ACM选修课

STL编程及应用

STL中的几个基本概念

STL有三大核心部分：容器（Container）、算法（Algorithms）、迭代器（Iterator）。

容器：可容纳各种数据类型的数据结构。

迭代器：可依次存取容器中元素的东西

算法：用来操作容器中的元素的函数模板。例如，STL用sort()来对一个vector中的数据进行排序，用find()来搜索一个list中的对象。

函数本身与他们操作的数据的结构和类型无关，因此他们可以在从简单数组到高度复杂容器的任何数据结构上使用。

比如，数组int array[100]就是个容器，而int * 类型的指针变量就可以作为迭代器，可以为这个容器编写一个排序的算法

- 容器：装东西的东西，装水的杯子，装咸水的大海，装人的教室……STL里的容器是可容纳一些数据的模板类。
- 算法：就是往杯子里倒水，往大海里排污，从教室里撵人……STL里的算法，就是处理容器里面数据的方法、操作。
- 迭代器：往杯子里倒水的水壶，排污的管道，撵人的那个物业管理人员……STL里的迭代器：遍历容器中数据的对象。对存储于容器中的数据进行处理时，迭代器能从一个成员移向另一个成员。他能按预先定义的顺序在某些容器中的成员间移动。对普通的一维数组、向量、双端队列和列表来说，迭代器是一种指针。

- 容器（container）：容器是数据在内存中组织的方法，例如，数组、堆栈、队列、链表或二叉树（不过这些都不是STL标准容器）。
- STL中的容器是一种存储T（Template）类型值的有限集合的数据结构，容器的内部实现一般是类。这些值可以是对象本身，如果数据类型T代表的是Class的话。
- 算法（algorithm）：算法是应用在容器上以各种方法处理其内容的行为或功能。例如，有对容器内容排序、复制、检索和合并的算法。
- 在STL中，算法是由模板函数表现的。这些函数不是容器类的成员函数。相反，它们是独立的函数。令人吃惊的特点之一就是其算法如此通用。不仅可以将其用于STL容器，而且可以用于普通的C++数组或任何其他应用程序指定的容器。
- 迭代器(iterator)：一旦选定一种容器类型和数据行为(算法)，那么剩下唯一要他做的就是用迭代器使其相互作用。可以把迭代器看作一个指向容器中元素的普通指针。可以如递增一个指针那样递增迭代器，使其依次指向容器中每一个后继的元素。迭代器是STL的一个关键部分，因为它将算法和容器连在一起。

容器概述

- 可以用于存放各种类型的数据（基本类型的变量，对象等）的数据结构。
- 容器分为三大类：

1) 顺序容器

vector：后部插入/删除，直接访问
deque：前/后部插入/删除，直接访问
list：双向链表，任意位置插入/删除

2) 关联容器

set：快速查找，无重复元素
multiset：快速查找，可有重复元素
map：一对一映射，无重复元素，基于关键字查找
multimap：一对一映射，可有重复元素，基于关键字查找

前2者合称为第一类容器

3) 容器适配器

stack：LIFO
queue：FIFO
priority_queue：优先级高的元素先出

容器的共有成员函数

1) 所有标准库容器共有的成员函数：

- 相当于按词典顺序比较两个容器大小的运算符：
  =, <, <=, >, >=, ==, !=
- empty：判断容器中是否有元素
- max_size: 容器中最多能装多少元素
- size: 容器中元素个数
- swap: 交换两个容器的内容

迭代器

- 用于指向第一类容器中的元素。有const 和非 const两种。
- 通过迭代器可以读取它指向的元素，通过非const迭代器还能修改其指向的元素。迭代器用法和指针类似。
- 定义一个容器类的迭代器的方法可以是：

  容器类名::iterator 变量名;

或：

  容器类名::const_iterator 变量名;

- 访问一个迭代器指向的元素：

  * 迭代器变量名

向量（vector容器类）

- 向量（vector容器类）：
- #include <vector>，vector是一种动态数组，是基本数组的类模板。

- 支持随机访问迭代器，所有STL算法都能对vector操作。
- 随机访问时间为常数。在尾部添加速度很快，在中间插入慢。

例1：

```
int main()  {

        int a[5] = {1,2,3,4,5 };
        vector<int>  v(5);
        for(int i  = 0;i < v.size();i ++ )  v[i] = i;
        for(int i = 0;i < v.size();i ++ )
                cout << v[i] << "," ;
        cout << endl;
        vector<int> v2(a,a+5);  //构造函数
        v2.insert( v2.begin() + 2, 13 ); //在begin()+2位置插入 13
        for( int i = 0;i < v2.size();i ++ )
                cout << v2[i] << "," ;
        for(int i = 0; i < v2.size(); i++)
                v2.push_back(a[i]);
        return 0;
}
```

输出：

5

0,1,2,3,4,

1,2,13,3,4,5,

关联容器

- set, multiset, map, multimap
  - 内部元素有序排列，新元素插入的位置取决于它的值，查找速度快
  - map关联数组：元素通过键来存储和读取
  - set大小可变的集合，支持通过键实现的快速读取
  - multimap支持同一个键多次出现的map类型
  - multiset支持同一个键多次出现的set类型
- 与顺序容器的本质区别
  - 关联容器是通过键(key)存储和读取元素的
  - 而顺序容器则通过元素在容器中的位置顺序存储和访问元素。

Set功能

- Set，顾名思义，就是集合的意思
- 它支持插入，删除，查找，首元素，末元素等操作，就像一个集合一样。
- 而且所有的操作都是在严格logn时间之内完成，效率非常高。对于动态维护可以说是很理想的选择
- Set是一个有序的容器，里面的元素都是排好序的

Set操作

- base.begin() 求第1个元素，返回迭代器，由于是顺序的容器，所以第1个元素就是最小的元素.
  base.end() 求容器的末尾，表示容器到了末尾，返回迭代器，这个迭代器里没有元素，专门表示末尾.
  base.rbegin() 求倒数第1个元素，即最大元素，返回迭代器，（注意：这个迭代器是逆向迭代器和base.begin()的返回值类型不同）

Set操作

- base.size() 求容器里元素的数量，返回整数
  base.empty() 判断容器是否是空，空返回true,否则返回false;
  base.find(a) 查找元素a,如果查到了，返回指向a的迭代器，否则返回容器末尾迭代器，即base.end();
  base.count(a) 查找元素a的数量，返回整数

```
int main()  {
        int a[5] = {1,2,3,4,5 };
        set<int>S;
        for(int i  = 0;i < v.size();i ++ )  S.insert(i);
        for(set<int>::iterator it = S.begin(); it != S.end(); ++it)
                cout << *it<< "," ;
        cout << endl;
        S.erase(2);
        S.erase(S.begin());
        for(set<int>::iterator it = S.begin(); it != S.end(); ++it)
                cout << *it<< "," ;
        S.clear();
        return 0;
 }
```

UVA 10815 Andy's First Dictionary（stl，set）

题目：给出一串单词，把所有单词改小写去重按字典序输出。

思路：set可以解决去重和排序问题。（这点很关键，从小到大排序）
set中每个元素最多只出现一次
如何通过迭代器从小到大遍历所有元素

```
for (set::iterator i = d.begin(); i != d.end(); i++)
cout << *i << endl;
```

```
#include <iostream>
#include <string>
#include <set>
using namespace std;
set<string>jihe;
int main()
{
    string s;
    while(getline(cin,s))
    {
        string tmp;
        for(int i=0;i<s.size();i++)
        {
            if(s[i]>='A'&&s[i]<='Z')
            {
                s[i]=s[i]+32;
             }
if(s[i]>='A'&&s[i]<='Z')
 {
      s[i]=s[i]+32;
  }
 else if(s[i]>='a'&&s[i]<='z')
 {
       ///不用操作
  }
jihe.insert(s);
for(set<string>::iterator
it=jihe.begin();it!=jihe.end();it++)
{
       cout<<*it<<endl;
}
jihe.clear();
return 0;
}
```

对于内置类型自定义比较关系

- 对于内置的数据类型，如int，double等是不能重载<的，如果想自定义比较关系<可以用以下格式实现,以int为例：

```
struct cmp ( cmp这个名字可以用其他的 )
{
        bool operator()( const int &a, const int &b ) const
       { 定义比较关系 < }
};
set<int,cmp> base;
```

- 这样就创建了一个元素类型是int，自定义比较关系的，名字是base的set.

map

- 可以用pairs[key] 访形式问map中的元素。
  - pairs 为map容器名，key为关键字的值。
  - 该表达式返回的是对关键值为key的元素的值的引用。
  - 如果没有关键字为key的元素，则会往pairs里插入一个关键字为key的元素，并返回其值的引用
- 如：

map<int,double> pairs;
则 pairs[50] = 5; 会修改pairs中关键字为50的元素，使其值变成5

- 查找：find函数：来定位数据出现位置，它返回的一个迭代器，当数据出现时，它返回数据所在位置的迭代器，如果map中没有要查找的数据，它返回的迭代器等于end函数返回的迭代器.
- 删除：int n = Pairs.erase(50);//用关键字删除,如果删除了会返回1，否则返回0

Map使用例子

- #include <map>

  #include <string>
  using namespace std;
  map<string, string> TNmap;
  map<char,int> MAP;
  //建立映射
  TNmap["kitty"]="中国人";
  TNmap["john"]="美国人";
  TNmap["jack"]="日本人";

  //查找语句
  if(TNmap.find("jack") != TNmap.end( )){ … }

  //查找语句
  if(TNmap.count("jack") != 0){ … }

HDU 1004

Problem Description

Contest time again! How excited it is to see balloons floating around. But to tell you a secret, the judges' favorite time is guessing the most popular problem. When the contest is over, they will count the balloons of each color and find the result.

This year, they decide to leave this lovely job to you.

Input

Input contains multiple test cases. Each test case starts with a number N (0 < N <= 1000) -- the total number of balloons distributed. The next N lines contain one color each. The color of a balloon is a string of up to 15 lower-case letters.

A test case with N = 0 terminates the input and this test case is not to be processed.

Output

For each case, print the color of balloon for the most popular problem on a single line. It is guaranteed that there is a unique solution for each test case.

Sample Input

```
5
green
red
blue
red
red
3
pink
orange
pink
0
```

Sample Output

```
red
pink
```

题目是让统计出现 次数最多的字符串是哪一个。

```
#include <string>
#include <stdio.h>
#include <iostream>
#include <map>
using namespace std;
int main()
{
    int n;
    string ballon;
    while(cin >> n,n)
    {
        map<string, int> m;
        int N=n;
        while(N--)
        {
            cin >> ballon;
            ++m[ballon];
        }
        map<string,int>::iterator p;
        int maxx=0;
        string max_ballon;
        for(p=m.begin(); p!=m.end(); ++p)
        {
            if(p->second>maxx)
            {
                maxx=p->second;
                max_ballon=p->first;
            }
        }
        cout << max_ballon << endl;
    }
    return 0;
}
```

算法（algorithm）

- #inlcude <algorithm>
- STL中算法的大部分都不作为某些特定容器类的成员函数，他们是泛型的，每个算法都有处理大量不同容器类中数据的使用。值得注意的是，STL中的算法大多有多种版本，用户可以依照具体的情况选择合适版本。中在STL的泛型算法中有4类基本的算法：
- 变序型队列算法：可以改变容器内的数据；
- 非变序型队列算法：处理容器内的数据而不改变他们；
- 排序值算法：包涵对容器中的值进行排序和合并的算法，还有二叉搜索算法、通用数值算法。（注：STL的算法并不只是针对STL容器，对一般容器也是适用的。）
- 变序型队列算法：又叫可修改的序列算法。这类算法有复制（copy）算法、交换（swap）算法、替代（replace）算法、删除（clear）算法，移动（remove）算法、翻转（reverse）算法等等。这些算法可以改变容器中的数据（数据值和值在容器中的位置）

排序和查找算法

- Sort

  template<class RanIt>
  void sort(RanIt first, RanIt last);

- find

  template<class InIt, class T>
  InIt find(InIt first, InIt last, const T& val);

- binary_search 折半查找，要求容器已经有序

  template<class FwdIt, class T>
  bool binary_search(FwdIt first, FwdIt last, const T& val);

Sort 使用例子

```
#include <vector>
#include <algorithm>
#include <functional>     // For greater<int>( )
#include <iostream>

// Return whether first element is greater than the second
bool UDgreater ( int elem1, int elem2 )
{   return elem1 > elem2;  }

int main( )
{
  using namespace std;     vector <int> v1;
   vector <int>::iterator Iter1;

  int i;
  for ( i = 0 ; i <= 5 ; i++ )
  {       v1.push_back( 2 * i );       }

  int ii;
  for ( ii = 0 ; ii <= 5 ; ii++ )
  {       v1.push_back( 2 * ii + 1 );     }

  cout << "Original vector v1 = ( " ;
  for ( Iter1 = v1.begin( ) ; Iter1 != v1.end( ) ; Iter1++ )
    cout << *Iter1 << " ";
  cout << ")" << endl;

 sort( v1.begin( ), v1.end( ) );
  cout << "Sorted vector v1 = ( " ;
  for ( Iter1 = v1.begin( ) ; Iter1 != v1.end( ) ; Iter1++ )
    cout << *Iter1 << " ";
  cout << ")" << endl;

// To sort in descending order. specify binary predicate
  sort( v1.begin( ), v1.end( ), greater<int>( ) );
  cout << "Resorted (greater) vector v1 = ( " ;
  for ( Iter1 = v1.begin( ) ; Iter1 != v1.end( ) ; Iter1++ )
    cout << *Iter1 << " ";
  cout << ")" << endl;

  // A user-defined (UD) binary predicate can also be used
  sort( v1.begin( ), v1.end( ), UDgreater );
  cout << "Resorted (UDgreater) vector v1 = ( " ;
  for ( Iter1 = v1.begin( ) ; Iter1 != v1.end( ) ; Iter1++ )
    cout << *Iter1 << " ";
  cout << ")" << endl;
}
```

Output

```
Original vector v1 = ( 0 2 4 6 8 10 1 3 5 7 9 11 )
Sorted vector v1 = ( 0 1 2 3 4 5 6 7 8 9 10 11 )
Resorted (greater) vector v1 = ( 11 10 9 8 7 6 5 4 3 2 1 0 )
Resorted (UDgreater) vector v1 = ( 11 10 9 8 7 6 5 4 3 2 1
      0 )
```

HDU 1425

Problem Description

给你n个整数，请按从大到小的顺序输出其中前m大的数。

Input

每组测试数据有两行，第一行有两个数n,m(0<n,m<1000000)，第二行包含n个各不相同，且都处于区间[-500000,500000]的整数。

Output

对每组测试数据按从大到小的顺序输出前m大的数。

HDU 1425

Sample Input

```
5 3
3 -35 92 213 -644
```

Sample Output

```
213 92 3
```

Hint

请用VC/VC++提交

```
#include <iostream>
#include <cstdio>
#include <algorithm>

using namespace std;

int num[1000000+10];

int main()
{
    int n,m;
    while (~scanf("%d%d",&n,&m))
    {
        for (int i=1;i<=n;i++)
            scanf("%d",&num[i]);
        sort(num+1,num+n+1);
        for (int i=n;i>n-m+1;i--)
        {
            printf ("%d ",num[i]);
        }
        printf ("%d",num[n-m+1]);
        printf ("\n");
    }
    return 0;
}
```

sort

- sort 实际上是快速排序，时间复杂度 O(n*log(n));
  - 平均性能最优。但是最坏的情况下，性能可能非常差。

  如果要保证“最坏情况下”的性能，那么可以使用stable_sort
- stable_sort
  - stable_sort 实际上是归并排序（将两个已经排序的序列合并成一个序列），特点是能保持相等元素之间的先后次序
  - 在有足够存储空间的情况下，复杂度为 n * log(n)，否则复杂度为 n * log(n) * log(n)

- stable_sort 用法和 sort相同
  - 排序算法要求随机存取迭代器的支持，所以list 不能使用排序算法，要使用list::sort

谢谢！