**无重复元素的全排列**

1. **void** Permutation(**char**\* pStr,**int** k,**int** m)
2. {
3. assert(pStr);
5. **if**(k == m)
6. {
7. **static** **int** num = 1;  //局部静态变量，用来统计全排列的个数
8. printf("第%d个排列\t%s\n",num++,pStr);
9. }
10. **else**
11. {
12. **for**(**int** i = k; i <= m; i++)
13. {
14. swap(\*(pStr+k),\*(pStr+i));
15. Permutation(pStr, k + 1 , m);
16. swap(\*(pStr+k),\*(pStr+i));
17. }
18. }
19. }

**有重复元素的全排列**

//在[nBegin,nEnd)区间中是否有字符与下标为pEnd的字符相等

1. **bool** IsSwap(**char**\* pBegin , **char**\* pEnd)
2. {
3. **char** \*p;
4. **for**(p = pBegin ; p < pEnd ; p++)
5. {
6. **if**(\*p == \*pEnd)
7. **return** **false**;
8. }
9. **return** **true**;
10. }
11. **void** Permutation(**char**\* pStr , **char** \*pBegin)
12. {
13. assert(pStr);
15. **if**(\*pBegin == '\0')
16. {
17. **static** **int** num = 1;  //局部静态变量，用来统计全排列的个数
18. printf("第%d个排列\t%s\n",num++,pStr);
19. }
20. **else**
21. {
22. **for**(**char** \*pCh = pBegin; \*pCh != '\0'; pCh++)   //第pBegin个数分别与它后面的数字交换就能得到新的排列
23. {
24. **if**(IsSwap(pBegin , pCh))
25. {
26. swap(\*pBegin , \*pCh);
27. Permutation(pStr , pBegin + 1);
28. swap(\*pBegin , \*pCh);
29. }
30. }
31. }
32. }

**位运算求子集**

1. **void** print\_subset(**int** n , **int** s)
2. {
3. printf("{");
4. **for**(**int** i = 0 ; i < n ; ++i)
5. {
6. **if**( s&(1<<i) )         // 判断s的二进制中哪些位为1，即代表取某一位
7. printf("%c ",str[i]);   //或者a[i]
8. }
9. printf("}\n");
10. }
12. **void** subset(**int** n)
13. {
14. **for**(**int** i= 0 ; i < (1<<n) ; ++i)
15. {
16. print\_subset(n,i);
17. }
18. }

**八皇后**

1. **void** EightQueen( )
2. {
3. **const** **int** queens = 8;
4. **int** ColumnIndex[queens];
5. **for**(**int** i = 0 ; i < queens ; ++i)
6. ColumnIndex[i] = i;    //初始化
7. Permutation(ColumnIndex , queens , 0);
8. }
10. **bool** Check(**int** ColumnIndex[] , **int** length)
11. {
12. **int** i,j;
13. **for**(i = 0 ; i < length; ++i)
14. {
15. **for**(j = i + 1 ; j < length; ++j)
16. {
17. **if**( i - j == ColumnIndex[i] - ColumnIndex[j] || j - i == ColumnIndex[i] - ColumnIndex[j])   //在正、副对角线上
18. **return** **false**;
19. }
20. }
21. **return** **true**;
22. }
23. **void** Permutation(**int** ColumnIndex[] , **int** length , **int** index)
24. {
25. **if**(index == length)
26. {
27. **if**( Check(ColumnIndex , length) )   //检测棋盘当前的状态是否合法
28. {
29. ++g\_number;
30. Print(ColumnIndex , length);
31. }
32. }
33. **else**
34. {
35. **for**(**int** i = index ; i < length; ++i)   //全排列
36. {
37. swap(ColumnIndex[index] , ColumnIndex[i]);
38. Permutation(ColumnIndex , length , index + 1);
39. swap(ColumnIndex[index] , ColumnIndex[i]);
40. }
41. }
42. }
44. **void** Print(**int** ColumnIndex[] , **int** length)
45. {
46. printf("%d\n",g\_number);
47. **for**(**int** i = 0 ; i < length; ++i)
48. printf("%d ",ColumnIndex[i]);
49. printf("\n");
50. }

**输入两个整数n和m，从数列1,2,3...n中随意取几个数，使其和等于m，要求列出所有的组合。**

1. list<**int**> list1;
2. **void** find\_factor(**int** sum,**int** n)
3. {
4. //递归出口
5. **if**(n<=0||sum<=0)
6. **return**;
7. //输出找到的数
8. **if**(sum==n)
9. {
10. list1.reverse();
11. **for**(list<**int**>::iterator iter=list1.begin();iter!=list1.end();iter++)
12. cout<<\*iter<<"+";
13. cout<<n<<endl;
14. list1.reverse();
15. }
16. list1.push\_front(n);
17. find\_factor(sum-n,n-1);//n放在里面
18. list1.pop\_front();
19. find\_factor(sum,n-1);//n不放在里面
20. }