

## 扑克排序

```
3 using namespace std;
4 struct node {
5     int shuzi, huase;
6     string pai;
7     bool operator <(const node &b) const {
8         if(shuzi != b.shuzi) return shuzi < b.shuzi;
9         return huase < b.huase;
10    }
11 };
12 node a[100001];
```

```
42 sort(a, a+n);
43 for(int i=0; i<n; i++) {
44     cout << a[i].pai << endl;
45 }
46 return 0;
```

```
16 cin >> n;
17 for(int i=0; i<n; i++) {
18     cin >> s; a[i].pai = s;
19     // 统一牌的单位
20     if(s[0] == 'J') {
21         a[i].shuzi = 11;
22     } else if(s[0] == 'Q') {
23         a[i].shuzi = 12;
24     } else if(s[0] == 'K') {
25         a[i].shuzi = 13;
26     } else if(s[0] == 'A') {
27         a[i].shuzi = 14;
28     } else {
29         a[i].shuzi = s[0] - '0';
30     }
31     // 花色
32     if(s[1] == 'D') {
```

## 练习： 《大米台灯》

```
3 struct node{
4     int id,nd,dm;
5     bool operator <(const node &x)const{//重载运算符
6         if(dm!=x.dm)return dm>x.dm;
7         if(nd!=x.nd)return nd<x.nd;
8         return id>x.id;
9     }
10    node(int ID,int ND,int DM){//重构
11        id=ID;nd=ND;dm=DM;
12    }
13 };
14 vector<node>a;
15 vector<int>res;
```



实验舱  
青少年编程  
走近科学 走进名校

# 实验舱蛟龙三班

Day9 stl 基础

zlj

2022.8

# Stl 介绍

部分数据结构与算法的封装，统一接口（命令方式），简化代码，  
NOIP NOI 都开放。（基本封装成容器）

**stl:**

**vector 、 string**

**map、 set** （关联式容器）

# 一、vector

## vector

`std::vector` 是 STL 提供的 内存连续的、可变长度 的数组（亦称列表）数据结构。能够提供线性复杂度的插入和删除，以及常数复杂度的随机访问。

# 常用函数操作汇总:

`v1.push_back()` //在数组的最后添加一个数据  
`v1.pop_back()` //去掉数组的最后一个数据  
`v1.front()` `v1.back()` //返回第一个元素(最后一个)  
`v1.begin()` //得到数组头的指针, 用迭代器接受  
`v1.end()` //得到数组的最后一个单元+1的指针, 用迭代器接受  
`v1.clear()` // 移除容器中所有数据  
`v1.empty()` //判断容器是否为空  
`v1.erase(pos)` //删除pos位置的数据  
`v1.erase(beg,end)` // 删除[beg,end)区间的数据  
`v1.size()` //返回容器中实际数据的个数  
`v1.insert(pos,data)` //在pos处插入数据

**`reverse(v1.begin(),v1.end())` //将v1容器数据倒置。**

**`Find(v.begin(),v.end(),3)` //查找v容器里3, 有返回迭 代器, 没有返回 `end()`**

- `clear()` 清除所有元素
- `insert()` 支持在某个迭代器位置插入元素、可以插入多个。复杂度与 `pos` 距离末尾长度成线性而非常数的
- `erase()` 删除某个迭代器或者区间的元素，返回最后被删除的迭代器。复杂度与 `insert` 一致。
- `push_back()` 在末尾插入一个元素，均摊复杂度为 常数，最坏为线性复杂度。
- `pop_back()` 删除末尾元素，常数复杂度。
- `swap()` 与另一个容器进行交换，此操作是 常数复杂度 而非线性的。



```
3  vector<int> a;
4  int main() {
5      for(int i=1;i<=5;i++){
6          a.push_back(i);
7      }
8      a.insert(a.begin()+2,9);
9      for(auto i:a){
10         cout<<i<<" ";
11     }
12     return 0;
13 }
```

## 4、Vector中结构体数据的读入

```
struct stu {  
    string xm;  
    int math;  
} x;  
vector<stu>a;  
int main() {  
    cin>>x.xm>>x.math;  
    a.push_back(x);  
}
```

# 重构函数

```
3 struct node {  
4     int id,nd,dm;  
5     node(int ID,int ND,int DM) { // 重构  
6         id=ID;  
7         nd=ND;  
8         dm=DM;  
9     }  
10 };  
11 vector<node>a;  
12 int main() {  
13     for(int i=1;i<=n;i++){  
14         int x,y;  
15         cin>>x>>y;  
16         a.push_back(node( i,x,y )); // 输入数据  
17     }  
18     return 0;
```

例： 《大米台灯》      《鼓掌总数》

## 二、set (集合关联容器)

set容器是一个关联容器，容器中的元素互不相同，称其为主键（或者关键字），并且容器中的元素自动排好序。每当插入或者删除一个元素时，容器都会重新排序，排序的复杂度为 $O(\log n)$ 。

set容器有两大特点，一个是元素唯一性，另一个有序，查询复杂度为 $O(\log n)$ （没有vector快，复杂度为 $O(1)$ ）。

set的定义：

```
set<int> s;    //定义int类型的set
```

```
set<int>::iterator it; //定义int类型set的迭代器
```

由于set是关联容器，内部元素不是按顺序存放，所以**只能用迭代器访问set容器中的元素。**

# Set（常用函数）

- s.insert( x )** 添加集合元素
- s.begin ( )** 返回指向第一个元素的迭代器
- s.end ( )** 返回指向最后一个元素之后的迭代器，不是最后一个元素
- s.clear ( )** 清除所有元素
- s.count ( )** 返回bool型,有返回1,无返回0
- s.empty()** 如果集合为空，返回true
- s.erase( )** 删除集合中的元素
- s.find()** 返回一个指向被查找到元素的迭代器，如果没找到则返回end()
- s.size()** 集合中元素的数目
- s.swap()** 交换两个集合变量

## 插入与删除操作

- `insert(x)` 当容器中没有等价元素的时候，将元素 `x` 插入到 `set` 中。
- `erase(x)` 删除值为 `x` 的所有元素，返回删除元素的个数。
- `erase(pos)` 删除迭代器为 `pos` 的元素，要求迭代器必须合法。
- `erase(first, last)` 删除迭代器在 `[first, last)` 范围内的所有元素。
- `clear()` 清空 `set`。



## 查找操作

- `count(x)` 返回 `set` 内键为 `x` 的元素数量。
- `find(x)` 在 `set` 内存在键为 `x` 的元素时会返回该元素的迭代器，否则返回 `end()`。
- `lower_bound(x)` 返回指向首个不小于给定键的元素的迭代器。如果不存在这样的元素，返回 `end()`。
- `upper_bound(x)` 返回指向首个大于给定键的元素的迭代器。如果不存在这样的元素，返回 `end()`。
- `empty()` 返回容器是否为空。
- `size()` 返回容器内元素个数。

# set的应用

set的使用举例：

```
set<int> s;    //定义int类型的set
set<int>::iterator it; //定义int类型set的迭代器， set
for(int i=0;i<10;i++){
    cin>>x;
    s.insert(x); //往set里插入数据
}
for(it=s.begin();it!=s.end();it++)cout<< *it<<" ";
```

insert ( x ) 函数会返回一个pair类型的值，其first是指向刚插入元素的迭代器。second是bool型表示插入是否成功。

**思考：**set元素是唯一的不重复，如果插入值x不在set中insert返回值是什么？如果x已经在set中返回值又是什么？

# set的应用

```
set<int> s;  
set<int>::iterator it;  
pair< set<int>::iterator ,bool> rst; //定义pair类型的的变量  
for(int i=0;i<10;i++){  
    cin>>x;  
    rst=s.insert(x); //往set里插入数据  
    if(rst.second==true) cout<<x<<" Insert success"<<endl;  
    else cout<<x<<" Insert fail"<<endl;  
}  
for(it=s.begin();it!=s.end();it++)cout<< *it<<" ";
```

我们可以根据second的值判断x是否已经存在,达到搜索中判重的目的。另外,插入重复值自动忽略的特点,可以用于去除重复的目的。

## 练习、《明明的随机数》

明明想在学校中请一些同学一起做一项问卷调查，为了实验的客观性，他先用计算机生成了N个1到1000之间的随机整数（ $N \leq 100$ ），对于其中重复的数字，只保留一个，把其余相同的数去掉，不同的数对应着不同的学生的学号。然后再把这些数从小到大排序，按照排好的顺序去找同学做调查。请你协助明明完成“去重”与“排序”的工作

**【输入格式】** 有2行，第1行为1个正整数，表示所生成的随机数的个数：N

第2行有N个用空格隔开的正整数，为所产生的随机数。

**【输出格式】** 第1行为1个正整数M，表示不相同的随机数的个数。

第2行为M个用空格隔开的正整数，为从小到大排好序的不相同的随机数。

**【输入样例】**

10

20 40 32 67 40 20 89 300 400 15

**【输出样例】**

8

15 20 32 40 67 89 300 400

**练习5、统计英文文章中出现了多少个不同的单词。**

**输入格式：一行全部由小写字母组成的英文文章，单词之间由空格隔开，无标点符号输**

**出格式：第一行文章中出现了多少个英文单词**

**接下来按英文单词字典序输出英文单词，每行一个**

**样例输入：**

**no pain no gain**

**样例输出：**

**3**

**gain**

**no**

**pain**

# set的常用函数总结：

set主要用于判重和去重，去重很简单不管什么往set里扔就好，反正它只保存一份，重复的值插不进去。**判重呢？**判重是在set中查找x，看x是否在set中，为此set给我们提供好几种查询的方法：

1、insert ( key )：往set中插入元素，返回pair类型值，first指向插入关键字的迭代器，second表示是否成功。若key已经在set中则second返回false。

适用于：**判断是否存在，如果不存在则插入**

我们经常**利用这个特点用于搜索状态的判重**：如果当前状态之前没有出现，则插入set中，并继续搜索下去；如果之前出现过，则跳过不用继续搜索。

2、count(key) : 返回key的数量，1或0，表示存在或不存在。根据其返回值判断key是否在set中。

```
set<int> s;
```

```
for(int i=0;i<10;i++){cin>>x; s.insert(x);}
```

```
cin>>x;
```

```
if(s.count(x)==1) cout<<"Yes"<<endl;
```

```
else cout<<"No"<<endl;
```

适用于：只需判断元素是否存在，不存在也不用插入。

# set的常用函数

3、find(key) : 返回指向key的迭代器，不存在返回end()；可以根据返回值是否等于end()判断是否存在于set中。

例如：set<int> s;

```
set<int>::iterator it;
```

```
for(int i=0;i<10;i++){cin>>x;s.insert(x);}
```

```
cin>>x;
```

```
it=s.find(x);
```

```
if(it!=s.end()) cout<<"Yes"<<endl;
```

```
else cout<<"No"<<endl;
```

适用于：只需判断元素是否存在，不存在也不用插入。



# set的常用函数

4、`clear()` : 清空set , 删除所有元素 , 时间复杂度 $O(n)$ ;

5、`empty()` :

6、`lower_bound ( key )` : 返回首个不小于 ( 大于等于 ) `key`的元素的迭代器

当`key`存在 , 返回指向`key`的迭代器;

若大于`key`的元素也不存在 , 则返回`end()`;

例如 : 2 3 6 8 9 10

`it=s.lower_bound(3);`it指向谁 ?

`it=s.lower_bound(4);`it又指向谁 ?

`it=s.lower_bound(11);`呢 ?

# set的常用函数

7、upper\_bound(key)：返回首个大于key的元素的迭代器；若不存在，返回end()。例如：2 3 6 8 10 13

it=s.upper\_bound(3);it指向谁？

it=s.upper\_bound(4);it又指向谁？

思考：it1=s.lower\_bound(3);

it2=s.upper\_bound(8);

区间[it1,it2)表示了哪些元素？

it1=s.lower\_bound(4);

it2=s.upper\_bound(9); [it1,it2)区间内又有哪些元素？

可以利用lower\_bound和upper\_bound组合起来表示某个区间内的元素

2	3	6	8	9	10
	↑			↑	
	it1			it2	

思考：[it1,it2)区间内有哪些元素？

# set的常用函数

8、erase 可以删除单个元素，参数**可以是删除元素的值**；也可以是**指向删除元素的迭代器**；还可以是**指向某个左闭右开的区间**，区间要用相应的迭代器，表示删除对应区间内的元素。

例如 set<int> s; 假设s中有：2 3 6 8 9 10 13 16 23

```
set<int>::iterator it;
```

```
s.erase(6); //元素6被删除
```

```
it=s.begin();
```

```
s.erase(it); //删除元素2。
```

```
s.erase(s.lower_bound(3),s.upper_bound(9));
```

思考：这里删除了哪些元素？

**注意：**由于插入、删除元素后，set会重新排序，各元素位置可能发生变化，所以已经赋值的迭代器变量会失效，需要重新赋值才能使用。

# 多重集合

类型名	作用	头文件
<code>set&lt;elemType&gt;name</code>	去重并且 排序	<code>#include&lt;set&gt;</code> <code>using namespace std;</code>
<code>multiset&lt;elemType&gt;name;</code>	只排序不 去重	<code>#include&lt;set&gt;</code> <code>using namespace std;</code>
<code>unordered_set&lt;elemType&gt; name;</code>	只去重不 排序	<code>#include&lt;unordered_set&gt;</code> <code>using namesp</code>

## 三、map (映射关联容器)

我们经常需要将一个值映射为另外一个值（映射就是两个元素的对应关系，**从一个可以找到另外一个**），常见的有：

一个数字映射一个字符串，例如：学号——姓名；

我们学过的数组其实就是一种映射，`a[5]="Chinese"`；这里是下标数字映射为字符串。

# map的定义

map是一种关联容器，它提供一对一的数据处理能力（其中第一个可以称为关键字，每个关键字只能在map中出现一次，第二个可以称为该关键字对应的值）。

map容器中的元素是根据关键字first大小排好序的，每当插入或者删除一个元素，容器都会重新排序，复杂度为 $O(\log n)$ 。

## 一、map定义

`map<int,string> p;` //int为关键字，string为该关键字对应的值。

`map<string,string> q;`

# map数据插入

## 二、数据插入

### 1、用insert函数插入pair类型数据（3种方式）

```
map<int,string> p
```

```
map<int,string>::iterator it;
```

```
p.insert(make_pair(1,"China"));
```

```
p.insert(pair<int,string>(5,"Japan")); //尖括号类变量类型要与map一致
```

```
p.insert(map<int,string>::value_type(3,"U.S.A"));
```

```
for(it=p.begin();it!=p.end();it++)
```

```
    cout<< it->first <<" "<< it->second <<endl;
```

# map数据插入

当map中有这个关键字时，insert操作是插入不了数据的，直接忽视这个insert操作，跟set类似这里insert也返回一个pair类型的值，first指向插入关键字的迭代器，second表示插入是否成功，为bool类型；可以检查second的值判断插入是否成功。

例如：map<int,string> p

```
pair< map<int,string>::iterator ,bool>rst;
```

```
rst=p.insert(make_pair(1,"China"));
```

```
if( rst.second==true )cout<<"1 Success"<<endl;
```

```
else cout<<"1 Fail"<<endl;
```

```
rst=p.insert(make_pair(1,"Japan"));
```

```
if(rst.second==true)cout<<"2 Success"<<endl;
```

```
else cout<<"2 Fail"<<endl; //思考：屏幕输出的内容是什么？
```



# map数据插入

2、用下标[ ]方式：**若关键字不存在，则添加；若存在则修改对应的值。**

例1：`map<int,string> p;`

`map<int,string>::iterator it;`

`p[1]="China";` //这里下标1表示关键字，p[1]是关键字1对应的值

`p[5]="Japan";` //下标不一定是连续的，与数组不同，仅表示关键字

`p[3]="U.S.A";`

`for(it=p.begin();it!=p.end();it++)`

`cout<<it->first<<" "<<it->second<<endl;`

例2：`map<string,int> p;`

`p["China"]=1;` //这里“China”就是关键字，1就是它对应的值

`p["Japan"]=5;`

# map数据插入

**注意：**用下标[]引用map元素，如果对应关键字不存在，则会插入该关键字，其second赋值为默认值。

例如：`map<int,int> p;`

```
cout<<p.size()<<endl;
```

```
cout<<p[2]<<" "<<p.size()<<endl;
```

输出显示：0

0 1

因为2不存在，所以把2作为关键字插入map中。所以为了统计某关键字出现的次数，可以直接用`p[2]++`;

# map数据插入

**insert和[]法都可以向map中插入数据，但二者还是有区别的。insert遇到重复关键字，忽略插入操作。但是用[]方式就不同了，它是覆盖原来的值。**

```
map<string,int> p;
```

```
p["China"]=2;
```

```
p["China"]=3;
```

```
cout<<p["China"]<<endl;
```

输出：3

# 容器的遍历:

## 1、迭代器进行遍历访问。

```
map<string,int>::iterator it;
```

```
for(it=p.begin();it!=p.end();it++)
```

```
    cout<<it->first<<" "<<it->second<<endl;
```

map元素是pair类型，it迭代器相当于指针

## 2、C++11 后支持的访问：

```
for( auto i : mp){
```

```
    cout<<i<<" " ;
```

```
}
```

# map实战应用

**例6、统计英文文章中出现了多少个不同的单词，以及这些单词出现的频率。**

**输入格式：一行全部由小写字母组成的英文文章，单词之间由空格隔开，无标点符号输**

**出格式：第一行文章中出现了多少个英文单词**

**接下来按英文单词字典序输出英文单词及该单词出现的次数，每行一个**

**样例输入：**

**no pain no gain**

**样例输出：**

**3**

**gain 1**

**no 2**

**pain 1**

# map数据元素查找

判断一个数据（关键字）是否在map中

1、count ( key ) : 返回0或1

```
map<int,int> p;
```

```
p[2]=3;
```

```
cout<<p.count(2)<<" "<<p.count(3)<<endl;
```

输出：1 0

2、find(key) : 若找到返回指向key的迭代器，否则返回end();

3、insert : 检查返回值的second

# map常用函数

- 1、**insert** 返回pair类型值，前一个指向关键字的迭代器，后一个表示是否成功
- 2、**count(key)**：返回key的数量，1或0，表示存在或不存在
- 3、**find(key)**：返回指向key的迭代器，不存在返回end()
- 4、**clear()**：清空map，删除所有元素
- 5、**empty()**：
- 6、**lower\_bound ( key )**：返回首个不小于（大于等于）key的元素的迭代器
- 7、**upper\_bound(key)**：返回首个大于key的元素的迭代器，若不存在，返回end()
- 8、**erase** 删除元素，可以删除单个元素，参数是关键字；也可以删除某个左闭右开区间的元素，参数是迭代器

# 例：统计单词后缀数

给你若干单词，请统计出以某个字符串为前缀的单词数量(单词本身也是自己的前缀)。

Input

输入数据的第一部分是一张单词表,每行一个单词,单词的长度不超过10,一个空行代表单词表提问,每个提问都是一个字符串。(提问不超过100)

注意:本题只有一组测试数据,处理到文件结束。

Output

对于每个提问,给出以该字符串为前缀的单词的数量。

Sample Input

```
banana
band
bee
absolute
acm

ba
b
band
abc
```

Sample Output

```
2
3
1
0
```



```
2  using namespace std; //统计单词前缀
3  char a;
4  map<string, int>st;
5  int main() {
6      string sum="";
7      while (1) {
8          scanf("%c",&a);
9          if (a=='\n') { //读到行末
10             scanf("%c",&a);
11             sum="";
12         }
13         if (a=='\n') { //读行末的下一位还是'\n'
14             break; //就退出
15         }
16         sum+=a;
17         st[sum]++;
18     }
19     while (cin>>sum) //输出
20         cout<<st[sum]<<endl;
21     return 0;
}
```

例题讲解：

## 〈DD老师的通话记录〉

分析:将电话号码当作关键字，用**map**统计即可

# 〈最懒的小朋友〉

分析:数据给出了没迟到的小朋友, 所以迟到的次数就是总次数减不迟到次数得到, 打擂台找答案即可

# map、set 区别

- 1、如果你想保证插入元素的唯一性，也就是你不想有重复值的出现，那么可以选择一个 **Set** 的实现类
- 2、如果你以键和值的形式进行数据存储那么 **Map** 是你正确的选择