第七讲 构造函数和析构函数

一、构造函数

二、拷贝构造函数

三、析构函数

四、构造函数和转型

一、构造函数

构造函数通常是类的公有成员函数，用于对一个类的对象进行初始化。

（1）除一些特殊情况，构造函数一般不需要显式地调用，当声明一个对象时，编译器会自动调用构造函数。

（2）构造函数名与类名相同，构造函数可以被重载（一个类有多个构造函数），但它们必须在调用时通过参数列表来区分；

（3）构造函数总是返回一个类的对象，所以返回值不需要也不能有；

（4）一个构造函数在调用时可以不传参数（构造函数没有参数或构造函数有参数但参数有默认值），这个构造函数称为默认构造函数。如果在程序员没有显式地声明一个构造函数，那么编译器会提供一个空的默认构造函数，这个构造函数不会对数据成员初始化；

（5）如果程序员在类中定义了一个构造函数，但没有定义默认构造函数，编译器不再提供默认构造函数，如果此时创建无参的对象，则发生编译错误，所以为每个类定义一个默认构造函数是编程的好习惯。

一、拷贝构造函数

拷贝构造函数：通过拷贝一个已经存在的对象来创建新的对象，也就是说，这个构造函数的参数就是另外一个对象。`Vector(Vector & v);`

（a）用一个已存在的对象初始化另一个对象

```
classT object(another_object), or
classT object = another_object;
```

第二种情况编译器会调用拷贝构造函数而不是赋值操作。

下面的代码将调用赋值运算符重载：

```
classT object;  //创建一个对象
classT another_object;  //创建另一个对象
object = another_object; //把一个对象赋值给另一个对象
```

（b）以传值的方式，给函数传递一个对象或从函数返回一个对象，也将调用拷贝构造函数；

从函数传值返回一个对象时，会调用拷贝构造函数，创建一个临时对象，接收返回值。

```
A func(A a)//传值时调用拷贝构造函数
{
    A b;
    return b;
    //传值返回时也将调用拷贝构造函数
}
```

注意：拷贝构造函数的参数，必须传引用，不能传值！！！如果拷贝构造函数的参数用传值的方式，那么当实参传递给拷贝构造函数的时候，拷贝构造函数的形参，为了复制实参，则又需要调用拷贝构造函数，即调用它自己，于是形成死循环—杯具了。

三、析构函数

析构函数也是类的特殊的成员函数，当一个类的对象被销毁（离开它的作用域）时用于释放对象占用的内存空间。

(i) 析构函数不需要显式地被调用，当对象被销毁时，编译器将自动调用析构函数来释放对象占用的内存空间。

(ii) 一个类中只能有一个析构函数，即析构函数不能被重载。

(iii) 析构函数名和类名一样，但是有一个～ 前缀，如 ~Vector()。

(iii) 析构函数无参也无返回值。

(iv) 如果一个类没有显式地声明一个析构函数，那么编译器将会提供一个析构函数，来释放对象的数据成员占用的内存，但对于指针类型的成员，仅仅释放指针本身的内存，但不释放指针指向的堆上的内存。这种情况，必须由程序员定义一个析构函数，用delete来释放堆内存。

因此，如果一个类包含指针成员变量，并且指向堆上空间，则程序员必须自己实现三个函数：拷贝构造函数、析构函数和赋值运算符的重载。以实现深拷贝。Big Three 三大件。

【例2-1】拷贝构造函数示例

```
/** Vector.h  **/
#include <iostream>
using namespace std;
class Vector{
public:
    Vector(int s = 0);
    Vector(int *, int);
    Vector(const Vector & v);
    ~Vector(){dispose();}
    int get_size()const{return size;}
    const Vector & operator=(const Vector & x);
    int & operator[](int index){return rep[index];}
    const int & operator[](int index)const{return rep[index];}

private:
    int * rep;
    int size;
    void clone(const Vector & a);
    void dispose();
};
```

```
/** Vector.cpp  **/
#include "Vectror.h"

void Vector::clone(const Vector & a) {
    this->size = a.size;
    rep = new int[size];
    for (int count = 0; count < size; ++count)
        rep[count] = a[count];
}
```

```
void Vector::dispose() {
    delete [] rep;
}

Vector::Vector(int s):size(s) {
    if (size <= 0)
        rep = NULL;
    else{
        rep = new int[size];
        for (int count = 0; count < size; ++count)
            rep[count] = 0;
    }
}
```

```
/** Vector.cpp  **/

Vector::Vector(int * a,int s):size(s),rep(new int[s]){
    for (int count = 0; count < size; ++count)
        rep[count] = a[count];
}


Vector::Vector(const Vector & v) {
    clone(v);
}

const Vector & Vector::operator=(const Vector &x) {
    if (this != &x) {
        delete [] rep;
        this->size = x.size;
        rep = new int[size];
        for (int count = 0; count <size; ++count)
            rep[count] = x[count];
    }
    return *this;
}
```

```
/** Vector.cpp  **/

ostream & operator<<(ostream & out, const Vector & x) {
    int s = x.get_size();
    for (int i = 0; i < s; ++i)
        out<<x[i]<<endl;
    out<<endl;
    return out;
}

bool operator==(const Vector & a,const Vector & b) {
    bool yes = true;
    if (a.get_size() != b.get_size())
        yes = false;
    else{
        int s = a.get_size(),index = 0;
       while (index < s && a[index] == b[index])
            ++index;
        if (index < s)
            yes = false;
    }
    return yes;
}
```

```
/** main.cpp  **/
#include <iostream>
#include "Vector.h"
using namespace std;

ostream& operator<<(ostream& out, const Vector& x);
bool operator==(const Vector& a, const Vector& b);

int main(int argc, const char * argv[]) {
    Vector vec1;  cout<<vec1<<endl;

    int array[5] = {1,2,3,4,5};
    Vector vec2(array, 5); cout<<vec2<<endl;

    Vector vec3(vec2);   cout<<vec3<<endl;

    if( vec3 == vec2 ) {
        cout<<"The vector3 is equal to vector2"<<endl;
    }

    Vector vec4;
    vec4 = vec3;
    cout<<vec4<<endl;
    return 0;
}
```

【例2-2】构造函数析构函数调用示例

```
/** Demo.h  **/

#ifndef Demo_h
#define Demo_h

class Demo{
private:
    int i;
public:
    Demo(int n = 0);
    Demo(const Demo & a);
    ~Demo();
    const Demo & operator=(const Demo & a);
    Demo & set_value(int n) ;
    int get_value() ;
};

#endif /* Demo_h */
```

【例2-2】构造函数析构函数调用示例

```
/** Demo.cpp  **/
#include "Demo.h"

Demo::Demo(int n = 0):i(n) { cout<<"default constructor called"<<endl; }

Demo::Demo(const Demo & a) {
    i = a.i;
    cout<<"copy constructor called"<<endl;
}

Demo::~Demo() { cout<<"destructor called"<<endl;
}

const Demo & Demo::operator=(const Demo & a) {
    this->i = a.i;
    cout<<"assignment operator used"<<endl;
    return *this;
}

Demo & Demo::set_value(int n) {i = n; return *this;}

int Demo::get_value() { return i;}
```

```
/** main.cpp  **/

Demo foo(Demo x){
    Demo d;
    return d;
}
int main(int argc, const char * argv[]) {
    Demo a(2);
    {
        Demo b;
        b = foo(a);
    }
    Demo c = a;
    return 0;
}
```

根据输出结果思考一下调用顺序。

运行结果如下：

```
default constructor called
default constructor called
copy constructor called
default constructor called
assignment operator used
destructor called
destructor called
destructor called
copy constructor called
destructor called
destructor called
```

二、构造函数和类型转换

当构造函数只有一个参数时，可以用来进行转型工作。类A构造函数的参数是T类型。当希望得到A 的对象，实际提供的是T类型的变量，则该构造函数将被自动调用，把T类型的变量，转型为A类型的对象。

```
/** String.h **/
#ifndef String_h
#define String_h
#include <iostream>
using namespace std;

class String {
    friend void print(const String& s);
public:
    String();
    String(char * s);
    String(const String& str);
    ~String();

private:
    int len; char * rep;
};
#endif /* String_h */
```

```
/** String.cpp  **/
#include "String.h"
String::String(): len(0) {
    rep = new char[1];
    strcpy(rep, "");
}

//用来进行转型的构造函数，把char*类型的s，转型为String类型的变量。
String::String(char* s) {
    if (!s) { //如果s是空串
        len = 0;
         s = new char[1];
         strcpy(rep, "");
    }else {//如果s不是空串
        len = strlen(s);
         rep = new char[len + 1];
         strcpy(rep, s);
    }
}

String::String(const String& str):len(str.len),rep(new char[str.len+1]) {
    strcpy(rep, str.rep);
}

String::~String( ) {delete [ ] rep;}
```

```
/** main.cpp  **/
#include <iostream>
using namespace std;
#include "String.h"

void print(const String& s){
    cout << s.rep << " ";
    return;
}

int main(int argc, const char * argv[]) {
    char * a = "first string";
    print(a);
    print("second string");
    String b = "bad initialization";
    print(b);
    return 0;
}
```

我们期望，print函数的参数是String类型（是print函数的定义），实际的参数是char*类型，此时，就会调用以char*作为参数的String的构造函数，来实现转型。把char*类型的a，转型成为String类型的对象，然后供print函数使用。转型过程中，生成了一个String类型的临时对象。

The End

渥瑞达·教学部