| 全局定义的对象 | 在程序开始执行时调用构造函数 | 程序结束或调用exit()函数终止程序时才调用析构函数 |
| static定义的局部对象 | 在首次到达对象定义位置 | 在程序结束时调用析构函数 |
| 用new运算符动态生成的对象 | 产生对象时 | 只有用delete释放对象时 |

8.4.2 构造函数和析构函数的调用顺序

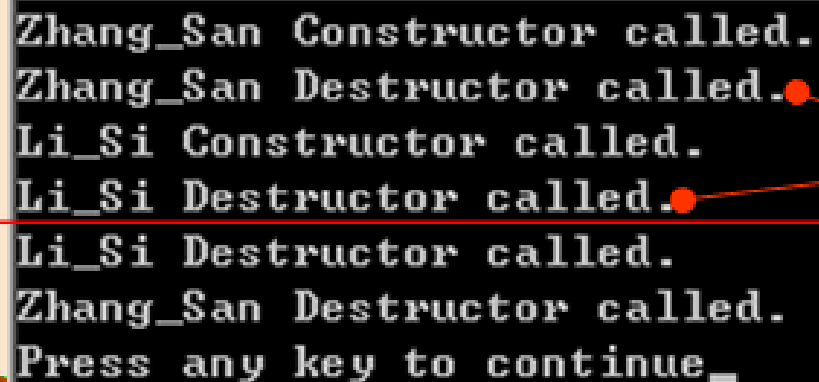
例构造函数与析构函数执行顺序

```
class CStudent
{
    private:
        string SName;
        float Score;
    public:
        CStudent (string name, float sc);
        ~CStudent{ };
};

CStudent::CStudent(string name, float sc)
{
    SName=name;
    Score=sc;
    cout<<SName<<" Constructor called."<<endl;
}
```

8.4.2 构造函数和析构函数的调用顺序

```
CStudent::~CStudent()  
{  
    cout<<SName<<" Destructor called."<<endl;  
}  
void main( )  
{  
    CStudent stud[2]={CStudent("Zhang_San",90),CStudent("Li_Si",80)};  
}
```

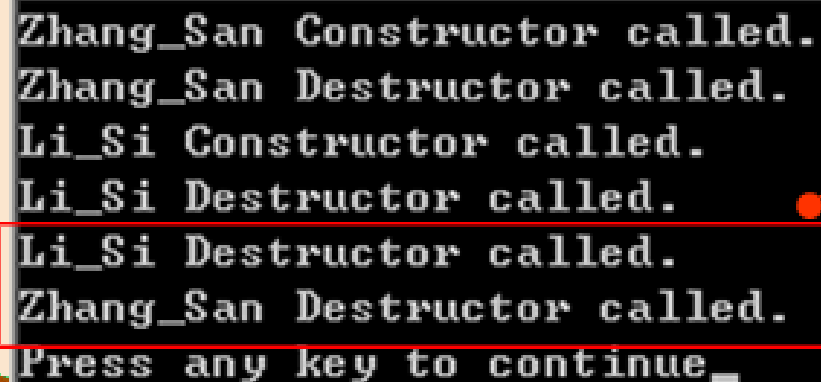


```
Zhang_San Constructor called.  
Zhang_San Destructor called.  
Li_Si Constructor called.  
Li_Si Destructor called.  
Li_Si Destructor called.  
Zhang_San Destructor called.  
Press any key to continue_
```

1. 实例化了2个临时对象，调用构造函数
2. 待使用默认的拷贝构造函数赋值完成后
3. 系统将自动调用析构函数将其析构掉。

8.4.2 构造函数和析构函数的调用顺序

```
CStudent::~CStudent()  
{  
    cout<<SName<<" Destructor called."<<endl;  
}  
void main( )  
{  
    CStudent  
    stud[2]={CStudent("Zhang_San",90),CStudent("Li_Si",80)};  
}
```



```
Zhang_San Constructor called.  
Zhang_San Destructor called.  
Li_Si Constructor called.  
Li_Si Destructor called.  
Li_Si Destructor called.  
Zhang_San Destructor called.  
Press any key to continue_
```

程序运行结束后，按调用默认构造函数的逆序stud[1]、stud[0]调用析构函数

注意：主函数语句如下所写后的运行结果。

```
void main()
```

```
{
```

```
    CStudentS1("Zhang_San",90);
```

```
    CStudentS2=CStudent("Li_Si",80);
```

```
}
```

```
Zhang_San Constructor called.  
Li_Si Constructor called.  
Li_Si Destructor called.  
Li_Si Destructor called.  
Zhang_San Destructor called.  
Press any key to continue
```


8.4.3 对象的动态建立与释放

- ▶ 可以用`new`运算符动态地建立对象。
 - 首先，`new`运算符为类对象分配内存空间
 - 然后，自动调用构造函数初始化对象的数据成员
 - 最后，将该对象的起始地址返回给指针变量。
- ▶ 动态分配的对象内存空间可以用`delete`运算符释放。
 - 只有在使用`delete`释放对象时，系统才会调用析构函数。
 - 如果不使用`delete`运算符来撤销动态生成的对象，程序结束时将不会调用析构函数。

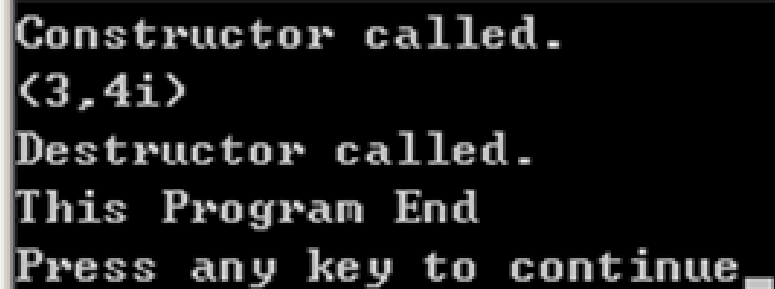
8.4.3 对象的动态建立与释放

例8-13 new运算符和delete运算符

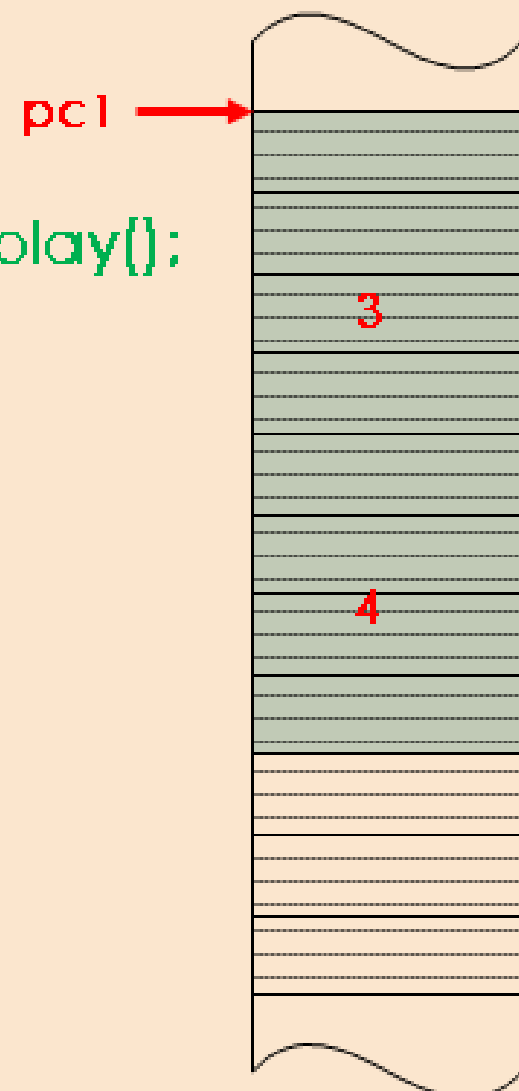
```
class Complex
{
public:
    Complex(double r,double i)
    {
        real=r;    imag=i;
        cout<<"Constructor called."<<endl;
    }
    ~Complex()
    {
        cout<<"Destructor called."<<endl;
    }
    void display()
    { cout<<"("<<real<<","<<imag<<"i)"<<endl; }
private:
    double real;    double imag;
};
```

8.4.3 对象的动态建立与释放

```
int main()  
{  
    Complex *pc1=new Complex (3,4);  
    pc1->display(); //或者    (*pc1).display();  
    delete pc1;  
    cout<<"This Program End"<<endl;  
    return 0;  
}
```

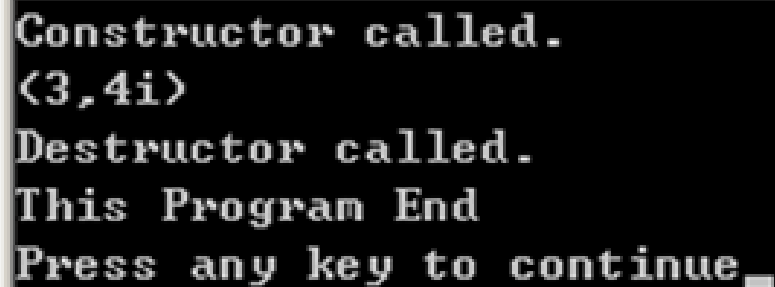


```
Constructor called.  
(3,4i)  
Destructor called.  
This Program End  
Press any key to continue_
```

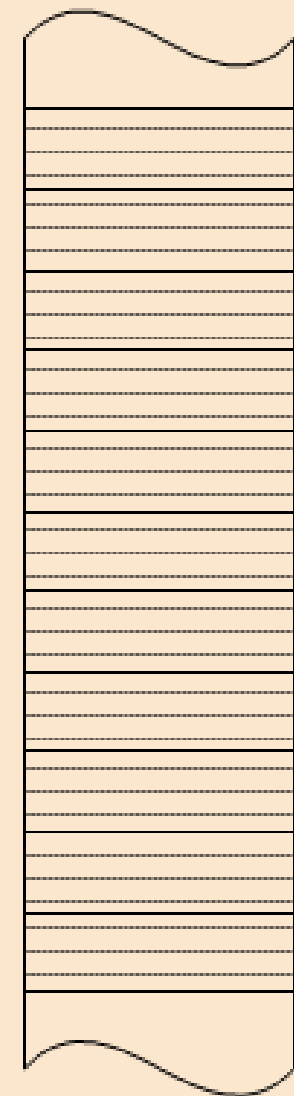


8.4.3 对象的动态建立与释放

```
int main()  
{  
    Complex *pc1=new Complex (3,4);  
    pc1->display(); //或者    (*pc1).display();  
    delete pc1;  
    cout<<"This Program End"<<endl;  
    return 0;  
}
```



```
Constructor called.  
(3,4i)  
Destructor called.  
This Program End  
Press any key to continue_
```



pc1