Const，Const函数，Const变量，函数后面的Const

看到const 关键字，C++程序员首先想到的可能是const 常量。这可不是良好的条件反射。如果只知道用const 定义常量，那么相当于把火药仅用于制作鞭炮。const 更大的魅力是它可以修饰函数的参数、返回值，甚至函数的定义体。

const 是constant 的缩写，“恒定不变”的意思。被const 修饰的东西都受到强制保护，可以预防意外的变动，能提高程序的健壮性。所以很多C++程序设计书籍建议：“Use const whenever you need”。

1.用const 修饰函数的参数

如果参数作输出用，不论它是什么数据类型，也不论它采用“指针传递”还是“引用传递”，都不能加const 修饰，否则该参数将失去输出功能。const 只能修饰输入参数：

如果输入参数采用“指针传递”，那么加const 修饰可以防止意外地改动该指针，起到保护作用。

例如StringCopy 函数：

void StringCopy(char \*strDestination, const char \*strSource);

其中strSource 是输入参数，strDestination 是输出参数。给strSource 加上const修饰后，如果函数体内的语句试图改动strSource 的内容，编译器将指出错误。

如果输入参数采用“值传递”，由于函数将自动产生临时变量用于复制该参数，该输入参数本来就无需保护，所以不要加const 修饰。

例如不要将函数void Func1(int x) 写成void Func1(const int x)。同理不要将函数void Func2(A a) 写成void Func2(const A a)。其中A 为用户自定义的数据类型。

对于非内部数据类型的参数而言，象void Func(A a) 这样声明的函数注定效率比较底。因为函数体内将产生A 类型的临时对象用于复制参数a，而临时对象的构造、复制、析构过程都将消耗时间。

为了提高效率，可以将函数声明改为void Func(A &a)，因为“引用传递”仅借用一下参数的别名而已，不需要产生临时对象。但是函数void Func(A &a) 存在一个缺点：

“引用传递”有可能改变参数a，这是我们不期望的。解决这个问题很容易，加const修饰即可，因此函数最终成为void Func(const A &a)。

以此类推，是否应将void Func(int x) 改写为void Func(const int &x)，以便提高效率？完全没有必要，因为内部数据类型的参数不存在构造、析构的过程，而复制也非常快，“值传递”和“引用传递”的效率几乎相当。

问题是如此的缠绵，我只好将“const &”修饰输入参数的用法总结一下。

对于非内部数据类型的输入参数，应该将“值传递”的方式改为“const 引用传递”，目的是提高效率。例如将void Func(A a) 改为void Func(const A &a)。

对于内部数据类型的输入参数，不要将“值传递”的方式改为“const 引用传递”。否则既达不到提高效率的目的，又降低了函数的可理解性。例如void Func(int x) 不应该改为void Func(const int &x)。

2 用const 修饰函数的返回值

如果给以“指针传递”方式的函数返回值加const 修饰，那么函数返回值（即指针）的内容不能被修改，该返回值只能被赋给加const 修饰的同类型指针。例如函数

const char \* GetString(void);

如下语句将出现编译错误：

char \*str = GetString();

正确的用法是

const char \*str = GetString();

如果函数返回值采用“值传递方式”，由于函数会把返回值复制到外部临时的存储单元中，加const 修饰没有任何价值。

例如不要把函数int GetInt(void) 写成const int GetInt(void)。

同理不要把函数A GetA(void) 写成const A GetA(void)，其中A 为用户自定义的数据类型。

如果返回值不是内部数据类型，将函数A GetA(void) 改写为const A & GetA(void)的确能提高效率。但此时千万千万要小心，一定要搞清楚函数究竟是想返回一个对象的“拷贝”还是仅返回“别名”就可以了，否则程序会出错。

函数返回值采用“引用传递”的场合并不多，这种方式一般只出现在类的赋值函数中，目的是为了实现链式表达。

例如：

class A

{

A & operate = (const A &other); // 赋值函数

};

A a, b, c; // a, b, c 为A 的对象

a = b = c; // 正常的链式赋值

(a = b) = c; // 不正常的链式赋值，但合法

如果将赋值函数的返回值加const 修饰，那么该返回值的内容不允许被改动。上例中，语句 a = b = c 仍然正确，但是语句 (a = b) = c 则是非法的。

3 const 成员函数

任何不会修改数据成员的函数都应该声明为const 类型。如果在编写const 成员函数时，不慎修改了数据成员，或者调用了其它非const 成员函数，编译器将指出错误，这无疑会提高程序的健壮性。以下程序中，类stack 的成员函数GetCount 仅用于计数，从逻辑上讲GetCount 应当为const 函数。编译器将指出GetCount 函数中的错误。

class Stack

{

public:

void Push(int elem);

int Pop(void);

int GetCount(void) const; // const 成员函数

private:

int m\_num;

int m\_data[100];

};

int Stack::GetCount(void) const

{

++ m\_num; // 编译错误，企图修改数据成员m\_num

Pop(); // 编译错误，企图调用非const 函数

return m\_num;

}

const 成员函数的声明看起来怪怪的：const 关键字只能放在函数声明的尾部，大概是因为其它地方都已经被占用了。

关于Const函数的几点规则：

a. const对象只能访问const成员函数,而非const对象可以访问任意的成员函数,包括const成员函数.

b. const对象的成员是不可修改的,然而const对象通过指针维护的对象却是可以修改的.

c. const成员函数不可以修改对象的数据,不管对象是否具有const性质.它在编译时,以是否修改成员数据为依据,进行检查.

e. 然而加上mutable修饰符的数据成员,对于任何情况下通过任何手段都可修改,自然此时的const成员函数是可以修改它的

C++中的mutable和const

声明：这里讨论的const是用来修饰函数的const，而不是用来修饰变量的const。虽然是同一个关键字，但yayv还是觉得把他们当作2个关键字来理解更好一些。

　　 C++中const关键字用来表示一个常量，同时const也用来修饰函数。const所修饰的函数只能是类的成员函数，因为const所修饰的函数中，要由编译器负责保护类的成员变量不被修改。而相对的，mutable则是用来修饰类的成员变量，让该变量在const所修饰的成员函数中可以被修改。而且const修饰的函数只能是类的成员函数，mutable修饰的变量只能是类的成员变量。简直就是一对冤家对头~

　　这里出现了3个问题：

　　第一：为什么要保护类的成员变量不被修改

　　第二：为什么用const保护了成员变量，还要再定义一个mutable关键字来突破const的封锁线？

第三：到底有没有必要使用const 和 mutable这两个关键字？

看法是：保护类的成员变量不在成员函数中被修改，是为了保证模型的逻辑正确，通过用const关键字来避免在函数中错误的修改了类对象的状态。并且在所有使用该成员函数的地方都可以更准确的预测到使用该成员函数的带来的影响。而mutable则是为了能突破const的封锁线，让类的一些次要的或者是辅助性的成员变量随时可以被更改。没有使用const和mutable关键字当然没有错，const和mutable关键字只是给了建模工具更多的设计约束和设计灵活性，而且程序员也可以把更多的逻辑检查问题交给编译器和建模工具去做，从而减轻程序员的负担。如果开发过程有比较严格的迭代过程，使用这两个关键字应该更能体现出他们的作用。

关于链式表达式

五、编写strcpy函数（10分）

已知strcpy函数的原型是 char \*strcpy(char \*strDest, const char \*strSrc);

其中strDest是目的字符串，strSrc是源字符串。

（1）不调用C++/C的字符串库函数，请编写函数

strcpy char \*strcpy(char \*strDest, const char \*strSrc);

{

assert((strDest!=NULL) && (strSrc !=NULL)); // 2分

char \*address = strDest; // 2分

while( (\*strDest++ = \* strSrc++) != ‘/0’ ) // 2分

NULL ;

return address ; // 2分

}

（2）strcpy能把strSrc的内容复制到strDest，为什么还要char \* 类型的返回值？

答：为了实现链式表达式。 // 2分

例如 int length = strlen( strcpy( strDest, “hello world”) );

--------------------------------------------------------------------------------------------------------------------------------

看到了这个词：链式表达式,之前没有听过,所以去百度了一下,发现有人解释的还算明白:

-------------------------------------------------------------------------------------------------------------------------------- http://topic.csdn.net/t/20061123/21/5180993.html 1，

就是方便一些，否则就要这样写：

char strDest[12];

strcpy( strDest, “hello world”);

int length = strlen(strDest);

什么链式就是一连串写下来的意思。。。

--------------------------------------------------------------------------------------------------------------------------------

其实,说白了,就是如果上面的字符串拷贝函数strcopy的返回值是void,那么,上面那句:

int length = strlen( strcpy( strDest, “hello world”) );

就要像上面那位的回答,写成好几句了:

char strDest[12];

strcpy( strDest, “hello world”);

int length = strlen(strDest);

而这种直接返回char \*的手段,就是为了后来函数调用者方便而设计的.不用你这么麻烦用上述方法去使用了,而直接可以使用拷贝后的dest字符串了.这种方便的实现方法,看起来就是链子链在一起的,所以称为 链式表达式