第六讲 操作符重载（二）

一、赋值运算符重载

二、==运算符重载

一、赋值运算符重载

尽管一个对象可以通过赋值语句分配给另一个对象，正如我们前面所提到的，这个操作可能只创建一个逻辑拷贝（即成员和成员的浅拷贝）。浅拷贝中，一个对象的成员，仅仅简单copy另一个对象的成员的值。

如果对象包含指针成员，copy的仅仅是指针变量本身的值（地址）。即两个对象的指针变量指向同一块空间，会被释放两次。

对于堆上的共享内存，我们不知道由谁来负责释放这块内存，这个内存有没有被释放，或者它被释放两次,这都是不允许的。

如果成员变量有指向堆内存的指针，需要重载赋值运算符，实现深拷贝。

【示例1-1】浅拷贝

```
/** Account.h **/
#define MAX_CHAR 10
class Account {
    friend ostream &operator<<(ostream &out,const Account &a);

public:
    Account(const char *_title = "Miss",const char *_owner =
"unknown",float _balance = 0.0);
    void changeTitle(const char *_title);
    void changeOwner(const char *_owner);
    ~Account();

private:
    char *title;
    char owner[MAX_CHAR];
    float balance;
};
```

```cpp
/** Account.cpp **/
Account::Account(const char *_title, const char *_owner, float _balance) {
    title = new char[strlen(_title)+1];
    strcpy(title, _title);
    strcpy(owner, _owner);
    balance = _balance;
}

void Account::changeTitle(const char *_title) {
    if (strlen(_title) > strlen(title)) {
        delete []title;
        title = new char[strlen(_title)+1];
        strcpy(title, _title);
    }else {
        strcpy(title, _title);
    }
}

void Account::changeOwner(const char *_owner) {
    strcpy(owner, _owner);
}

Account::~Account() {
    delete []title;
}

ostream &operator<<(ostream &out, const Account &a) {
    out<<"Who:"<<a.title<<" "<<a.owner<<"\thow much:"<<a.balance;
    return out;
}
```

```
/** main.c **/
int main(int argc, const char * argv[]) {
    Account account1("Miss","li",100);
    Account account2;

    account2 = account1;
    cout<<account1<<endl;
    cout<<account2<<endl;

    account1.changeTitle("Mr");
    cout<<account1<<endl;
    cout<<account2<<endl;

    account1.changeOwner("wang");
    cout<<account1<<endl;
    cout<<account2<<endl;
}
```

运行结果如下:

```
Who:Miss li   how much:100
Who:Miss li   how much:100
Who:Mr li     how much:100
Who:Mr li     how much:100
Who:Mr wang   how much:100
Who:Mr li     how much:100
```

示例分析：

预期结果错误：修改account1对象的title，account2对象的title也会改变；

运行时错误：试图两次删除同一块内存。如果析构时不删除指针，title指向的内存就没有被回收。

解决方案：重载赋值运算符来开辟一块新的堆内存，实现深拷贝。

在类里重载赋值运算符，通常需要做五件事情：

(1) 判断是否是自赋值，即判断是否赋值给自身；

(2) 释放掉指针成员原来指向的内存，避免内存泄露；

(3) 为指针成员开辟新内存；

(4) 右值的内容拷贝给左值；

(5) 返回*this实现链式表达式

下面这个例子定义了类的两个版本:

```
/** Vector.h **/
class Vector {
public:
    Vector(int _size,int array[]);
    const int &operator[](int index) const;
    int &operator[](int index);
    int getSize() const;
    const Vector &operator=(const Vector &v);
    ~Vector();

private:
    int *rep;
    int size;
};
```

```cpp
/** Vector.cpp **/
Vector::Vector(int _size,int array[]):size(_size),rep(new int[_size]) {
    for (int i = 0; i < _size; i++) {
        rep[i] = array[i];
    }
}
int Vector::getSize() const {
    return size;
}
const int &Vector::operator[](int index) const {
    return rep[index];
}
int &Vector::operator[](int index) {
    return rep[index];
}
const Vector &Vector::operator=(const Vector &v) {
    if (this != &v) {
        delete []rep;
        rep = new int[v.size];
        for (int i = 0; i < v.size; i++) {
            rep[i] = v[i];
        }
        size = v.size;
    }
    return *this;
}
Vector::~Vector() {
    delete []rep;
}
```

```
ostream &operator<<(ostream &out, const Vector &v) {
    for (int i = 0; i < v.getSize(); i++) {
        out<<v[i]<<"\t";
    }
    return out;
}
int main(int argc, const char * argv[]) {
    int a[] = {1,2,3,4,5};
    int b[] = {0,0,0,0,0};
    Vector v1(5,a);
    Vector v2(5,b);

    cout<<v1<<endl;
    cout<<v2<<endl;

    v2 = v1;
    cout<<v1<<endl;
    cout<<v2<<endl;

    v2[0] = 100;
    cout<<v1<<endl;
    cout<<v2<<endl;
}
```

运行结果如下：

```
1   2   3   4   5
0   0   0   0   0

1   2   3   4   5
1   2   3   4   5

1   2   3   4   5
100   2   3   4   5
```

示例分析：这个类中，成员变量是一个动态数组，这时候拷贝构造函数、析构函数、 赋值运算符重载函数是必要的“三大件”。

二、==运算符重载

在很多情况下，我们需要判断两个对象时候相等，那么我们需要对==运算符进行重载，下面我们看看应该如何对==运算符进行重载：

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

The End

渥瑞达·教学部