MyStr& operator =(const MyStr& str)//赋值运算符 {

cout << "operator =" << endl;

if (this != &str)

{

if (name != NULL)

delete[] name;

this->id = str.id;

int len = strlen(str.name);

name = new char[len + 1];

strcpy\_s(name, strlen(str.name) + 1, str.name);

}

return \*this;

}

**Ⅱ.参数**

一般地，赋值运算符重载函数的参数是函数所在类的const类型的引用（如上面例1），加const是因为：

①我们不希望在这个函数中对用来进行赋值的“原版”做任何修改。

②加上const，对于const的和非const的实参，函数就能接受；如果不加，就只能接受非const的实参。

用引用是因为：

这样可以避免在函数调用时对实参的一次拷贝，提高了效率。

**注意**：

上面的规定都不是强制的，可以不加const，也可以没有引用，甚至参数可以不是函数所在的对象，正如后面例2中的那样。

**Ⅲ.返回值**

一般地，返回值是被赋值者的引用，即\*this（如上面例1），原因是

①这样在函数返回时避免一次拷贝，提高了效率。

②更重要的，这样可以实现连续赋值，即类似a=b=c这样。如果不是返回引用而是返回值类型，那么，执行a=b时，调用赋值运算符重载函数，在函数返回时，由于返回的是值类型，所以要对return后边的“东西”进行一次拷贝，得到一个未命名的副本（有些资料上称之为“匿名对象”），然后将这个副本返回，而这个副本是右值，所以，执行a=b后，得到的是一个右值，再执行=c就会出错。

**注意**：

这也不是强制的，我们可以将函数返回值声明为void，然后什么也不返回，只不过这样就不能够连续赋值了。

**Ⅳ.调用时机**

      当为一个类对象赋值（注意：可以用本类对象为其赋值（如上面例1），也可以用其它类型（如内置类型）的值为其赋值，关于这一点，见后面的例2）时，会由该对象调用该类的赋值运算符重载函数。

如上边代码中

str2 = str1;

一句，用str1为str2赋值，会由str2调用MyStr类的赋值运算符重载函数。

需要注意的是，

MyStr str2;

str2 = str1;

和

MyStr str3 = str2;

在调用函数上是有区别的。正如我们在上面结果中看到的那样。

      前者MyStr str2;一句是str2的声明加定义，调用无参构造函数，所以str2 = str1;一句是在str2已经存在的情况下，用str1来为str2赋值，调用的是拷贝赋值运算符重载函数；而后者，是用str2来初始化str3，调用的是拷贝构造函数。

**Ⅴ.提供默认赋值运算符重载函数的时机**

①如果匹配的构造函数和赋值运算符重载函数同时存在（如例2），会调用赋值运算符重载函数。

②如果只有匹配的构造函数存在，就会调用这个构造函数

**Ⅶ.显式提供赋值运算符重载函数的时机**

①用非类A类型的值为类A的对象赋值时（当然，从Ⅵ中可以看出，这种情况下我们可以不提供相应的赋值运算符重载函数而只提供相应的构造函数来完成任务）。

②当用类A类型的值为类A的对象赋值且类A的成员变量中含有指针时，为避免浅拷贝（关于浅拷贝和深拷贝，下面会讲到），必须显式提供赋值运算符重载函数（如例1）。

**Ⅸ.赋值运算符重载函数只能是类的非静态的成员函数**

是因为静态成员函数只能操作类的静态成员，不能操作非静态成员

**Ⅹ. 赋值运算符重载函数不能被继承**

为什么赋值运算符重载函数不能被继承呢？

     因为相较于基类，派生类往往要添加一些自己的数据成员和成员函数，如果允许派生类继承基类的赋值运算符重载函数，那么，在派生类不提供自己的赋值运算符重载函数时，就只能调用基类的，但基类版本只能处理基类的数据成员，在这种情况下，派生类自己的数据成员怎么办？

**Ⅺ.赋值运算符重载函数要避免自赋值**

对于赋值运算符重载函数，我们要避免自赋值情况（即自己给自己赋值）的发生，一般地，我们通过比较赋值者与被赋值者的地址是否相同来判断两者是否是同一对象（正如例1中的if (this != &str)一句）。

 ①为了效率。显然，自己给自己赋值完全是毫无意义的无用功，特别地，对于基类数据成员间的赋值，还会调用基类的赋值运算符重载函数，开销是很大的。如果我们一旦判定是自赋值，就立即return \*this，会避免对其它函数的调用。

②如果类的数据成员中含有指针，自赋值有时会导致灾难性的后果。对于指针间的赋值（注意这里指的是指针所指内容间的赋值，这里假设用\_p给p赋值），先要将p所指向的空间delete掉（为什么要这么做呢？因为指针p所指的空间通常是new来的，如果在为p重新分配空间前没有将p原来的空间delete掉，会造成内存泄露），然后再为p重新分配空间，将\_p所指的内容拷贝到p所指的空间。如果是自赋值，那么p和\_p是同一指针，在赋值操作前对p的delete操作，将导致p所指的数据同时被销毁。那么重新赋值时，拿什么来赋？