通常我们希望检查n 个不同元素的所有排列方式以确定一个最佳的排列。比如，

a，b 和c 的排列方式有：a b c, a c b, b a c, b c a, c a b 和c b a。

n 个元素的排列方式共有n ！种。

由于采用非递归的C + +函数来输出n 个元素的所有排列方式很困难，所以可以开发一个递

归函数来实现。令E= { ,..., }表示n 个元素的集合，我们的目标是生成该集合的所有排列方式。

令 为E中移去元素i 以后所获得的集合，perm (X) 表示集合X 中元素的排列方式，

.perm(X)表示在perm (X) 中的每个排列方式的前面均加上以后所得到的排列方式。

例如，如果

E= {a, b, c}，那么 = {b, c}，perm () = (bc, cb)，.perm () = (abc, acb)。

对于递归的基本部分，采用 n = 1。当只有一个元素时，只可能产生一种排列方式，所以

perm (E) = (e)，其中e 是E 中的唯一元素。当 n> 1时，perm (E) = .perm () + .perm

() + .perm( ) + ⋯ +.perm ()。

这种递归定义形式是采用n 个perm (X) 来定义perm (E),

其中每个X 包含n- 1个元素。至此，一个完整的递归定义所需要的基本部分和递归部分都已完成。

当n= 3并且E=（a, b, c）时，按照前面的递归定义可得

perm (E) =a.perm ( {b, c} ) +b.perm ( {a,c} ) +c.perm ( {a, b} )。

同样，按照递归定义有 perm ( {b, c} ) =b.perm ( {c} ) +c.perm ( {b}), 所以

a.perm ( {b, c} ) = ab.perm ( {c} ) + ac.perm ( {b}) = ab.c + ac.b = (abc, acb)。

同理可得

b.perm ( {a, c}) = ba.perm ({c}) + bc.perm ({a}) = ba.c + bc.a = (bac, bca)，

c.perm ( {a, b}) =ca.perm ({b}) + cb.perm ({a}) = ca.b + cb.a = (cab, cba)。

所以perm (E) = (abc, acb, bac, bca,cab,cba)。

注意a.perm ( {b, c} )实际上包含两个排列方式：abc 和a c b，a 是它们的前缀，perm ( {b, c} )

是它们的后缀。同样地，ac.perm ( {b}) 表示前缀为a c、后缀为perm ( {b}) 的排列方式。

程序把上述perm (E) 的递归定义转变成一个C++ 函数，这段代码输出所有前缀为list [ 0：

k-1], 后缀为list [ k：m] 的排列方式。调用Perm(list, 0, n-1) 将得到list[0: n-1] 的所有n! 个排列方式，在该调用中，k=0, m= n - 1，因此排列方式的前缀为空，后缀为list[0: n-1] 产生的所有排列方式。

当k =m 时，仅有一个后缀list[ m ]，因此list[0: m] 即是所要产生的输出。

当k<m时，先用list[k] 与list[ k：m] 中的每个元素进行交换，然后产生list[k+1: m] 的所有排列方式，并用它作为list[0: k] 的后缀。Swap是一个inline 函数，它被用来交换两个变量的值，其定义见程序。

Perm的正确性可用归纳法来证明。

使用递归函数生成排列  
template<class T>

void Perm(T list[], int k, int m)

{

/ /生成list [k：m ]的所有排列方式

int i;

if (k == m)

{

//输出一个排列方式

for (i = 0; i <= m; i++)

cout << list [i];

cout << endl;

}

else

// list[k：m ]有多个排列方式

// 递归地产生这些排列方式

for (i=k; i <= m; i++)

{

Swap (list[k], list[i]);

Perm (list, k+1, m);

Swap (list [k], list [i]);

}

}

template <class T>

inline void Swap(T& a, T& b)

{

// 交换a和b

T temp = a;

a = b;

b = temp;

}