

实验九 复制构造函数

一、 问题描述

1. 实验目的：

掌握复制构造函数的若干基本概念和特性，并能够应用于程序编写。

掌握验证性实验的基本方法和过程(认知、实验、总结)。

2. 实验内容：

分别编写一段测试代码来回答任务书中的相关问题（每一个问题，用一个工程文件，同时需要记录相应的调试过程），具体问题请参考“实验任务说明09.doc”；

调试的过程；（动态调试的相关截图，比如 设置断点、查看当前变量值等）；

编译出来的可执行程序单独放在一个目录下（bin/exe/debug目录下，同时附上输入数据说明和输出结果）

二、 实验过程

1. 选择题

1.1、假定MyClass为一个类，则该类的拷贝构造函数的声明语句为（C）

A MyClass&(MyClass x); B MyClass(Myclass x);

C MyClass (MyClass &x); D MyClass(Myclass *x);

答：拷贝构造函数的形参不限制为const，但是必须是一个引用，以传址方式传递参数，否则导致拷贝构造函数无穷的递归下去，指针也不行，本质还是传值。

2. 程序阅读题，分析并讨论结果

类声明核心代码

```
#include <iostream>
using namespace std;
class Base {
    public: Base(int a) : val(a) {    }
    private: int val;
};
class A {
    public:
        A(int v) : p(v), b(v) {    }
        void print_val() {cout << "hello:" << p <<
endl;}
    private:
        int p;
        Base b;
};
```

主函数核心内容:

```
int pp = 45;
A b(pp),c=b;

b.print_val();
c.print_val();
```

1、分析程序执行过程，重点分析类成员的形成
答：运行过程中，定义了对象b，c。b中的成员通过传入参数pp即45进行赋值，先调用Base类的构造函数，再调用A类的构造函数。

2、测试并给出程序中构造函数的执行次数。若是只执行一次构造函数，但是程序中定义了两个对象，这是什么原因（是什么在起作用？）？
答：先调用Base类的构造函数，再调用A类的构造函数，但是只是在定义b对象时调用。这是复制构造函数的作用，它用类的对象b去初始化另一个对象c。

3、对象c是否有数据？是如何获得数据的？
答：c对象有数据，其变量p、Base b的成员val的值同样是45，是通过b对象进行的初始化，使用了复制构造函数。

4、A b(pp),c=b;修改成：A b(pp),c(b);程序的结果会有什么变化？
答：程序运行结果相同，c(b)也是调用复制构造函数

类声明核心代码

```
#include <iostream>
using namespace std;
class Base {
    public: Base(int a) : val(a) {    }
    private: int val;
};
class A {
    public:
        A(int v) : p(v) {    }
        void print_val() {cout << "hello:" << p
<< endl;}
    private:
        int p;
        Base b;
};
```

主函数核心内容:

```
int pp = 45;
A b(pp);
b.print_val();
```

1、观察程序运行结果

答：出现“类Base不存在默认构造函数”的报错。

2、若有问题，请分析是什么原因造成问题的出现

答：这是因为在类A的构造函数定义中，参数列表没有Base b,所以出现报错。

数的一种方式。

5、请完善程序，描述各类构造函数的执行顺序


```
#include <iostream>
using namespace std;

class Base {
public: Base(int a) : val(a) { cout << "构造函数Base" << endl; }
    int getval() { return val; }
private: int val;
};

class A {
public:
    A(int v) : p(v), b(v) { cout << "构造函数A" << endl; }
    void print_val() {
        cout << "hello:" << p << endl;
        cout << "base:" << b.getval() << endl;
    }
private:
    int p;
    Base b;
};

int main() {
    int pp = 45;
    A b(pp), c = b;

    b.print_val();
    c.print_val();
}
```



Microsoft Visual Studio 调试控制台

```
构造函数Base
构造函数A
hello:45
base:45
hello:45
base:45
```

答：先调用Base类的构造函数，再调用A类的构造函数。

2.2 下面的类定义了拷贝构造函数，请完成该类的定义和实现

```
class Myclass {
public:
    Myclass(int xx=0,int yy=0) { X=xx;Y=yy }
    Myclass(const Myclass & c); //拷贝构造函数
private:
    int X,Y;
};
```

//类外定义拷贝构造函数

```
Myclass::Myclass(const Myclass & c) {
    X=c.X;
```

```
Y=c.Y;
}
```

2.3 完善下列代码

```
class Person {
private:
    char* pName;
public:
    Person(char *pN="noName") {
        cout<<"构造中  "<<pN<<"\n";
        pName=new char[strlen(pN)+1];
        if (pName) strcpy(pName,pN);
    }...
}; ...
Person p1("John");//使用构造函数来创建p1
```

A 描述语句执行过程

答：运行程序，输出“构造中”，“John”。首先使用构造函数来创建p1，打印字符串构造中，之后创建一个新的字符数组空间，将pN中的字符串拷贝到pName中。

B 描述执行语句 Person p2(p1); 出现的问题

答：将会调用默认复制构造函数。

C 为避免B问题的出现，该如何处理？请完善拷贝构造函数。

答：可以自定义拷贝构造函数。

```
Person(const Person& p) {
    pName = new char[strlen(p.pName) + 1];
    if (p.pName) strcpy(pName, p.pName);
    cout << "调用复制构造函数" << endl;
}
```

D 完善析构函数

```

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;

class Person {
private:
    char* pName;
public:
    Person(const char* pN = "noName") {
        cout << "构造中 " << pN << "\n";
        pName = new char[strlen(pN) + 1];
        if (pName) strcpy(pName, pN);
    }
    Person(const Person& p) {
        pName = new char[strlen(p.pName) + 1];
        if (p.pName) strcpy(pName, p.pName);
        cout << "调用复制构造函数" << endl;
    }
    void print() {
        cout << pName << endl;
    }
};

int main() {
    Person p1("John"); //使用构造函数来创建p1
    p1.print();
    Person p2(p1);
    p2.print();
}

```

Microsoft Visual Studio 调试控制台

```

构造中 John
John
调用复制构造函数
John

```

2.4 比较1和2代码的不同，分析程序运行过程

1、在构造函数中添加必要输出语句（比如：输出语句），分析程序运行过程（选择哪个函数来运行）

```

#include <iostream>
using namespace std;
class Sample {
    int x;
public:
    Sample() {};
    Sample(int a) {x=a; }

```

2、比较1和2（代码的不同），修改可能存在的问题，分析程序运行

```

#include <iostream>
using namespace std;
class Sample {
    int x;
public:
    Sample() {};
    Sample(int a) {x=a; }
    Sample(Sample a) {x=a.x+1}

```

```

    Sample(Sample &a) {x=a.x+1}
    void disp() {cout<<"x="<<x<<endl; }
}
void main() {
    Sample s1(2),s2(s1);
    S2.disp();
}

```

答：在main函数中，定义了Sample类的对象s1，通过调用带参构造函数进行初始化，定义了Sample类的对象s2，通过调用拷贝构造函数进行初始化。


```

#include <iostream>
using namespace std;

class Sample {
    int x;
public:
    Sample() { cout << "调用无参构造函数" << endl; };
    Sample(int a) {
        cout << "调用带参构造函数" << endl;
        x = a;
    }
    Sample(Sample& a) {
        cout << "调用拷贝构造函数" << endl;
        x = a.x + 1;
    }
    void disp() { cout << "x = " << x << endl; }
};

void main() {
    Sample s1(2), s2(s1);
    s2.disp();
}

```

 Microsoft Visual Studio 调试控制台

```

调用带参构造函数
调用拷贝构造函数
x = 3

```

```

    void disp()
    {cout<<"x="<<x<<endl; }
}
void main() {
    Sample s1(2),s2(s1);
    S2.disp();
}

```

答：类Sample的拷贝构造函数的参数没用引用。若不用引用，则会在参数传递上不停地调用拷贝构造函数 很多编译器会报错。

2.5 用于类型转换的构造函数

```

#include <iostream>
using namespace std;

class Cube {
private:
    double side;
public:
    Cube(double side); //构造函数
    double volume(); //求体积
    bool compareVolume(Cube aCube);
};

Cube::Cube(double length) :side(length) {}
double Cube::volume() { return side * side * side; }
bool Cube::compareVolume(Cube aCube) { return volume() > aCube.volume(); }

int main() {
    Cube box1(5.0); //构造函数创建对象
    Cube box2(3.0);
    if (box1.compareVolume(box2))
        cout << "box1 is larger\n";
    else
        cout << "box1 is not larger\n";
    if (box1.compareVolume(50.0))
        cout << "Volume of box1 is greater than 50\n";
    else
        cout << "Volume of box1 is not greater than 50\n";
    return 0;
}

```

1、针对以上代码，输出结果。

 Microsoft Visual Studio 调试控制台

```

box1 is larger
Volume of box1 is not greater than 50

```

2、分析 if(box1.compareVolume(50.0)) 语句的执行内涵

答：类型转换构造函数其功能是将一个其他类型的数据转换成一个指定的对象。编译器可以使用构造函数把参数的类型隐式转换为类类型。实际是通过构造函数创建一个无名对象，将无名对象复制给函数形参后，系统自动调用析构函数，释放资源。所以这里传入double类型50.5,编译器将其隐式转换成了类类型。

3、若在 **Cube(double side); //构造函数** 这个语句中添加上explicit，程序结果如何？分析原因

答：将会报错：“没有与参数列表匹配的构造函数，参数类型为: (double)”，“没有与指定类型匹配的重载函数实例”。这是因为通过将构造函数声明为explicit，抑制了由构造函数定义的隐式转化。

4、修改 `if(box1.compareVolume(50.0))` 语句中的实参，使其是一个无名的对象，能显示使用构造函数的类型转换功能。并给出结果

答：输出结果相同。

```
if (box1.compareVolume(Cube(50.0)))  
    cout << "Volume of box1 is greater than 50\n";  
else  
    cout << "Volume of box1 is not greater than 50\n";
```

Microsoft Visual Studio 调试控制台

```
box1 is larger  
Volume of box1 is not greater than 50
```

3. 代码题

3.1 自定义类似string的类（有以下不完整的类信息，请完善构造函数、析构函数、复制构造函数、输出函数等）

```
class Mystring {  
    private:  
        char *text;  
    public:  
        Mystring(char *ch); //构造函数  
        //请完善  
};
```

（1）设计代码


```

#include <iostream>
using namespace std;
class Mystring {
private:
    char* text;
public:
    //构造函数
    Mystring(char* ch) {
        cout << "调用构造函数" << endl;
        text = new char(strlen(ch) + 1);
        if (text) { strcpy(text, ch); }
    };
    //拷贝构造函数
    Mystring(const Mystring& s) {
        cout << "调用拷贝函数" << endl;
        text = new char(strlen(s.text) + 1);
        if (text) { strcpy(text, s.text); }
    };
    void print() {
        cout << "打印String: " << text << endl;
    }
    //析构函数
    ~Mystring() {
        cout << "调用析构函数" << endl;
    }
};

int main() {
    char* a = new char[6]{'d','a','t','e','\0','\0'};
    Mystring one(a);
    Mystring two(one);
    return 0;
}

```

(2) 运行结果

Microsoft Visual Studio 调试控制台

```

调用构造函数
调用拷贝函数
调用析构函数
调用析构函数

```

四、简答题

4.1 如何应用构造函数显示地生成一个没有名字的临时对象？

答：以这行代码为例：item.same_isbn(Sales_item(null_book)); 例子中显式调用了构造函数创建临时对象。这里系统调用拷贝构造函数，创建临时对象，并将临时对象作为实参传

入函数中。

4.2 构造函数中，有哪几种情形只能用初始化列表，而不能用函数体中对数据成员赋值？

请给出代码，用对比法（即采用和未采用初始化列表两种形式）并进一步说明

答：1、某个类中的类成员，没有默认构造函数。若没有为类成员提供初始化式，则编译器会隐式地使用类成员的默认构造函数。而该类成员若恰好没有默认构造函数，那么编译器尝试使用默认构造函数就会失败。2、某个类中有const成员；3、某个类中有引用类型的成员；4、如果类存在继承关系，派生类必须在其初始化列表中调用基类的构造函数。

```
#include <iostream>
using namespace std;

//某个类中有const成员
class A {
public:
    A(int size) : SIZE(size) { }
private:
    const int SIZE;
};

//某个类中有引用类型成员
class B {
public:
    B(int& v) : i(v), p(v), j(v) { }
    void print_val() {
        cout << "hello:" << i << " " << j << endl;
    }
private:
    const int i;
    int p;
    int& j;
};

int main() {
    A a(100);
    int r = 45; B b(r); b.print_val();
}
```

4.3 构造函数常常被应用来对象初始化，但是不能简单认为构造函数就只能用来完成对象的初始化工作。请思考分析，还有哪些工作可以在构造函数中完成？

答：构造函数的3个作用：构造对象，初始化对象，类型转换。可以利用类型转换构造函数将一个其他类型的数据 转换成一个指定的对象。

4.4请对拷贝构造函数的形参形式做分析？

答：一般用：常引用做形参，在函数中不能更新引用所绑定的对象，便不会意外地发生对实参的更改（即：只能访问）。

4.5请描述复制构造函数调用的情形。

答：发生以下三种情况复制构造函数被调用：1 用类的对象去初始化该类的另一个对象时，这是用类的对象去初始化另一个对象时的另外一种形式。2 对象作为函数的实参传递给函数的形参时。3 如果函数的返回值是类的对象，函数调用返回时，调用拷贝构造函数。

4.6请描述栈的特点。

答：栈是一种由若干个按线性次序排列的元素所构成的复合数据；插入或删除栈元素只能在栈的一端进行，成为栈顶；栈只能实施两种操作：进栈push和退栈pop；两个操作必须在栈的同一端（栈顶）进行；

4.7请简述析构函数的功能和特性

答：当对象消亡时，在系统收回它所占的内存空间之前，对象类的析构函数会被自动调用。对象空间（本体空间）是由操作系统撤销的；若程序执行过程，对象有申请资源，则需要有析构函数来进行资源的回收。

4.8 深入分析：为何析构函数无法被重载？

答：析构函数是不能重载的，因为没有返回值，也没有参数。对于任何该类的对象，在析构时所做的事情一样。而函数的重载是指：对于同名的多个函数，可以根据他们的参数列表的不同，由compiler选择调用那一个。所以析构函数无法被重载。而且，析构函数也不能被继承，因为子类的与父类的操作不一定一样。

4.9请描述如何将基本类型转为类类型？请描述实现的原理。通过代码来验证类型转换过程。

答：转换构造函数只有一个形参，用户根据需要在函数体中指导编译器如何进行转换。实际是通过构造函数创建一个无名对象，将无名对象复制给函数形参后，系统自动调用析构函数，释放资源。如下代码中，即通过转换构造函数，将基本类型double转换为了类类型Cube。

```

#include <iostream>
using namespace std;
class Cube {
private:
    double side;
public:
    Cube(double side); //构造函数
    double volume(); //求体积
    bool compareVolume(Cube aCube);
};

Cube::Cube(double length) :side(length) { }
double Cube::volume() { return side * side * side; }
bool Cube::compareVolume(Cube aCube) { return volume() > aCube.volume(); }

int main() {
    Cube box1(5.0); //构造函数创建对象
    Cube box2(3.0);
    if (box1.compareVolume(box2))
        cout << "box1 is larger\n";
    else
        cout << "box1 is not larger\n";
    if (box1.compareVolume(50.0))
        cout << "Volume of box1 is greater than 50\n";
    else
        cout << "Volume of box1 is not greater than 50\n";
    return 0;
}

```

五、 附录

源程序文件项目清单： 2.1 2.3 2.4 2.5 3.1 4.2 4.9