C++重载=（赋值运算符）

在《到底什么时候会调用拷贝构造函数？》一节中，我们讲解了初始化和赋值的区别：在定义的同时进行赋值叫做初始化（Initialization），定义完成以后再赋值（不管在定义的时候有没有赋值）就叫做赋值（Assignment）。初始化只能有一次，赋值可以有多次。

当以拷贝的方式初始化一个对象时，会调用拷贝构造函数；当给一个对象赋值时，会调用重载过的赋值运算符。

即使我们没有显式的重载赋值运算符，编译器也会以默认地方式重载它。默认重载的赋值运算符功能很简单，就是将原有对象的所有成员变量一一赋值给新对象，这和默认拷贝构造函数的功能类似。

对于简单的类，默认的赋值运算符一般就够用了，我们也没有必要再显式地重载它。但是当类持有其它资源时，例如动态分配的内存、打开的文件、指向其他数据的指针、网络连接等，默认的赋值运算符就不能处理了，我们必须显式地重载它，这样才能将原有对象的所有数据都赋值给新对象。

仍然以上节的 Array 类为例，该类拥有一个指针成员，指向动态分配的内存。为了让 Array 类的对象之间能够正确地赋值，我们必须重载赋值运算符。请看下面的代码：

```
01.  #include <iostream>
02.  #include <cstdlib>
03.  using namespace std;
04.
05.  //变长数组类
06.  class Array{
07.  public:
08.      Array(int len);
09.      Array(const Array &arr);  //拷贝构造函数
10.      ~Array();
11.  public:
12.      int operator[](int i) const { return m_p[i]; }  //获取元素（读取）
13.      int &operator[](int i){ return m_p[i]; }  //获取元素（写入）
14.      Array & operator=(const Array &arr);  //重载赋值运算符
15.      int length() const { return m_len; }
16.  private:
17.      int m_len;
18.      int *m_p;
19.  };
20.
21.  Array::Array(int len): m_len(len){
22.      m_p = (int*)calloc( len, sizeof(int) );
23.  }
```

```
24.
25. Array::Array(const Array &arr){  //拷贝构造函数
26.     this->m_len = arr.m_len;
27.     this->m_p = (int*)calloc( this->m_len, sizeof(int) );
28.     memcpy( this->m_p, arr.m_p, m_len * sizeof(int) );
29. }
30.
31. Array::~Array(){ free(m_p); }
32.
33. Array &Array::operator=(const Array &arr){  //重载赋值运算符
34.     if( this != &arr){  //判断是否是给自己赋值
35.         this->m_len = arr.m_len;
36.         free(this->m_p);  //释放原来的内存
37.         this->m_p = (int*)calloc( this->m_len, sizeof(int) );
38.         memcpy( this->m_p, arr.m_p, m_len * sizeof(int) );
39.     }
40.     return *this;
41. }
42.
43. //打印数组元素
44. void printArray(const Array &arr){
45.     int len = arr.length();
46.     for(int i=0; i<len; i++){
47.         if(i == len-1){
48.             cout<<arr[i]<<endl;
49.         }else{
50.             cout<<arr[i]<<", ";
51.         }
52.     }
53. }
54.
55. int main(){
56.     Array arr1(10);
57.     for(int i=0; i<10; i++){
58.         arr1[i] = i;
59.     }
60.     printArray(arr1);
61.
62.     Array arr2(5);
63.     for(int i=0; i<5; i++){
64.         arr2[i] = i;
65.     }
66.     printArray(arr2);
67.     arr2 = arr1;  //调用operator=()
```

```
68.     printArray(arr2);
69.     arr2[3] = 234;  //修改arr1的数据不会影响arr2
70.     arr2[7] = 920;
71.     printArray(arr1);
72.
73.     return 0;
74. }
```

运行结果：
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
0, 1, 2, 3, 4
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
0, 1, 2, 3, 4, 5, 6, 7, 8, 9

将 arr1 赋值给 arr2 后，修改 arr2 的数据不会影响 arr1。如果把 operator=() 注释掉，那么运行结果将变为：
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
0, 1, 2, 3, 4
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
0, 1, 2, 234, 4, 5, 6, 920, 8, 9

> 去掉operator=()后，由于 m_p 指向的堆内存会被 free() 两次，所以还会导致内存错误。

下面我们就来分析一下重载过的赋值运算符。

1) operator=() 的返回值类型为 `Array &`，这样不但能够避免在返回数据时调用拷贝构造函数，还能够达到连续赋值的目的。下面的语句就是连续赋值：

```
arr4 = arr3 = arr2 = arr1;
```

2) `if( this != &arr)` 语句的作用是「判断是否是给同一个对象赋值」：如果是，那就什么也不做；如果不是，那就将原有对象的所有成员变量一一赋值给新对象，并为新对象重新分配内存。下面的语句就是给同一个对象赋值：

```
arr1 = arr1;
arr2 = arr2;
```

3) `return *this` 表示返回当前对象（新对象）。

4) operator=() 的形参类型为 `const Array &`，这样不但能够避免在传参时调用拷贝构造函数，还能够同时接收 const 类型和非 const 类型的实参，这一点已经在《C++拷贝构造函数》中进行了详细讲解。

5) 赋值运算符重载函数除了能有对象引用这样的参数之外，也能有其它参数。但是其它参数必须给出默认值，例如：

```
Array & operator=(const Array &arr, int a = 100);
```