交换两个变量的算法探究

以交换两个整型变量为例，通常在C语言中实现如下：

1. **int** swap(**int** \*a, **int** \*b)
2. {
3. int temp;
4. temp = \*a ;
5. \*a = \*b ;
6. \*b = temp ;
7. **return** 0;
8. }

那么能否不使用中间变量temp来实现交换呢：

1. **int** swap1(**int** \*a, **int** \*b)
2. {
3. \*a = \*a+\*b;
4. \*b = \*a-\*b;
5. \*a = \*a-\*b;
6. **return** 0;
7. }

另外还有一种较为“高效”的算法，采用异或实现如下：

1. **int** swap2(**int** \*a, **int** \*b)
2. {
3. \*a = \*a^\*b;
4. \*b = \*a^\*b;
5. \*a = \*a^\*b;
6. **return** 0;
7. }

**实现原理：**

先将a，b当成两个布尔类型，那么a，b会有四种组合：

0,0 (1)a ^= b;变成 0,0 (2)b ^= a;变成 0,0 (3)a ^= b;变成 0,0

1,0 (1)a ^= b;变成 1,0 (2)b ^= a;变成 1,1 (3)a ^= b;变成 0,1

0,1 (1)a ^= b;变成 1,1 (2)b ^= a;变成 1,0 (3)a ^= b;变成 1,0

1,1 (1)a ^= b;变成 0,1 (2)b ^= a;变成 0,1 (3)a ^= b;变成 1,1

这样三句代码执行完成后，四种组合中的数值都得到了交换。

问题在于，上述的异或交换的算法实用性非常有限，它只能交换两个整数（包含char，int，long），要想交换两个浮点数是不行的，因为浮点数不能参与位运算，要想交换两个指针也是不行的，编译器不允许你把两个指针拿来做位运算，要想交换两个用户自定义对象也是不行的，因为它仍然不能参与位运算。那么是不是利用异或交换两个变量就没法用于浮点数、指针和用户自定义的对象了呢？

答案是否定的。实际上异或运算是针对二进制的，既然计算机所有的数据类型都是以二进制进行保存的，那么当然可以用异或运算交换任何数据类型。

既然位运算与BIT相邻数值无关的，那么8个BIT的char类型、16个BIT的short以及long，long long都可以使用^来交换。

实例探究异或交换对象算法

首先我们自定义一个Person类型的对象：

class Person{

public:

Person(int age ,const char\* name ):m\_Age(age)

{  
                   int len = strlen(name);  
                   this->m\_Name = new char[len+1];  
                   strcpy(this->m\_Name,name);  
         }

Person(){  
                   this->m\_Age = -1;  
                   this->m\_Name = 0;  
         }  
         void PrintSelf()  
         {  
                   cout<<this->m\_Name<<":"<<this->m\_Age<<endl;  
         }  
         Person& operator= (const Person& other)  
         {  
                   if (this == &other)  
                   {  
                            return \*this;  
                   }  
                   else  
                   {  
                            this->m\_Age = other.m\_Age;  
                            delete this->m\_Name;  
                            int len = strlen(other.m\_Name);  
                            this->m\_Name = new char[len+1];  
                            strcpy(this->m\_Name,other.m\_Name);  
                            return \*this;  
                   }  
         }  
         ~Person()  
         {  
                   delete this->m\_Name;  
         }  
private:  
         int m\_Age;  
         char\* m\_Name;  
};

定义如下指针：

Person youngMan(18,” young man”);

Person oldMan(81,” old man”);

Person\* pYoungMan = &youngMan;

Person\* pOldMan = &oldMan;

**利用异或实现两个float和指针的交换**

 之前的函数无法实现两个浮点数和指针的交换，其原因是浮点数和指针均不直接支持位运算。那么如何才能利用异或来交换两个浮点数和指针呢？方法仍然是“强制类型转换”。因为浮点数在内存中仍然是用一串二进制bit来表示的嘛，只要把浮点数看作（强制类型转换）二进制bit构成的整数，那么就能进行位运算了，至于指针嘛，处理方法完全相同。具体如何做呢，其实现大概是这样的：

template <class T>  
void Xor\_Swap\_2(T& a,T& b)  
{  
         \*((unsigned\*)(&a)) = \*((unsigned\*)(&a)) ^ \*((unsigned\*)(&b));  
         \*((unsigned\*)(&b)) = \*((unsigned\*)(&a)) ^ \*((unsigned\*)(&b));  
         \*((unsigned\*)(&a)) = \*((unsigned\*)(&a)) ^ \*((unsigned\*)(&b));  
}

利用这个函数可以交换两个float类型的变量，也可以交换任意类型的指针！非常值得注意的是：用它交换两个double类型数据或者两个Person类的对象(youngMan,oldMan)均能编译通过，但是其结果却是错的。测试如下：

Xor\_Swap\_2(youngMan,oldMan);

youngMan.PrintSelf();

oldMan.PrintSelf();

输出结果：

young man:81

old man:18

可以看出两个double数据并没被交换，**而两个Person对象在交换之后发生了怪异现象：产生了81岁的年轻人和18岁的老年人！**这一点正好说明强制类型转换时发生了位截断，由于Person类的第一个数据成员m\_Age正好是int型，在Xor\_Swap\_2内部做强制类型转换时正好取得了两个对象的m\_Age成员，于是出现了两个对象被部分交换的情况，那么又如何解释两个double数据没有变法呢？事实上两个double数据仍然发生了部分交换，因为这里的两个double数(a,b)的前4个字节正好相同，因此看不出部分交换。

既然我们知道了Xor\_Swap\_2为什么不能用于交换两个double类型的数据和两个用户自定义的数据，那么就有办法对它进行改进。具体改进的思想就是把参数按照一个byte一个byte地分别异或，按照这个思路函数实现如下：

template <class T>  
void Xor\_Swap\_3(T& a,T& b){  
         int size = sizeof(T);  
         for (int i = 0;i<size;i++) {  
                   \*((unsigned char\*)(&a)+i) = (\*((unsigned char\*)(&a)+i)) ^ (\*((unsigned char\*)(&b)+i));  
                   \*((unsigned char\*)(&b)+i) = (\*((unsigned char\*)(&a)+i)) ^ (\*((unsigned char\*)(&b)+i));  
                   \*((unsigned char\*)(&a)+i) = (\*((unsigned char\*)(&a)+i)) ^ (\*((unsigned char\*)(&b)+i));  
         }  
}

**这个版本的函数不仅能交换两个整数、任何指针、float数和double数，更厉害的是它能交换两个用户定义类型的变量！**事实上它基本上是在内存一级上操作数据，而任何类型的数据对象最终都表现为内存对象。这其实就是通过内存拷贝实现两个对象的交换的一个版本吧，当然还有利用memcpy等手段进行内存拷贝来实现两个变量的交换的，这里我就不赘述了。

异或交换算法中的陷阱

上文详细阐述了异或交换算法的实现。可是此算法中隐藏有一个陷阱，测试如下：

1. int main()
2. {
3. int a[2] = {1, 2};
4. swap(a[0], a[1]); //交换a[0]和a[1]的值
5. printf("1---a[0]=%d a[1]=%d\n", a[0], a[1]);
6. swap(a[0], a[0]); //将a[0]与自己进行交换
7. printf("2---a[0]=%d a[1]=%d\n", a[0], a[1]);
8. Return 0;
9. }
10. void swap(int &a, int &b)
11. {
12. a ^= b;
13. b ^= a;
14. a ^= b;
15. }

输出结果：

1---a[0]=2 a[1]=1

2---a[0]=0 a[1]=1

原因：

第一次的交换正确的执行了，但是第二次调用swap的时候却将a[0]置为了0. 仔细分析，不难发现，这正是我们在swap里面用异或实现交换所造成的。如果输入a和b是同一个数，swap里面代码相当于：

a ^= a;

a ^= a;

a ^= a;

成了a做了3次于自己的异或，其结果当然是0了。

出现这个问题，其实，是因为存储器别名的影响。如果，调用者传入的两个参数指向同一内存空间，那么就会出现意想不到的情况。编写程序的时候，要注意避免此类问题。

既然这样，我们就不能够在任何使用交换的地方采用异或了，即使要用，也一定要在交换之前判断两个数是否已经相等了，如下：

1. void exchange(int &a, int &b) {
2. if(a == b) return; //防止&a，&b指向同一个地址；那样结果会错误。
3. a ^= b;
4. b ^= a;
5. a ^= b;
6. }

写在后面

受一些过时的教科书的误导，有人认为程序里少用一个变量，节省一个字节的空间，会让程序运行更快。这是不对的，至少在这里不成立：

1.这个所谓的“技巧”在现代的机器上只会更慢（我甚至怀疑它从来就不可能比原始办法快）。原始办法是两次内存读和写，这个"技巧"是六读三写加三次异或（或许编译器可以优化成两读三写加三次异或）。

2.同样也不能节省内存，因为中间变量 tmp 通常会是寄存器（稍后有汇编代码供分析）。就算它在函数的局部堆栈(stack)上，反正栈已经开在那儿了，也没有进一步的函数调用，根本节约不了一丁点内存。

3.相反，由于计算步骤较多，会使用更多的指令，编译后的机器码长度会增加。

引用《代码简洁之道》中的一段话：

“我喜欢优雅和高效的代码。代码逻辑应当直截了当，叫缺陷难以隐藏；尽量减少依赖关系，使之便于维护；以某种全局策略一以贯之地处理全部出错情况；性能调校至接近最优，省得引诱别人实施无原则的优化(unprincipled optimizations)，搞出一团乱麻。整洁的代码只做好一件事。”

这恐怕就是Bjarne提及的没有原则的优化，甚至根本连优化都不是。代码的清晰性是首要的。

参考资料

<http://www.cnblogs.com/WhyEngine/p/4040246.html>

<http://blog.chinaunix.net/uid-1844931-id-3034714.html>

<http://www.cppblog.com/w57w57w57/archive/2011/08/10/152931.html>

<http://blog.csdn.net/rockics/article/details/7027949>

<http://blog.csdn.net/Solstice/article/details/5166912>