让程序在运行时决定内存分配，而不是在编译时。C++用new和delete来动态控制内存，在类中使用这些运算，析构函数必不可少。

静态类成员是所有类的实例共享的参数，不能再类声明中初始化静态成员，除非是const整数类型或枚举型。

static int num\_strings; //声明

int StingBad::num\_strings = 0; //初始化，不能在头文件中

在构造函数中使用new分配的内存，必须在析构函数中使用delete来释放。

P429 StringBad的糟糕示例，即P434的分析re。

特殊成员函数

如果没有定义的话，C++自动提供下列成员函数：默认构造函数；默认析构函数；复制构造函数；赋值运算符和地址运算符。C++11还提供移动构造函数和移动赋值函数。

隐式地址运算符返回调用对象的地址，即this指针的值。默认析构函数不执行任何操作。

默认的默认构造函数是不接受任何参数，也不执行任何操作的构造函数。若定义了构造函数，但希望在创建对象时不显式地进行初始化，则须显式地定义默认构造函数。只能有一个默认构造函数。

复制构造函数用于将一个对象复制到新创建的对象中。它用于初始化过程中，而不是常规的赋值过程中。复制构造函数的原型：

Class\_name(const Class\_name &);

新建一个对象并将其初始化为同类现有对象时，复制构造函数将被调用。每当程序生成对象副本时，编译器都将使用复制构造函数，如函数按值传递对象或函数返回对象时。默认复制构造函数逐个复制非静态成员(浅复制，仅复制其值)，若有字符串，则只复制了指向字符串的指针，容易出问题。解决方案是定义一个显式的复制构造函数，进行深度复制，即复制字符串，并返回副本的地址。*只要涉及到指针就要加倍小心*。如果类中包含了使用new初始化的指针成员，应当应以一个复制构造函数，以复制指向的数据，而不是指针，这就叫深度复制。

赋值运算符。C++允许类对象赋值，这是通过自动为类重载赋值运算符实现的，其原型如下：

Class\_name & Class\_name::operator=(const Class\_name &);

将已有的对象赋给另一个对象时，将使用重载的赋值运算符。赋值运算符的隐式实现也是对成员逐个复制，对指针成员，易引起问题。解决方法是提供赋值运算符，进行深度复制，其实现与复制构造函数类似，但有一些差别：1.由于目标对象可能引用了以前分配的数据，所以函数应使用delete[]来释放这些数据；2.函数应避免将对象赋给自身，否则，给对象重新赋值前，释放内存操作可能删除对象的内容；3.函数返回一个指向调用对象的引用，可以像常规赋值操作那样进行连续赋值。示例：

StringBad & StringBad::operator=(const StringBad & st)

{

if (this==&st) // 检查是否为自我复制

return \*this;

delete [] str; // 释放旧数据

len = st.len;

str = new char [len+1];

std::strcpy(str,st.str);

return \*this;

}

一般用NULL或0表示空指针，在C++11中关键字nullptr表示空指针，如：str=nullptr;

12.2re

在构造函数中使用new时的注意事项：

1.如果构造函数中用new来初始化指针成员，则在析构函数中必须用delete

2.new和delete必须兼容，new对应delete，new[]对应delete[]

3.若有多个构造函数，必须以相同的方式使用new(要么都有[]，要么都没有)，但可以用空指针，因为delete都可以用于空指针

4.应定义一个复制构造函数，实现深度复制

5.应定义一个赋值运算符，实现深度复制

返回对象的说明：可以返回指向对象的引用，指向对象的const引用或const对象。

下列情况下，析构函数将被调用

如果对象是动态变量，当执行完定义该对象的程序块时，将调用该对象的析构函数；如果对象是静态变量，则在程序结束时调用对象的析构函数。如果对象是new创建的，则仅当使用delete删除对象时，调用析构函数。

定位new运算符P456

12.7，示例