通常情况下，编译器会为类定义一个默认的拷贝构造函数，但当类中有资源时（堆内存，指针等），则需要自己定义一个拷贝构造函数。

拷贝构造函数：

只有一个形参，且该形参是对本类类型对象的引用。

class Student

{

Student(Student& s); // 拷贝构造函数

}

如果显式的定义了一个拷贝构造函数，则至少需要再定义一个其他的构造函数，因为这时无法再使用默认构造函数了。

拷贝构造函数的用途：

1. 根据另一个同类型的对象显式或隐式初始化一个对象。
2. 复制一个对象，将它作为实参传给一个函数。
3. 从函数返回时复制一个对象。
4. 初始化顺序容器中的元素。
5. 根据元素初始化式列表初始化数组元素。

C++支持两种初始化形式：直接初始化和复制初始化

int a(10); // 直接初始化

int a = 10; // 复制初始化

例：程序class\_test8

#ifndef STUDENT\_H

#define STUDENT\_H

class Student

{

public:

Student(char\* name\_ptr = "no name", int ssID = 0);

Student(Student& s); // 拷贝构造函数，其形参必须为引用，而不能直接为对象，否则编译会报错，原因：如果直接为对象，这是值传递，会再次调用拷贝构造函数，从而引起无穷递归。

~Student();

protected:

char name\_[40];

int id\_;

};

#endif // STUDENT\_H

#include <iostream>

#include <string>

#include "student.h"

using namespace std;

Student::Student(char\* name\_ptr,int ssID)

{

strcpy(name\_, name\_ptr);

cout << "constructing new student " << name\_ << endl;

}

Student::Student(Student& s)

//拷贝构造函数，必须用引用，不能直接用Student(student s)

//因为不能像基本数据类型那样直接传递对象的值。

{

cout << "constructing copy of " << s.name\_ << endl;

strcpy(name\_, "copy of ");

strcat(name\_, s.name\_);

id\_ = s.id\_;

}

Student::~Student()

{

cout<< "destructing " << name\_ <<endl;

}

void func(Student s);

int main(int argc, char\* argv[])

{

Student randy("Randy", 1234);

func(randy); // 调用拷贝构造函数

std::cout << "retruned from func()\n";

Student jenny = randy; // 调用拷贝构造函数

return 0;

}

void func(Student s)

{

std::cout << "In function func()\n";

}

输出为：

constructing new student Randy

constructing copy of Randy

In function func()

destructing copy of Randy //函数结束，发生析构，析构的是func中s对象

retruned from func()

constructing copy of Randy

destructing copy of Randy

destructing Randy

可以看出析构函数的调用与构造函数正好相反。

用一个类对象区构造一个新的对象时，如果没有自定义的拷贝构造函数，C++会调用默认的拷贝构造函数，但如果构造函数中有资源（堆内存）时，会出问题，因为默认的拷贝构造函数只会简单的拷贝资源，而不会分配资源，这时会出现：两个对象都拥有同一个资源的局面，在析构时会产生错误。这时需要自定义拷贝构造函数做深拷贝。如下例所示：

例：程序class\_test9

#ifndef PERSON\_H

#define PERSON\_H

class Person

{

public:

Person(char\* name\_ptr);

Person(Person& p);//拷贝构造函数

~Person();

protected:

char\* name\_ptr\_;

};

#endif // PERSON\_H

#include <iostream>

#include <string>

#include "person.h"

using namespace std;

Person::Person(char\* name\_ptr)

{

cout <<"constructing " << name\_ptr << endl;

name\_ptr\_ = new char[strlen(name\_ptr) + 1];

if (name\_ptr\_ != 0)

{

strcpy(name\_ptr\_, name\_ptr);

}

}

Person::Person(Person& p)

{

/\*

\*这个拷贝构造函数是必须的，原因是析构函数中使用deltete将堆空间返回给系统

\*如果没有拷贝构造函数，而使用默认的拷贝构造函数，那么后面的程序中p1,p2将拥有完全相同的堆空间，

在析构时，析构了p2后，堆空间已经返回给系统，再析构p1时，执行delete pName;就会出错

\*/

cout<< "Copying " << p.name\_ptr\_ << " into its own block\n";

name\_ptr\_ = new char[strlen(p.name\_ptr\_) + 1];

if (name\_ptr\_ != 0)

{

strcpy(name\_ptr\_,p.name\_ptr\_);

}

}

Person::~Person()

{

cout << "destructing " << name\_ptr\_ << endl;

delete [] name\_ptr\_;

name\_ptr\_ = NULL;

}

#include <iostream>

#include <string>

#include "person.h"

using namespace std;

int main(int argc, char\* argv[])

{

Person p1("Randy");

Person p2 = p1;//即Person p2(p1)

return 0;

}

堆内存并不是唯一一个需要拷贝构造函数的资源，打开文件，占有硬件资源都需要拷贝构造函数，很好的经验：

如果类需要析构函数来析构资源，则它也需要一个拷贝构造函数。

禁止拷贝：

将拷贝构造函数定义为私有的，可以禁止对该类类型的对象进行复制。

如果需要进一步禁止类的友元和成员进行拷贝，可以声明一个private的拷贝构造函数，但却不定义它。

经验：

绝大多数情况下，类应该定义默认构造函数和拷贝构造函数。

类如果需要拷贝构造函数，那也需要重载赋值操作符。