

STL

C标准库

`strlen()` 字符串长度

`strcmp()` 字符串比较

`strcpy()` 字符串拷贝

`memset()` 暴力清空

`memcpy()` 暴力拷贝

三角函数、指数函数、浮点取整函数

`qsort()` C语言快排

`rand()` 随机数

`malloc()` `free()` C语言动态分配内存

`time(0)` 从1970年到现在的秒数（配合随机数）

`clock()` 程序启动到目前位置的毫秒数

`isdigit()`, `isalpha()`, 判断字符是否为数字、大小写字母

生成随机数

```
1 //生成随机数
2 #include<iostream>
3 #include<cstdlib>
4 #include<ctime>
5 using namespace std;
6 int main(){
7     srand(time(NULL)); //设置随机数种子
8     rand() % 100; //生成一个[0,100)的随机数
9     return 0;
10 }
```

动态数组 (vector)

成员函数

```
vector<type>v 创建动态数组v, 后面可以加{}或()进行初始化
type v[index] 获取v中第 index 个元素 O(1)
v.push_back(type item) 向v后面添加一个元素item O(1)
v.pop_back() 删除 v 最后一个元素 O(1)
v.size() 获取 v 中元素个数, 返回size_type类型 O(1)
v.resize(int n) 把 v 的长度设定为 n 个元素 O(n)
v.empty() 判断 v 是否为空, 空返回1, 不空返回0, O(1)
v.clear() 清空 v 中的元素 O(size)
v.insert(iterator it,type x)向迭代器it指向元素前增加一个元素x,O(n)
v.erase(iterator it) 删除向量中迭代器指向元素,O(n)
v.front()返回首元素的引用O(1)
v.back()返回尾元素的引用O(1)
v.begin()返回首迭代器, 指向第一个元素O(1)
v.end()返回尾迭代器, 指向向量最后一个元素的下一个位置O(1)
```

建立

```
1 //vector的创建
2 #include <bits/stdc++.h>
3 using namespace std;
4 int main()
5 {
6     vector<int>v1;
7     //创建一个存int类型的动态数组, int可以改成其它类型
8     vector<double>v2{1,1,2,3,5,8};
9     //创建一个存double类型的动态数组, 长度为6, 1 1 2 3 5 8分别存在v[0]~v[5]
10    vector<long long>v3(20);
11    //创建一个存long long类型的动态数组, 长度为20, v[0]~v[19]默认为0
12    vector<string>v4(20, "zzuacm");
13    //创建一个存string类型的动态数组, 长度为20, 存的都是"zzuacm"
14    vector<int>v5[3];
15    //相当于存int的二维数组, 一共3行, 每行的列可变
16    vector<vector<int> >v5{{1,2},{1},{1,2,3}};
17    //存int的二维数组, 行和列都可变, 初始状态
18    return 0;
19 }
```

操作

```
1 int main()
2 {
3     vector<int>v;
4     for(int i=1;i<=5;i++)
5         v.push_back(i); //向动态数组中插入1~5
6     cout<<v.size()<<endl; //输出数组的大小, 有几个值
7     for(int i=0;i<5;i++)
8         cout<<v[i]<<" "; //输出v[0]~v[4], 也就是1~5
9     cout<<endl;
10    v.clear(); //将v清空, 此时size为0
11    v.resize(10); //为v重新开辟大小为10的空间, 初始为0
12    for(int i=0;i<v.size();i++)
```

```

13     cout<<v[i]<<" "; //遍历每一个元素
14     while(!v.empty()) //当v还不空的话，去掉v的最后一个元素，等同于v.clear();
15         v.pop_back();
16     return 0;
17 }
```

删除 遍历

```

1 int main()
2 {
3     vector<int>v{0,1,2,3,4};
4     v.erase(v.begin() + 3); //删除v[3], v变为{0,1,2,4}
5     v.insert(v.begin() + 3, 666); //在v[3]前加上666, v变成{0,1,2,666,4}
6     v.front() = 10; //将v[0]改成10, 等同于v[0] = 10;
7     v.back() = 20; //将v[4]改成20等同于v[v.size() - 1] = 20;
8
9     //下标遍历
10    for(int i=0; i < v.size(); i++)
11        cout<<v[i]<<" "; //使用下标访问的方法遍历v
12    cout<<endl;
13
14    //迭代器遍历
15    for(vector<int>::iterator it=v.begin(); it != v.end(); it++)
16        cout<<*it<<" "; //使用迭代器，从v.begin()到v.end() - 1
17    for(auto i=v.begin(); i != v.end(); i++)
18        cout<<*i<<" "; //使用迭代器，从v.begin()到v.end() - 1
19    cout<<endl;
20    for(auto i:v) //使用C++11新特性循环遍历v, 如果需要改变i的值，还需要在前面加上&
21        cout<<i<<" ";
22
23    cout<<endl;
24    return 0;
25 }
```

迭代器

```

1 vector<int>::iterator it=v.begin(); //很奇怪的数据类型（对应STL的指针）
2 auto it=v.begin();
```

在这里 `it` 类似于一个**指针**，指向 `v` 的第一个元素

`it` 等价于 `&v[0]`

`*it` 等价于 `v[0]`

`it` 也可以进行加减操作，例如 `it + 3` 指向第四个元素

`it++` 后 `it` 指向的就是 `v` 的第二个元素(`v[1]`)了

字符串

成员函数

```
string str = "hello";
str.length(); str.size(); // O(n)
str.insert(1, "aaa"); // 在下标为1处插入一个字符或字符串 O(1)
str.insert(str.begin(), 'a'); // 在迭代器处插入一个字符或字符串 O(n)
str.c_str(); // 返回C语言字符串，用于printf O(n)
str.append(str2); // 把str2拼接到str后面 O(n)
str.compare(str2); // strcmp(str, str2)
str == str2; // strcmp(str, str2) == 0;
str += str2; // str.append(str2);
str += 'a'; // str.push_back('a');
```

- 所有参数为字符串的地方既可以是string也可以是c字符串
- 字符串操作与vector类似，但size,length复杂度较高
- 可通过下标访问字符串元素

1. 截取子串

s.substr(pos, n) 截取s中从pos开始（包括0）的n个字符的子串，并返回

s.substr(pos) 截取s中从从pos开始（包括0）到末尾的所有字符的子串，并返回

2. 替换子串

s.replace(pos, n, s1) 用s1替换s中从pos开始（包括0）的n个字符的子串

3. 查找子串

s.find(s1) 查找s中第一次出现s1的位置，并返回（包括0）

s.rfind(s1) 查找s中最后次出现s1的位置，并返回（包括0）

s.find_first_of(s1) 查找在s1中任意一个字符在s中第一次出现的位置，并返回（包括0）

s.find_last_of(s1) 查找在s1中任意一个字符在s中最后一次出现的位置，并返回（包括0）

s.find_first_not_of(s1) 查找s中第一个不属于s1中的字符的位置，并返回（包括0）

s.find_last_not_of(s1) 查找s中最后一个不属于s1中的字符的位置，并返回（包括0）

加速读取

```
1 #include<iostream>
2 //#include<string>
3 using namespace std;
4 int main()
5 {
6     string a;
7     char ch[100];
8     scanf("%s", ch);
9     a = string(ch);
10    cout<<a<<endl;
11    return 0;
12 }
```

栈

是一种线性结构

成员函数

头文件

```
queue<int> q;

q.front(); //返回 queue 中第一个元素的引用。
q.back(); //返回 queue 中最后一个元素的引用。
q.push(1); //在 queue 的尾部添加一个元素。
q.pop(); //删除 queue 中的第一个元素。
q.size(); //返回 queue 中元素的个数。
q.empty(); //如果 queue 中没有元素的话，返回 1，否则返回0。
```

队列

成员函数

创建方法:

queue<type>q; 建立一个存放数据类型为type的队列q

使用方法:

- `q.push(item)`: 在 q 的最后添加一个type类型元素item $O(1)$
- `q.pop()`: 使 q 最前面的元素出队 $O(1)$
- `q.front()`: 获取 q 最前面的元素 $O(1)$
- `q.size()`: 获取 q 中元素个数 $O(1)$
- `q.empty()`: 判断 q 是否为空, 空返回1, 不空返回0 $O(1)$

优先队列

优先队列是按照优先级出列的。每次的首元素都是优先级最大的。

优先队列的优先级是以定义的运算符 < 来说, 最大的那个元素。

`q.top()` 获取队列首位 (最大) 的元素

注意运算符的重载

map

05 / map

map是一类关联式容器。可以理解为以任意数据类型为下标 的超级数组。
需要加头文件<map>
初始化格式:
map<键的数据类型,值的数据类型>变量名

```
#include<iostream>
#include<map>
using namespace std;
int main()
{
    map<string,int>data;
    data["星期天"] = 7;
    cout<<data["星期天"]<<endl;
    return 0;
}
```

运行结果:

7

Map添加元素的方式也很简单。

格式:
变量名[键] = 值

内置pair, 见后

成员函数

```
mp.size(): 获取 mp 中元素个数 O(1)
mp.empty(): 判断 mp 是否为空 O(1)
mp.clear(): 清空 mp O(1)
mp.begin(): 返回 mp 中最小 key 的迭代器, 和set一样, 只可以用到--和++操作 O(1)
mp.end(): 返回 mp 中最大 key 的迭代器的后一个迭代器 O(1)
mp.find(key): 在 mp 中查找一个 key 并返回其 iterator, 找不到的话返回 mp.end() O(logn)
mp.count(key): 在 mp 中查找 key 的数量, 因为 map 中 key 唯一, 所以只会返回 0 或 1 O(logn)
mp.erase(key): 在 mp 中删除 key 所在的项, key 和 value 都会被删除 O(logn)
mp.lower_bound(item): 返回 mp 中第一个 key 大于等于 item 的迭代器, 找不到就返回 mp.end() O(logn)
mp.upper_bound(item): 返回 mp 中第一个 key 大于 item 的迭代器, 找不到就返回 mp.end() O(logn)
mp[key]: 返回 mp 中 key 对应的 value。如果 key 不存在, 则返回 value 类型的默认构造器
(defaultconstructor) 所构造的值, 并该键值对插入到 mp 中 O(logn)
mp[key] = xxx: 如果 mp 中找不到对应的 key 则将键值对 (key: xxx) 插入到 mp 中, 如果存在 key 则将这个 key 对应的值改变为 xxx O(logn)
```

遍历

```
1 #include<iostream>
2 #include<map>
3 using namespace std;
4 int main()
5 {
6     map<string,int>data;
7     data["星期天"] = 7;
8     data["星期六"] = 6;
9     data.insert(pair<string,int>("星期五",5));
10
11    for(map<string,int>::iterator it = data.begin();it!=data.end();it++){
12        cout<< it->first << " " << it->second << endl;
13    }
14
15    for(auto i:mp)
16        cout<<i.first<< ' '<<i.second<< endl;
17
18    return 0;
19 }
```

注意

- 有的题目在使用map时会卡时间，原因是map的访问添加都是O(nlogn)。
- 遇到这种情况，只需要使用unordered_map，unordered_map 的访问添加是O(1)。
- 除了初始化时写成unordered_map<键类型,值类型>变量名外，其他的操作都是一样的。
- map是有序的，unordered_map是无序的

pair

简易版struct

06 / pair

pair 是C++里内置的一种结构体，里面只有两个元素first 和 second。
除了定义时和普通结构体不一样以外，其他的操作都是一样的。

定义格式：

pair<键的数据类型,值的数据类型>变量名

这里的键就是第一个元素first，值就是第二个元素second。

```
#include<iostream>
using namespace std;
int main()
{
    pair<int,double>p;
    p.first = 1;
    p.second = 2;
    cout<<p.first<<endl;
    cout<<p.second<<endl;

    return 0;
}
```

运行结果：

```
1  
2
```

其实map就是一个储存pair的容器。
因此map的添加也可以通过insert一个pair来添加。

```
map<string,int>data;
data["星期天"] = 7;
data["星期六"] = 6;
data.insert(pair<string,int>("星期五",5));
```

构造

```
1 #define pii pair<int, int>
2 //#define x first
3 //#define y second
4 using pii = pair<int, int>;
5 typedef pair<int, int> pii;
6
7 int main(){
8     pii a (1, 2);
9     pii b = make_pair(3, 4);
10
11     pair<string, pii> c ("name", make_pair(1, 2));
12     auto d = c;
13     return 0;
14 }
```

set

- 集合(set)是一种包含对象的容器，可以快速地 ($\log n$) 查询元素是否在已知几集合中。
- set 中所有元素是有序地，且只能出现一次，因为 set 中元素是有序的，所以存储的元素必须已经定义过「 $<$ 」运算符 (因此如果想在 set 中存放 struct 的话必须手动重载「 $<$ 」运算符，和优先队列一样)
- 与set类似的还有
 - multiset元素有序可以出现多次
 - unordered_set元素无序只能出现一次 **unordered_map是乱序，不会按照插入元素序列排序**
 - unordered_multiset元素无序可以出现多次

成员函数

```
s.insert(item): 在 s 中插入一个元素 O(logn)
s.size(): 获取 s 中元素个数 O(1)
s.empty(): 判断 s 是否为空 O(1)
s.clear(): 清空 s O(n)
s.find(item): 在 s 中查找一个元素并返回其迭代器，找不到的话返回 s.end() O(logn)
s.begin(): 返回 s 中最小元素的迭代器，注意set中的迭代器和vector中的迭代器不同，无法直接加上某个数，因此要经常用到--和++O(1)
s.end(): 返回 s 中最大元素的迭代器的后一个迭代器 O(1)
s.count(item): 返回 s 中 item 的数量。在set中因为所有元素只能在 s 中出现一次，所以返回值只能是 0 或 1，在multiset中会返回存的个数 O(logn)
s.erase(position): 删除 s 中迭代器position对应位置的元素O(logn)
s.erase(item): 删除 s 中对应元素 O(logn)
s.erase(pos1, pos2): 删除 [pos1, pos2) 这个区间的位置的元素 O(logn + pos2 - pos1)
s.lower_bound(item): 返回 s 中第一个大于等于item的元素的迭代器，找不到就返回s.end() O(logn)
s.upper_bound(item): 返回 s 中第一个大于item的元素的迭代器，找不到就返回s.end() O(logn)
```

建立与遍历

```
1 //建立方法:
2 set<Type>s;
3 multiset<Type>s;
4 unorded_set<Type>s;
5 unorded_multiset<Type>s;
6 //如果Type无法进行比较，还需要和优先队列一样定义<运算符
7 //遍历方法:
8 for(auto i:s)cout<<i<<" ";
9 //和vector的类似
```

查找元素

```
1 set<int>s;
2 if(s.find(666) == s.end()){
3     cout << "666 was not in set";
4 }
5 else{
6     cout << *(s.find(666));
7 }
```

注意

```
1 int main(){
2     set<int> s;
3     auto i = s.begin();
4     i++, i++, i++, i++;
5     //i += 4;    错误
6     cout << *i;
7     return 0;
8 }
```

重载比较

set 容器模版需要3个泛型参数，如下：`template<class T, class C, class A> class set;`
第一个T是元素类型，必选；

第二个C指定元素比较方式，缺省为 Less, 即使用 < 符号比较；

第三个A指定空间分配对象，一般使用默认类型。

因此：

- (1) 如果第2个泛型参数你使用默认值的话，你的自定义元素类型需要重载 < 运算操作；
- (2) 如果你第2个泛型参数不使用默认值的话，则比较对象必须具有 () 操作，即：

```
bool operator()(const T &a, const T &b)
```

```
1 #include <cstdio>
2 #include <algorithm>
3 #include <set>
4 using namespace std;
5 int m,k;
6 struct cmp{
7     bool operator() (int a,int b){
8         if(abs(a-b)<=k){
9             return false;
10        }
11        else{
12            return a<b;
13        }
14    }
15 };
16 set<int,cmp> s;
17 char op[10];
18 int x;
19 int main(void){
20     scanf("%d%d",&m,&k);
21     while(m--){
22         scanf("%s%d",op,&x);
23         if(op[0]=='a'){
24             if(s.find(x)==s.end()){
25                 s.insert(x);
26             }
27         }
28         else if(op[0]=='d'){
29             s.erase(x);
30         }
31         else{
32             if(s.find(x)!=s.end()){
33                 puts("Yes");
34             }
35         }
36     }
37 }
```

```
34         }
35     }  
36     else{
37         puts("No");
38     }
39 }
40 return 0;
41 }
```

algorithm

sort()

sort(first, last, compare)

- first: 排序起始位置 (指针或 iterator)
- last: 排序终止位置 (指针或 iterator)
- compare: 比较方式, 可以省略, 省略时默认按升序排序, 如果排序的元素没有定义比较运算 (如结构体), 必须有compare
- sort 排序的范围是 [first, last), 时间为 O(nlogn)
- 作用: 使指定容器范围内的元素有序, 默认从**小到大**排序。
- 可以排序所有已经定义的数据类型

cmp()

对于未定义 < 小于号的数据类型, 可以写一个cmp函数来定义排序的规则。

同样的, 基本数据类型也可以通过cmp来自定义排序规则。

可更改升降序

```
1 #include<iostream>
2 #include<cstdlib>
3 #include<ctime>
4 #include<algorithm>
5 using namespace std;
6 const int n = 10;
7 struct st{
8     int A_score;
9     int B_score;
10 };
11 st stu[n+5];
12 void putt();//输出stu
13
14 //降序
15 bool cmp(st a,st b){//两个关键词的排序
16     if(a.A_score != b.A_score) return a.A_score > b.A_score;
17     return a.B_score > b.B_score;
18 }
19 //升序
20 bool cmp(st a,st b){//两个关键词的排序
21     if(a.A_score != b.A_score) return a.A_score < b.A_score;
22     return a.B_score < b.B_score;
23 }
24
```

```

25 int main()
26 {
27     srand(time(NULL));
28     for(int i = 0; i < n; i++){
29         stu[i].A_score = rand()%5 + 1;
30         stu[i].B_score = rand()%5 + 1;
31     }
32     cout<<"排序前: "<<endl;
33     putt();
34     sort(stu+0,stu+n,cmp);
35     cout<<endl<<"排序后: "<<endl;
36     putt();
37
38     return 0;
39 }
40 void putt(){
41     for(int i = 0; i < n; i++){
42         cout<<stu[i].A_score<< " "<< stu[i].B_score<<endl;
43     }
44 }
```

`std::sort(first,last,cmp);`

使用的范围是`[first,last)`

- 省略 `cmp`, 使用 `sort(first,last)`, 则默认从 **小到大排序**。
- 使用 `sort(first,last, greater())`, 则从 **大到小排序**。
- 如果是结构体或者自定义排序规则, 则需要自定义`cmp` 函数。
- 相等最好返回 `false`

cmp函数的含义, 如果返回值是 `True`, 表示要把序列 (X,Y) , X 放 Y 前。

```

1 bool cmp(int &x,int &y){
2     return x>y;//意味着x>y的时候, 把x放到y前, 按大到小排序。
3 }
```

重载<

重载结构体的排序规则

```

1 #include<iostream>
2 #include<cstdlib>
3 #include<ctime>
4 #include<algorithm>
5 using namespace std;
6 const int n = 10;
7 struct st{
8     int A_score;
9     int B_score;
10    bool operator < (const st b){
11        if(this->A_score != b.A_score) return this->A_score < b.A_score;
12        return this->B_score < b.B_score;
13    }
14 };
15 st stu[n+5];
```

```

16 void putt(); //输出stu
17 int main()
18 {
19     srand(time(NULL));
20     for(int i = 0; i < n; i++){
21         stu[i].A_score = rand()%5 + 1;
22         stu[i].B_score = rand()%5 + 1;
23     }
24     cout<<"排序前: "<<endl;
25     putt();
26     sort(stu+0, stu+n);
27     cout<<endl<<"排序后: "<<endl;
28     putt();
29
30     return 0;
31 }
32 void putt(){
33     for(int i = 0; i < n; i++){
34         cout<<stu[i].A_score<< " "<< stu[i].B_score<<endl;
35     }
36 }
```

next_permutation()

作用：用于求序列[first,last)元素全排列中一个排序的下一个排序

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 int main()
4 {
5     int a[3] = {0, 1, 2};
6     do
7     {
8         for (int i = 0; i < 3; i++)
9             cout<<a[i]<<' ';
10        cout<<endl;
11    }
12    while (next_permutation(a, a + 3));
13    return 0;
14 }
15 //如果有下一个排列 则返回1, 否则返回0 .
16 //即:对于 4 3 2 1, 按照字典序来说, 它没有下一个排列了, 返回0.
```

返回值：如果有下一个排列 则返回1，否则返回0。

二分函数

lower_bound(first, last, value)

- ▶ first: 查找起始位置 (指针或 iterator)
- ▶ last: 查找终止位置 (指针或 iterator)
- ▶ value: 查找的值
- ▶ lower_bound 查找的范围是 [first, last), 返回的是序列中第一个大于等于 value 的元素的位置, 时间为 O(logn)
- ▶ [first, last)范围内的序列必须是提前排好序的, 不然会错
- ▶ 如果序列内所有元素都比 value 小, 则返回last

```
upper_bound(first, last, value)
```

- upper_bound 与 lower_bound 相似，唯一不同的地方在于upper_bound 查找的是序列中第一个大于 value 的元素

```
1 int main()
2 {
3     int arr[]={3,2,5,1,4};
4     sort(arr,arr+5); //需要先排序
5     cout << *lower_bound(arr,arr+5,3); //输出数组中第一个大于等于3的值
6     return 0;
7 }
```

unique()

去重函数 (unique)

unique(first, last):

- [first, last)范围内的值必须是一开始就提前排好序的
- 移除 [first, last) 内连续重复项
- 返回值：去重之后的返回最后一个元素的下一个地址 (迭代器)

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 int main()
4 {
5     int arr[]={3,2,2,1,2},n;
6     sort(arr,arr+5); //需要先排序
7     n=unique(arr,arr+5)-arr; //n是去重后的元素个数
8     return 0;
9 }
```

reverse()

可反转容器等

```
1 string str="hello world , hi";
2 reverse(str.begin(),str.end()); //str结果为 ih , drow olleh
3 vector<int> v = {5,4,3,2,1};
4 reverse(v.begin(),v.end()); //容器v的值变为1,2,3,4,5
```

max()

min()

swap()

交换指针，可用于容器

取整函数

使用floor函数。 floor(x)返回的是小于或等于x的最大整数。

如： floor(10.5) == 10 floor(-10.5) == -11

使用ceil函数。 ceil(x)返回的是大于x的最小整数。

如： ceil(10.5) == 11 ceil(-10.5) == -10

floor()是向负无穷大舍入， floor(-10.5) == -11；

ceil()是向正无穷大舍入， ceil(-10.5) == -10

nth_element()

当采用默认的升序排序规则 (std::less) 时，该函数可以从某个序列中找到第 n 小的元素 K，并将 K 移动到序列中第 n 的位置处。不仅如此，整个序列经过 nth_element() 函数处理后，所有位于 K 之前的元素都比 K 小，所有位于 K 之后的元素都比 K 大。

```
1 | 3 4 1 2 5
2 | nth_element(s, s + n, s + len);
3 | 2 1 3 4 5
```

二分查找

在数组中的应用

在从小到大的排序数组中，

lower_bound(begin,end,num)：从数组的begin位置到end-1位置二分查找第一个大于或等于num的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。

upper_bound(begin,end,num)：从数组的begin位置到end-1位置二分查找第一个大于num的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。

在从大到小的排序数组中，重载lower_bound()和upper_bound()

lower_bound(begin,end,num,greater())：从数组的begin位置到end-1位置二分查找第一个小于或等于num的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。

upper_bound(begin,end,num,greater())：从数组的begin位置到end-1位置二分查找第一个小于num的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。

在map中的应用

lower_bound(k)寻找 k <= ? 并返回其迭代器

upper_bound(k)寻找 k < ? 并返回其迭代器

(其中 ? 为那个最接近的key值)

```
1 | // map::lower_bound/upper_bound
```

```
2 #include <iostream>
3 #include <map>
4
5 int main () {
6     std::map<char,int> mymap;
7     std::map<char,int>::iterator itlow,itup;
8
9     mymap['a']=20;
10    mymap['b']=40;    //注释看看
11    mymap['c']=60;
12    mymap['d']=80;
13    mymap['e']=100;
14
15    itlow=mymap.lower_bound ('b'); // 寻找 'b' <= ?
16    itup=mymap.upper_bound ('d'); // 寻找 'c' < ?
17
18    mymap.erase(itlow,itup);      // erases [itlow,itup)
19
20    // print content:
21    for (std::map<char,int>::iterator it=mymap.begin(); it!=mymap.end(); ++it)
22        std::cout << it->first << " => " << it->second << '\n';
23
24
25
26    return 0;
27 }
```