在 C++中,对于普通类型的数据类型来说,它们之间的复制是很简单的,例如: int a= 100; int b= a;利用一个简单的赋值语句就可以实现普通数据的复制。而类与普通数据类型不同,类对象内部结构一般较为复杂,存在各种成员变量或者包含指向其他资源的指针。如

```
class MyExample
{
    public:
        MyExample( ) { pBuffer=NULL; nSize=0; }
        ~MyExample( ) { delete pBuffer; }
        void Init(int n) { pBuffer= new char[n]; nSize=n; }
        void Out( ){ cout<<nSize<<pBuffer<<endl; }
    private:
        char *pBuffer; //类的对象中包含指针,指向动态分配的内存资源
        int nSize;
} ;
```

这个类的主要特点是包含指向其他资源的指针, pBuffer 指向堆中分配的一段内存空间。对于一个结构复杂的类来说,如果要实现它的复制,这就要谈到拷贝构造函数。

C++语法中,对于一个对象在定义时都要进行初始化,所以一般在对象创建时都会自动调用构造函数。除了提供缺省形式的构造函数外,还规范了另一种特殊的构造函数: 拷贝构造函数,如果类中定义了拷贝构造函数,这对象建立时,调用的将是拷贝构造函数,在拷贝构造函数中,可以根据传入的变量,复制指针所指向的资源。提供了拷贝构造函数后的 MyExample 类定义为 :

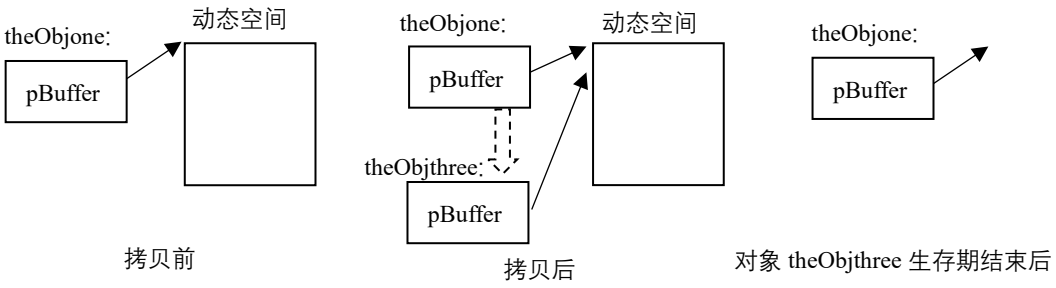
```
MyExample:: MyExample ( const MyExample &RightSides)
{
    nSize= RightSides.nSize; //复制常规成员
    pBuffer= new char[nSize]; //复制指针指向的内容
    memcpy ( pBuffer , RightSides.pBuffer , nSize * sizeof( char ) ) ;
}
```

这样, 定义新对象, 并用已有对象初始化新对象时,MyExample(const MyExample &RightSides)将被调用,而已有对象用别名 RightSides 传给构造函数,以用来作复制。原则上,应该为所有包含动态分配成员的类都提供拷贝构造函数。

如果在类中没有显式地声明一个拷贝构造函数,那么,编译器将会自动生成一个默认的拷贝构造函数,该构造函数完成对象之间的位拷贝。位拷贝又称浅拷贝,浅拷贝会带来数据安全方面的隐患。

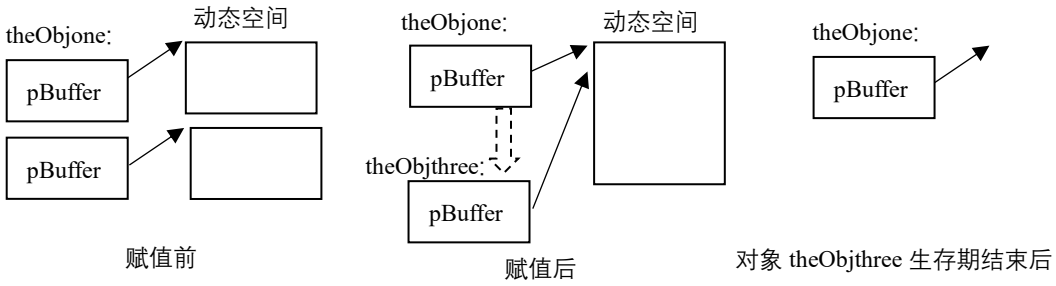
```
int main()
{
    MyExample theObjone;
    theObjone.Init(4) ;
    MyExample theObjthree(theObjone);
    theObjone.Out();
    return 0;
}
```

从下图可以看出，这里只是将对象theObjone中字符指针的首地址简单赋值给了对象theObjthree的指针成员，对象theObjthree根本没有构造自己的字符数组空间，结果造成对象theObjone和theObjthree共同使用同一块内存空间存放字符数组元素。所以当对象theObjthree生存期结束，调用析构函数时，对象theObjone中指针成员对应的内存空间也被释放掉了，这种情况也被称为指针悬挂。所以对象theObjone在访问这块内存时，就会出现运行错误。



```
int main()
{
	MyExample theObjone;
	theObjone.Init(4) ;
	MyExample theObjthree;
	theObjthree.Init(6) ;
	theObjthree=theObjone;
	theObjthree.Out();
	return 0;
}
```

再下以图为例，看一下默认的赋值运算符函数实现的功能。这里同样只是将对象theObjone中字符指针的首地址简单赋值给了对象theObjthree的指针成员，造成对象theObjone和theObjthree共同使用同一块内存空间存放字符数组，所以也出现了上述的运行结果。上述两种赋值方式都是仅仅将对象的数据成员进行了对应的拷贝。这种拷贝方式称为浅拷贝。



为了避免上述结果，对MyExample类进行修改。加进自定义的拷贝构造函数和赋值运算符重载函数，从例子注释中可以看到，二者都有内存空间开辟和字符串拷贝的语句，亦即在进行赋值之前，为指针类型的数据成员另辟了一个独立的内存空间，实现真正内容上的拷贝。这种拷贝称为深拷贝。深拷贝方式有效的解决了上述的指针悬挂问题。

```
MyExample & MyExample::operator=( const MyExample &RightSides)//赋值操作符重载
{
	nSize= RightSides.nSize; //复制常规成员
	char *temp= new char[nSize]; //复制指针指向的内容
	memcpy(temp, RightSides.pBuffer, nSize*sizeof( char ) ) ;
	delete []pBuffer; //删除原指针指向内容(将删除操作放在后面,避免 X= X 特殊情况下,内容的丢失)
	pBuffer= temp; //建立新指向
	return * this;
}

MyExample::MyExample ( const MyExample &RightSides)//拷贝构造函数使用赋值运算符重载的代码
{
	pBuffer= NULL;
	*this= RightSides; //调用重载后的“ = ”
}
```

要实现正确的拷贝也就是深拷贝,必须编写拷贝构造函数。如上例,就是利用赋值操作符重载解决数据丢失来实现完整的类的复制。自定义拷贝构造函数是一种良好的编程风格,它可以阻止编译器形成默认的拷贝构造函数,提高源码效率。