# 常用特殊运算符

##### 递增和递减运算符

**重载此运算符只能重载为成员运算符，无法重载为全局运算符**

前置格式

类型 operator++（）

int operator++();

后置格式

类型 operator++(int)

后置格式中的int类型形参无实际意义，只是在重载运算符中用来区分前置和后置运算，编译器为这个值提供的是0。

##### 重载赋值运算符（等于号=）

void operator=(const int&);

A a1;

a1 = 100;//使用重载后的赋值运算符

##### 下标运算符重载[]

int operator[](const A&) const;

注意，参数必须要给予，参数类型没有硬性要求

##### 重载运算符->

注意此运算符不是为指针数据进行重载，而是为了非指针类型对象进行重载。所以操作时需要注意，指针类型数据的->运算符由编译器给予，我们无法进行重载。

**重载为成员运算符，并且此运算符不接受参数。一般将内部的一个指针数据进行返回。如果无返回参数，则此重载无意义。**

##### 运算符重载的基本规则

1. 不能重载C++基本数据类型运算规则，重载运算符至少需要包含一个自定义的数据类型（类，结构体）
2. 禁止额外虚构超过C++提供的运算符
3. 运算符重载是遵循优先级约束的
4. 重载运算符不限定运算数据类型，可以通过多次重载完成更多的数据类型运算
5. 重载运算符后，左侧的操作数据是调用对象，前后位置不能颠倒
6. 注意运算符的参数个数，所有的运算符一般只会和一个数据进行运算，所以除了全局运算符重载以外，成员的运算符重载参数一般只有一个。
7. 对于关系运算符，如果重载了==运算符，也应该重载！=运算符，这是操作配套。

# C++中的特殊函数

特殊函数：构造函数，析构函数，拷贝构造函数，赋值函数，移动构造函数（C++11标准），移动赋值函数（C++11标准），在自定义类型中，这些函数都可以不显示声明，由编译器帮助我们构建默认的操作方式，编译器为我们生成的函数被称为合成函数

##### 析构函数调用场景

1. 变量离开作用域
2. 当一个对象被销毁时
3. 容器被销毁时（数组或是标准库均是如此），其元素也被销毁
4. 动态分配对象时，调用delete运算符
5. 临时对象，创建他的完整表达式结束时

##### 拷贝构造函数

定义：如果一个构造函数的第一个参数是自身的引用，且任何额外的参数都有默认值，此构造函数是拷贝构造函数

Hero(Hero&);

**拷贝发生的情况：（非指针类型数据）**

1. 将一个对象作为实参传递给一个非引用类型的形参
2. 从一个返回类型为非引用类型的函数返回一个对象
3. 用花括号初始化一个数组中的元素或一个聚合类中的成员

##### 聚合类

没有构造函数，基类，虚函数，私有或是受保护的成员（函数和属性）的类、结构体、联合

##### 深拷贝和浅拷贝

自定义数据中，由于默认合成拷贝构造函数实行的是位拷贝（浅拷贝），拷贝可能出现不合理的情况。例如指针的拷贝，原内容被释放，将导致拷贝内容出错（因为拷贝的只是指针值，而不是指针指向的内容）

深拷贝：不考虑数据的实际意义，完全进行拷贝生成新的内存空间。需要进行数据递归操作。

浅拷贝：根据对象内容，进行逐成员位拷贝复制，不会关心成员内容，只关心成员值。

##### 拷贝赋值运算符

拷贝赋值运算符发生在已经成功构建的对象后，进行相互赋值。

通过重载复制运算符完成拷贝赋值操作。

重载赋值运算符的第一个参数是当前类引用，我们称此类赋值运算符为**拷贝赋值运算符**

##### 移动构造函数

移动构造函数的意义

对于编程来说，临时变量是可以被修改，并且不会影响其他数据的，所以我们可以在赋值过程中直接掠夺临时变量中的成员数据空间，然后将临时变量释放逻辑进行调整，从而减少了对新对象进行空间开辟的浪费

**如果没有显示编写移动构造函数，所有临时变量赋值将使用拷贝构造函数**

拷贝构造函数无法区分**短暂对象和持久对象**，而移动构造函数可以**只处理短暂对象**。当类内部有移动构造函数和拷贝构造函数时，使用短暂对象构建对象，将采用移动构造函数，使用持久对象进行构建时，将采用拷贝构造函数构建。

##### 移动赋值运算符

普通赋值运算符的第一个参数是该类的右值引用，我们称此赋值运算符为移动赋值运算符

##### 右值引用（C++11）

新标准提供的一种引用类型，用来绑定右值的引用。使用操作符&&用来标记。右值引用只能绑定到一个将要销毁的对象身上。所以我们可以放心的将右值引用的资源进行“窃取”，到另一个对象中。

**左值持久，右值短暂。**

右值引用的格式

A&& ta = Get();

int&& a = 11;

我们一般不声明右值，因为没有操作实际意义。