.C++ STL

1.vector

- #include<vector>
- vector<类型>Vectorname

v.front( )	返回对第一个元素的引用
v.back( )	返回对最后一个元素的引用
v.clear( )	清空 vector
v.empty( )	如果为空，返回 true，否则返回 false
v.begin( )	返回指向第一个元素的迭代器（iterator）
v.end( )	返回指向最后一个元素的迭代器
v.pop_back( )	删除 vector 的最后一个元素
v.push_back(value)	将 value 放到 vector 的最后
v.size( )	返回 vector 中元素的个数
v.rbegin( )	返回指向末尾的逆向迭代器
v.rend( )	返回指向开头之前位置的逆向迭代器
v.erase(loc); v.erase(start,end) ■ ( )中的均为 iterator，删除后元素前移	● 删除 loc 所指元素，并返回下一元素迭代器 ● 删除[start,end）元素，并返回最后被删除元素的下一个迭代器
v.insert(loc,value) v.insert(loc,num,value) v.insert(loc,start,end)	● 在 loc 位置插入一个 value 并返回其迭代器 ● 在 loc 位置插入 num 个 value ● 在 loc 位置插入[start,end)间的元素 ● 插入后元素均后移

v.erase(unique(v.begin( ),v.end( )), v.end( ));
v.erase(remove(v.begin( ), v.end( ), value),
v.end( ));
sort(v.begin( ), v.end( ))//要加 algorithm
	● 对 vector 进行排重
	● 删除 vector 中值为 value 的元素
	● 对 vector 进行从小到大排序,可加 cmp 函数

2.stack

- #include<stack>
- stack<类型名[,存储容器]>Stackname

s.empty( )	栈空返回 true，否则返回 false
s.pop ( )	移除堆栈中最顶层元素
s.push( )	压栈
s.size( )	返回当前栈中的元素个数
s.top( )	引用栈顶元素

3、queue

- #include<queue>

- queue<类型[, 存储容器]>QueueName

q.empty()	队列为空返回 true
q.pop()	删除队首元素
q.size()	返回当前队列中的元素数目
q.push(val)	将 val 加在队尾
q.back()	返回队尾元素的引用
q.front()	返回队首元素的引用

4、priority_queue

- #include<queue>
- priority_queue<类型[, 存储容器, 比较谓词]>PriorityQueueName
- 默认的比较方式是使用小于号运算符(<)进行比较，如果是系统提供的能够使用小于号比较的元素类型就可以只写元素类型；如果想用系统提供的大于号进行比较，则还需要使用自定义的 struct/class，则需要重载大于号运算符。

```
struct node{
    int i;
    bool operator<(const node &a){
        return (i < a.i);
    }
};
priority_queue<node> minNodeQ;
```

Pq.empty()	优先队列为空，返回 true
Pq.pop()	删除队首元素
Pq.push(val)	将 val 加到队尾，并排序
Pq.size()	返回当前队列中的元素数目
Pq.top()	返回队首元素的引用

5、list

- #include<list>
- list<类型>ListName

L.back()	返回对最后一个元素的引用
L.front()	返回对第一个元素的引用
L.begin()	返回指向第一个元素额迭代器
L.end()	返回指向链表末尾的迭代器
L.rbegin()	返回一个逆向迭代器，指向链表的末尾
L.rend()	返回个指向开头之前的逆向迭代器
L.clear()	清空链表
L.empty()	如果链表为空返回 true
L.pop_back()	删除链表的最后一个元素
L.pop_front()	删除链表的第一个元素
L.push_back(val)	将 val 连接到链表的最后
L.push_front(val)	将 val 连接到链表的头部

L.remove(val)	删除表中所有值为 val 的元素
L.size()	返回 list 中元素的个数
L.unique()	去除表中重复元素（离散化）
L.reverse()	将链表元素倒转
L.sort()/L.sort(cmp)	$n\log_2 n$ 由小到大排序，可自定义比较函数
L.erase(pos) L.erase(start,end) //pos、start、end 均为 iterator	● 删除 pos 所指元素，并返回下一元素迭代器 ● 删除[start,end]间的元素，并返回下一元素的迭代器
L.insert(pos,val) L.insert(pos,n,val) //pos 为 iterator	● 插入 val 在 pos 位置，并返回其迭代器 ● 插入 n 个 val 在 pos 位置

6、map/multimap

- #include<map>
- map<key 类型, value 类型>MapName;
- multimap<key 类型, value 类型>MultiMapName;
- 使用 element.first 和 element.second 来访问其中的值
- set/multiset：用法同 map
- #include<set>
- set<类型>SetName
- multiset<类型>SetName

m.begin()	返回指向第一个元素额迭代器
m.end()	返回指向末尾元素的迭代器
m.rbegin()	返回逆向迭代器，指向链表末尾
m.rend()	返回指向开头之前位置的迭代器
m.clear()	清空迭代器
m.empty()	如果为空，返回 true
m.size()	返回元素的数量
m.insert(pair<keytype,valuetype> val) m.insert(loc,pair<keytype, valuetype>val)	● 插入 pair 类型元素，对 map 返回一个 pair，first 指向插入元素的迭代器，second 表示插入是否成功 ● 从 loc 寻找一个可以插入值为 value 的元素的位置并将其插入返回 map
m.erase(loc) m.erase(start,end) m.erase(key_type key)	● 删除 loc 所指元素 ● 删除[start,end）之间的元素 ● 删除 key 值为 value 的元素，并返回删除的个数
m.find(key_type key)	● 返回一个迭代器指向键值为 key 的元素，未找到返回 end（）
m.lower_bound(key_type key) m.upper_bound(key_type key)	● 返回一个迭代器指向>=key 的第一个元素 ● 返回一个迭代器，指向>key 的第一个元素

key_compare key_comp(); value_compare value _comp();	● 返回一个比较key的函数。 ● 返回一个比较value的函数。

7. dequeue

■ #include<deque>

■ deque<类型>DequeName

d.back()	返回对最后一个元素的引用
d.front()	返回对第一个元素的引用
d.begin()	返回指向第一个元素的迭代器
d.end()	返回指向末尾的迭代器
d.rbegin()	返回一个逆向迭代器，指向链表的末尾
d.rend()	返回一个指向开头之前位置的迭代器
d.erase(pos) d.erase(start,end) //pos、start、end 均为 iterator	● 删除 pos 所指元素，并返回下一元素迭代器 ● 删除[start,end]间的元素,并返回下一元素的迭代器
d.insert(pos,val) d.insert(pos,n,val) //pos 为 iterator	● 插入 val 在 pos 位置，并返回其迭代器 ● 插入 n 个 val 在 pos 位置
d.pop_front()	删除第一个元素
d.pop_back()	删除最后一个元素
d.push_front(val)	将 val 放置到开头
d.push_back()	将 val 放置到最后

8.string

■ #include<string> //默认情况下包含了 iostream 就可以用

■ string StringName;

str.begin()	返回指向头部的迭代器
Str.end()	返回指向最后一个字符下一位置的迭代器
Str.clear()	清空 string，不回收空间
Str.empty()	返回 true 如果字符串长度为 0
Str.length()=str.size()	返回 string 的长度

Str.erase(loc);
str.erase(start,end);
str.erase(index = 0, size_type num = npos);
	·删除 loc 位置的字符
	·删除[start,end)之间的字符 ·删除从 index 开始的 num 个字符，返回*this

Str.find(str,index); //rfind 从逆向查找	返回从index开始str第一次出现的位置，找不到就返回
Str.insert(i,ch); str.insert(index,str);	● 在迭代器i所指位置插入ch ● 在index位置插入一个字符串
Str.push_back(c)	插入字符到末尾

删除排序后的重复字符 str.erase(unique(str.begin(), str.end()), str.end());

string的忽略大小写排序仿函数：

```
struct stringcmp{
bool operator()(const string &a, const string &b){
    for(unsigned int i=0; i<a.size() && i<b.size(); i++){
        if(toupper(a[i]) < toupper(b[i])){return true; }
        if(toupper(a[i]) > toupper(b[i])){ return false; }
    }
    return a.size() < b.size();
    }
};
set<string, stringcmp>ignoreCaseStringSet;
map<string, value_type, stringcmp>ignoreCaseStringMap;
```