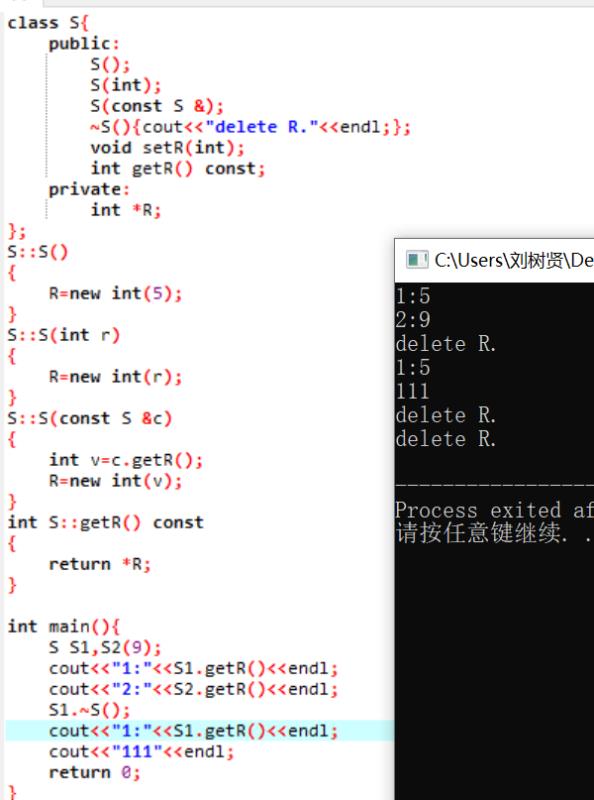
**讨论面向对象编程和指针的几个问题**

**ex6-20.cpp 这个程序存在什么问题？**

**析构函数不是应该能销毁对象吗？**



那这个不是释放不了吗 你在使用以后应该还能访问吧？

这里要**如何才能把对象释放**？

首先澄清一下概念。构造函数和析构函数只是系统留给编程者在对象构建和析构时，你能插入代码的函数。

而程序中定义的对象，是由系统自己释放的不是你的析构函数。这就是你调用析构函数并不能释放对象原因。

就是说析构函数跟释放对象是两个步骤吗？对，析构函数的执行在释放对象之前。

我百度了一下好像默认的析构函数也是空的?

你想想默认为空的析构函数怎么释放对象？

所以该题的两个对象的释放不是由你的析构函数释放的，是系统自动释放的。但对程序中不能自动释放的内容，就需要自己在析构函数中释放了。明白了吗？

析构函数的机制是在这个时机，给你个执行代码的机会，把需要自己释放的内容放到析构函数中释放。

**但这个程序中，有什么错误呢？**

感觉他new的两个动态内存没释放。

对。 对于程序员自己用new申请的内存，系统是不会自动释放的，所以这时就需要程序员在析构函数中用delete释放掉，否则就造成了内存泄漏。

即，~SimpleCircle() { delete itsRadius;}

**再一个问题，系统为什么不能自动释放指针所指的内存呢？**因为在编译时是无法知道运行时指针所指申请的内存大小。

另因为指针本身是变量，在运行时也可以动态变动，所以编译器无法检查其所指是否超界。所以指针的灵活性，也伴随着隐藏的危险性。同时指针的概念也破坏了面向对象封装的思想。(因为指针可以轻松指向它不该指的地方)。

**该题中的复制构造函数如果没有**，就会造成对象在复制时的简单指针复制(即浅拷贝)，后果是两个对象中的指针变量指向了同一内存地址。

因为复制构造函数只在创建新对象时调用，不用考虑原指针所指单元的释放问题。但如果是其他普通函数或赋值运算符=重载来完成对象中指针成员的复制问题，就应考虑原指针所指单元的释放问题！之后再接受形参传来的新值，否则还会造成内存泄漏。

加上复制构造函数的（深拷贝）代码后，不同对象就有各自独立的内存空间。如果你的程序里没有指针成员，系统会自动完成对象成员的复制（浅拷贝）和释放。

**一般写代码都要用深拷贝吗?**  没有指针成员的程序，也就不必写什么深拷贝的代码

你以为这个题有了复制构造函数里的代码就安全了吗？

**ex6-20\_2.cpp中的问题**

即刚才那个题，经过在析构函数里加了delete，在复制构造函数里加了代码实现深拷贝。但在对象赋值时又出现了两个对象指针指向同一地方（浅拷贝）的现象。

t4和CircleTwo地址一样；= 执行的仍然是浅拷贝，并没有执行复制构造函数。

这个问题是一般程序员看不到的。它的问题在于不同的对象操纵同样的内存单元；当对象撤销时，会导致在多个对象各自的析构函数里执行delete同样的地址（非法）。

这就是用指针成员的危险性表现。而系统程序大多数都是用C++编写的，里面暗礁重重。

**ex6-20\_3.cpp 分析结果**

在定义处的=与在程序中单独的赋值=是不同的。

所以该题还需要对赋值运算符进行重载（实现深拷贝）才能解决问题。

运算符重载在后面的课程里要讲。

**总结：**

通过这个题的分析，同学们需要真正理解：

（1）构造函数、析构函数的本质

（2）动态内存的释放

（3）指针的灵活性与伴随的危险性

（4）对象中指针类成员的特殊处理