vector库:

vector<T>a; vector<T>::iterator it;

|  |  |
| --- | --- |
| a.push\_back(x) | 尾部插入元素 |
| int sz=a.size() | 返回变长数组的大小 |
| it=a.begin(); it=a.end(); | (end是最后一个元素的下一个地址) |
| a.clear(); | 清空 |
| a.insert(a.begin()+k,x); | 在k号元素前插入一个x |
| a.erase(a.begin()+k); | 删除k号元素 |

stack库：

|  |  |
| --- | --- |
| stack.empty(); |  |
| stack.push(x); |  |
| stack.pop(); |  |
| while(!stack.empty()) stack.pop(); | 清空栈 |
| x=stack.top(); |  |
| num=stack.size(); |  |

queue库：

普通queue操作同栈，除了top()变成了front()函数

优先队列：priority\_queue<int>pq; 默认为最大队，即定义了小于后，大的排在前

|  |  |
| --- | --- |
| struct mydata  {  int x;  friend bool operator<(mydata one,mydata two)  {return one.x>two.x;}  }; | priority\_queue<mydata>pq;  这样就是最小队 |

操作完全同栈，也是用top(),唯一需要注意push和pop的时间复杂度为()

set库：

set<int>s，自动从小到大，自定义<过后可以用自己的数据类型

multiset<int>s，元素可以重复出现

|  |  |
| --- | --- |
| s.size() |  |
| set<int>::iterator it; |  |
| s.begin(); s.end(); |  |
| s.insert(x) |  |
| s.erase(x); s.erase(it); | 分别为删除值为x和it指向的元素，成功返回1，否则返回0 |
| s.clear() |  |
| it=s.find(x) | 查找，若it==s.end(),则不存在 |
| it=s.lower\_bound(x);it=s.upper\_bound(x); | 返回大于等于/大于x的第一个元素的迭代器，返回s.end()表示没有找到 |

map库：

|  |  |
| --- | --- |
| mp.size() |  |
| map<T1,T2>::iterator it; |  |
| mp.begin(); mp.end(); |  |
| mp[x]=y | 赋值 |
| mp.erase(x); mp.erase(it); | 同set,注意是第一关键字 |
| mp.clear(); |  |
| it=mp.find(x) | x为第一关键字 |
| mp.count(x) | x为第一关键字 |
| mp.lower\_bound(x),mp.upper\_bound(x) | 同set，不过均为第一关键字 |

string库:

string s;

|  |  |
| --- | --- |
| cin>>s | 输入 |
| s.c\_str() | 得到和char s[]等效的结果（用s的地方编译不过的时候就用这个） |
| s.begin();s.end(); |  |
| s=”xxx” | 直接赋值 |
| s.size(); s.length(); | 返回字符串长度 |
| s.clear() | 清空 |
| s.empty() | 判断是否为空 |
| s[2] | 返回第3个字母 |
| s+=s2, s.append(s2) | 表示添加字符串s2 |
| s.push\_back(x) | 末尾添加一个字母 |
| s.compare(s2) | 比较s和s2 |
| s+s2 | 把两个字符串拼接在一起 |
| s.find(s2) | 返回s2在s中的出现位置，找不到会返回-1 |