**STL容器之vector**

1. **作用**
2. Vector 属于动态数组。可根据下表访问某一元素。序列容器。
3. 头文件 include<vector> 。
5. **常用函数**
6. 声明一个变量

Vector<int> vec ;

1. vec.size(); 数组大小。
2. vec.empty(); 容器是否为空，如果为空则返回1 ，不为空返回0。
3. Vec.push\_back(xx) ; 在数组尾端添加元素xx 。xx：为变量名。
4. Vec.erase(xx); 删除某一元素。参数为一个迭代器常量。
5. vec.begin(); 返回第一个元素的迭代器。
6. vec.end() ; 返回最后一个元素的迭代器。
7. Vec.clear(); 清除所有元素。
8. **遍历**

Vector<int>::iterator it;

It = vec.begin();

while(it!=vec.end())

{

Xxxxx;

在循环中删除的时候，一定不能删除以后再自加。

Vec.erase(it++); （不可以）

It = vec.erase(it); (可以，再删除的时候会返回下一个元素的迭代器)

Vec.erase(it); it++; (不可以)

It++;

}

1. **优缺点**

典型的序列容器，C++标准严格要求次容器的实现内存必须是连续的唯一可以和标准c兼容的stl容器，任意元素的读取，修改具有常数时间复杂度，在序列尾部进行插入，删除是常数时间复杂度，但在序列的头部插入，删除的时间复杂度是O(n)，可以在任何位置插入新元素，有随机访问功能，插入删除操作需要考虑。

对大数据量的操作，如果操作内存，会重新分配两倍的内存，并将旧数据拷贝过去，会导致效率特别慢。

**STL容器之map**

**一 作用**

1. 是STL的一个关联容器，存储键值对。
2. [] 符号可以根据key ，获取到对应的value 。
3. include<map>
4. Map 不能插入相同key的元素。表现为插入失败。
5. 如果[] 取的key 不存在，则返回空。

**二 常用函数**

1. 定义

Map<int ,string> \_map;

1. \_map.insert( pair<int,string>(1,”apple”)); 插入某个元素。
2. \_map.insert(map<int, string>::value\_type (1, "student\_one"));  插入
3. \_map.size(); map中元素个数。
4. \_map.erase(); 删除某个元素，参数为迭代器。
5. \_map.erase(Key&key); 通过key去删除某元素。
6. \_map[1] = “apple” ; 修改key为“1” 的元素的值。如果没有该key，则 插入这对元素。
7. \_map.begin(); 返回第一个元素的迭代器。
8. \_map.end(); 返回第二个元素的迭代器。
9. \_map.empty(); 若map为空，则返回1，map不为空返回0。
10. \_map.clear(); 清除所有元素。

**三 遍历**

Map<int,string>::iterator it = \_map.begin();

While(it!=\_map.end())

{

Xxx;

遍历时删除元素。

\_map.erase(it++) ; (可以)

It = \_map.erase(it); (可以)

It++;

}

**四 优缺点**

1. 插入和删除快速。2.6所示。
2. 存取方便。
3. 查找时间复杂度为O(logN)

**STL容器之SET**

**一 作用**

1. 内部基于红黑树实现。集合，集合里的元素都是唯一的。关联容器。
2. 包含头文件 include<set>
3. 不能使用[]。集合中的元素自动排序。
4. 不能插入相同元素，

**二 常用函数**

1. 初始化

Set<string> \_set;

1. \_set.begin(); 获取排序最前面的元素，不是第一个插入的元素。
2. \_set.end(); 排序最后面的元素，不是插入的第一个元素。
3. \_set.clear(); 删除所有元素。
4. \_set.insert(); 插入元素。 参数为元素类型。
5. \_set.erase(); 删除某个元素，参数为迭代器。
6. \_set.erase(); 删除某个元素，参数为值。
7. \_set.size(); 返回集合大小。

**三 遍历**

set<int>::iterator it = \_set.begin();

While(it!= \_set.end())

{

遍历时删除元素

It = \_set.erase(it); (可以)

\_set.erase(it++); (可以)

It++;

}

**四 优缺点**

1.查找时间复杂度是O(logN)

**STL容器之deque**

**一 作用**

1. 序列式容器，双端队列，采用动态数组来管理元素。
2. 头文件 include<deque>
3. []可以取出对应序号的元素。

**二 常用函数**

1. 初始化

Deque<int> \_deq;

1. \_deq.begin() ; 返回指向第一个元素的迭代器。
2. \_deq.end(); 返回指向最后一个元素的迭代器。
3. \_deg.empty(); 如果队列为空，返回1。否则返回0.
4. \_deq.push\_back(); 在队尾插入元素。参数为元素类型。
5. \_deq.pop\_back(); 在队尾删除元素。没有参数
6. \_deq.insert(pos,xx); 在某个位置插入元素xx. Pos为迭代器类型。
7. \_deq.erase(pos); 删除某个位置的元素。 pos为迭代器类型。
8. \_deq.push\_front(xx); 在开头插入元素。Xx为元素类型。
9. \_deq.pop\_front(); 删除开头位置元素。
10. \_deq.clear(); 清空元素。

**三 遍历**

deque<int>::iterator it =\_deq.begin();

while(it!=\_deq.end())

{

if(\*it==5)

{

it = \_deq.erase(it); (可以)

\_deq.erase(it++); (不可以)

}

it++;

}

**四 优缺点**

1.容器deque和vector非常相似，属于序列式容器。都是采用动态数组来管 理元素，提供随机存取，并且有着和vector一样的接口。不同的是deque具 有首尾两端进行快速插入、删除的能力。（dqeue有点像list 和vector的结 合体）。

2. 如果 只需要高效的随即存储，而不在乎插入和删除的效率，使用vector.

3. 如果只需要大量的插入和删除，不关心随即存取，则应使用list。

4. 如果既需要随即存取，有关心两端的数据的插入和删除，则应使用deque。

5. 在序列的头部插入，删除的时间复杂度是O(n)

**STL容器之list**

**一 作用**

1. 双向链表。
2. 头文件 #include<list>
3. 不能使用[] 取某一个元素。

**二 常用函数**

1. 初始化

List \_list;

1. \_list.begin(); 返回第一个迭代器。
2. \_list.end(); 返回最后一个元素的迭代器。
3. \_list.push\_back(); 在尾部插入数据。参数为数据类型。
4. \_list.push\_front(); 在头部插入数据。参数为数据类型。
5. \_list.empty(); 判空，如果队列为空返回1，否则返回0；
6. \_list.clear(); 清空队列。
7. \_list.size(); 队列大小。
8. \_list.pop\_back(); 删除尾部元素。 （确保list不为空）
9. \_list.pop\_front(); 删除头部元素。 （确保list不为空）
10. \_list.erase(pos); 删除某个位置的元素。Pos为迭代器类型。
11. \_list.insert(pos,xx); 在pos处插入元素xx. Pos为迭代器类型。

**三 遍历**

list<int>::iterator it = \_list.begin();

while(it!=\_list.end())

{

if(\*it==3)

{

\_list.erase(it++);（可以）

it = \_list.erase(it);（可以）

}

it++;

}

**四 优缺点**

1. 不能使用[]， 存取不是很方便。
2. 内存必须是连续的。
3. 任意元素的读取，修改具有常数复杂度，在序列尾部进行插入，删除是常数时间复杂度。但在序列的头部插入，删除的时间复杂度是O(n) ，可以在任何位置插图新元素。

红黑树

Skiplist