C++ STL 全排列函数详解(排列组合与匹配算法)

一、概念

从n个不同元素中任取m（m≤n）个元素，按照一定的顺序排列起来，叫做从n个不同元素中取出m个元素的一个排列。当m=n时所有的排列情况叫全排列。如果这组数有n个，那么全排列数为n!个。

比如a，b，c的全排列一共有3！= 6 种 分别是{a, b, c}、{a, c, b}、{b, a, c}、{b, c, a}、{c, a, b}、{c, b, a}。

二、常用操作

1.头文件

```
#include <algorithm>
```

2.使用方法

这里先说两个概念：“下一个排列组合”和“上一个排列组合”，对序列 {a, b, c}，每一个元素都比后面的小，按照字典序列，固定a之后，a比bc都小，c比b大，它的下一个序列即为{a, c, b}，而{a, c, b}的上一个序列即为{a, b, c}，同理可以推出所有的六个序列为：{a, b, c}、{a, c, b}、{b, a, c}、{b, c, a}、{c, a, b}、{c, b, a}，其中{a, b, c}没有上一个元素，{c, b, a}没有下一个元素。

1）next_permutation：求下一个排列组合

a.函数模板：next_permutation(arr, arr+size);
b.参数说明：
arr： 数组名
size：数组元素个数
c.函数功能： 返回值为bool类型，当当前序列不存在下一个排列时，函数返回false，否则返回true，排列好的数在数组中存储

d.注意：在使用前需要对欲排列数组按升序排序，否则只能找出该序列之后的全排列数。
比如，如果数组num初始化为2,3,1，那么输出就变为了：{2 3 1} {3 1 2} {3 2 1}

2）prev_permutation：求上一个排列组合

a.函数模板：prev_permutation(arr, arr+size);
b.参数说明：
arr： 数组名
size：数组元素个数
c.函数功能： 返回值为bool类型，当当前序列不存在上一个排列时，函数返回false，否则返回true
d.注意：在使用前需要对欲排列数组按降序排序，否则只能找出该序列之后的全排列数。

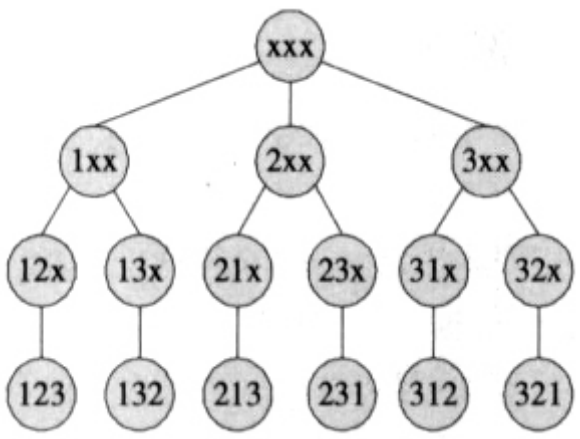
三、代码

```cpp
//  /* C++ STL 全排列函数详解   https://www.cnblogs.com/aiguona/p/7304945.html*/
#include <iostream>
#include <algorithm>
using namespace std;
int main ()
{
    int arr[] = {3,2,1};
    cout<<"用prev_permutation对3 2 1的全排列"<<endl;
    do
    {
        cout << arr[0] << ' ' << arr[1] << ' ' << arr[2]<<'\n';
    }
    while ( prev_permutation(arr,arr+3) );      ///获取上一个较大字典序排列，如果3改为2，只对前两个数全排列

    int arr1[] = {1,2,3};
    cout<<"用next_permutation对1 2 3的全排列"<<endl;
    do
    {
        cout << arr1[0] << ' ' << arr1[1] << ' ' << arr1[2] <<'\n';
```

```
    }     while ( next_permutation(arr1,arr1+3) );      ///获取下一个较大字典序排列，如果3改为2，只对前两个数全排列
    ///注意数组顺序，必要时要对数组先进行排序

    return 0;
}
```

四、全排列递归思路

我们可以将这个排列问题画成图形表示，即排列枚举树，比如下图为{1,2,3}的排列枚举树，此树和我们这里介绍的算法完全一致；



算法思路：

(1)n个元素的全排列=（n-1个元素的全排列）+（另一个元素作为前缀）；

(2)出口：如果只有一个元素的全排列，则说明已经排完，则输出数组；

(3)不断将每个元素放作第一个元素，然后将这个元素作为前缀，并将其余元素继续全排列，等到出口，出口出去后还需要还原数组；

C++ 全排列函数 std::next_permutation与std::prev_permutation

C++ STL中提供了std::next_permutation与std::prev_permutation可以获取数字或者是字符的全排列，其中std::next_permutation提供升序、std::prev_permutation提供降序。

1.std::next_permutation函数原型

template <class BidirectionalIterator>

bool next_permutation (BidirectionalIterator first, BidirectionalIterator last);

template <class BidirectionalIterator, class Compare>

bool next_permutation (BidirectionalIterator first,BidirectionalIterator last, Compare comp);

说明：next_permutation，重新排列范围内的元素[第一，最后一个）返回按照**字典序排列**的下一个值较大的组合。

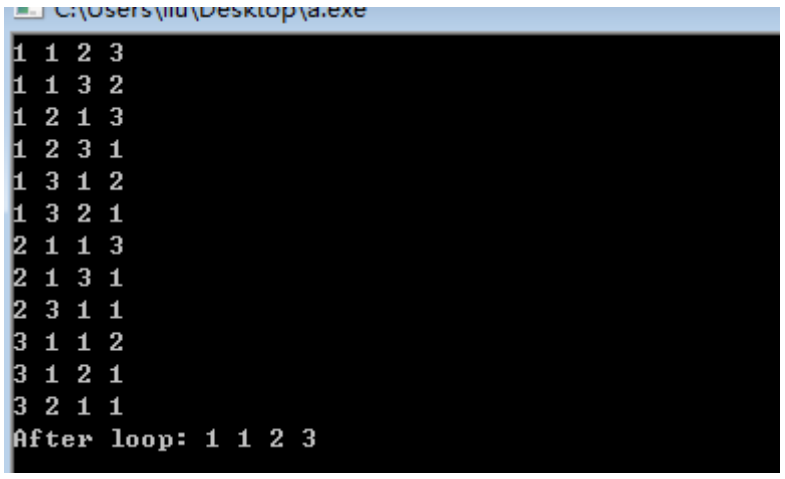
返回值：如果有一个更高的排列，它重新排列元素，并返回true；如果这是不可能的（因为它已经在最大可能的排列），它按升序排列重新元素，并返回false。

2.代码实例

```
#include <iostream>
#include <algorithm>    /// next_permutation, sort
using namespace std;
int main () {
```

```
  int myints[] = {1,2,3,1};
                              sort (myints,myints+4);

  do {
    cout << myints[0] << ' ' << myints[1] << ' ' << myints[2] << ' '<< myints[3]<<'\n';
  } while ( next_permutation(myints,myints+4) );    ///获取下一个较大字典序排列

  cout << "After loop: " << myints[0] << ' ' << myints[1] << ' ' << myints[2] << ' '<< myints[3] <<'\n';
  return 0;
}
```

输出：



3.算法实现原理

见：http://hi.baidu.com/bellgrade/item/70b65b8a7ea3c9c398255fd4

算法描述：

1、从尾部开始往前寻找**两个相邻**的元素

第1个元素i，第2个元素j（从前往后数的），且i<j

2、再从尾往前找第一个大于i的元素k。将i、k**对调**

3、[j,last)范围的元素置逆（**颠倒排列**）

c++排列组合函数

写个c++排列组合函数的使用以后使用得着的

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <boost/assign.hpp>
#include <boost/function.hpp>

using namespace std;
using namespace boost;
using namespace boost::assign;

inline void print_(int t){cout<<t<<" ";}
inline void print(vector<int>& vec)
{
    for_each(vec.begin(),vec.end(),print_);
    cout<<endl;
}

//! 全排列测试
void test1()
{
    vector<int> vec;
    vec += 1,2,3,4,5,6,7,8;
```

```
    sort(vec.begin(),vec.end());        int i = 0;
    do
    {
       print(vec);
       i++;
    }
    while(next_permutation(vec.begin(),vec.end()));
    std::cout<<i<<std::endl;//40320
}

//! 组合测试
size_t test2(int n,int m,boost::function<void(std::vector<int>& vec)> fn)
{
    vector<int> p,set;
    p.insert(p.end(),m,1);
    p.insert(p.end(),n-m,0);
    for(int i = 0;i != p.size();++i)
        set.push_back(i+1);
    vector<int> vec;
    size_t cnt = 0;
    do{
        for(int i = 0;i != p.size();++i)
            if(p[i])
               vec.push_back(set[i]);
        fn(vec);
        cnt ++;
        vec.clear();
    }while(prev_permutation( p.begin(), p.end()));
    return cnt;
}

int main()
{
//    test1();
    std::cout<<test2(20,3,print)<<std::endl;
    return 0;
}
```