一、**vector 数组** 通过下标、迭代器、迭代器+整数访问

|  |  |
| --- | --- |
| push\_back() | push\_back(x)：在尾部插入值为x的元素 |
| pop\_back() | 删除尾部元素 |
| size() | 返回vector中的元素个数 |
| clear() | 清空vector中的所有元素 |
| insert() | insert(it,x)：在迭代器为it处插入一个值为x的元素 |
| erase() | （1）erase(it)：删除迭代器为it处的元素  （2）erase(first,last)：删除[first,last)区间的元素 |

二、**set 集合** 只能通过迭代器访问

|  |  |
| --- | --- |
| insert() | insert(x)：将x插入set中，并自动升序排序、去重复 |
| find() | find(value)：返回值为value对应元素的迭代器 |
| erase() | （1）erase(it)：删除迭代器为it处的元素  （2）erase(value)：删除值为value的元素  （3）erase(first,last)：删除[first,last)区间的元素 |
| size() | 返回set中的元素个数 |
| clear() | 情况set中的所有元素 |

三、**string 字符串** 通过下标、迭代器、迭代器+整数访问

|  |  |
| --- | --- |
| 拼接 | str3 = str1 + str2 ：将str1与str2拼接后，赋值给str3  str1 += str2 ：将str2拼接到str1后面 |
| 比较 | 按字典序进行 |
| length()/size() | 返回string的长度=字符数量 |
| insert() | （1）insert(pos,str)：在pos号位置插入字符串str  （2）insert(it,it\_first,it\_last)：it为原字符串欲插入位置，it\_first、it\_last为待插入字符串的.begin()/.end() |
| erase() | （1）erase(it)：删除迭代器为it处的元素  （2）erase(first,last)：删除[first,last)区间的元素  （3）erase(pos,length)：删除pos开始的length个字符 |
| clear() | 清空string中的数据 |
| substr() | substr(pos,length)：返回从pos号位开始，长度为length的子串 |
| string::npos | find()匹配失败后的返回值 |
| find() | （1）find(str2)：当str2是str子串时，返回str2在str中首次出现的位置；否则返回string::npos  （2）find(str2,pos)：从str的pos号位开始匹配str2，返回值同上 |
| replace() | （1）replace(pos,len,str2)：用str2替换str中从pos号位开始、长度为len的子串  （2）replace(it1,it2,str2)：用str2替换str中迭代器范围为[it1,it2)的子串 |

四、**map 映射** 通过下标（mp[type1]可以输出type2的值）、迭代器访问（it->first访问键、it->second访问值）

map<type1,type2> mp; ==>> 将type1的值映射到type2的值

|  |  |
| --- | --- |
| find() | find(key)：返回键值type1==key的迭代器 |
| erase() | （1）erase(it)：删除迭代器为it处的元素  （2）erase(key)：删除键值为key的元素  （3）erase(first,last)：删除[first,last)区间的元素 |
| size() | 获得map中映射对的个数 |
| clear() | 清空map |

五、**queue 队列** 通过q.front()来访问队首元素、q.back()来访问队尾元素

|  |  |
| --- | --- |
| push() | push(x)：将x入队，压入队尾 |
| front()/back() | 访问队首元素/队尾元素 |
| pop() | 出队，弹出队首元素 |
| empty() | 检测queue是否为空 |
| size() | 返回queue的元素个数 |

六、**priority\_queue 优先队列** 只能用top()访问队首元素（优先级最高的元素）

堆排序实现，保证队首元素必为优先级最高的那个

|  |  |
| --- | --- |
| push() | push(x)：将x入队 |
| top() | 获得队首元素 |
| pop() | 令队首元素出队 |
| empty() | 检测优先队列是否为空 |
| size() | 返回优先队列中元素的个数 |

优先级的设置问题：

|  |
| --- |
| 对于基本数据类型  默认的priority\_queue<int> q; == priority\_queue<int,vector<int>,less<int> >  如果相要修改优先级，则只需要把less(递减)改成greater(递增)  即priority\_queue<int,vector<int>,greater<int> > |
| 对于结构体  重载“<”  (1)价格高的优先级高：  struct fruit{  string name;  int price;  friend bool operator < (fruit f1,fruit f2){  return f1.price < f2.price;  }  };  (2)价格低的优先级高：  struct fruit{  string name;  int price;  friend bool operator < (fruit f1,fruit f2){  return f1.price > f2.price;  }  }; |

七、**stack 栈** 只能通过top()来访问栈顶元素

|  |  |
| --- | --- |
| Push() | Push(x)：将x压入栈顶 |
| Top() | 获得栈顶元素 |
| Pop() | 弹出栈顶元素 |
| Empty() | 检测stack是否为空 |
| Size() | 返回stack中元素个数 |