**一. Set 的特点**

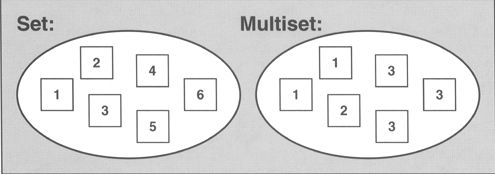
Set 和 multiset 会根据特定的排序准则，自动将元素排序。两者不同处在于multiset允许元素重复而set不允许重复。

**优点：**

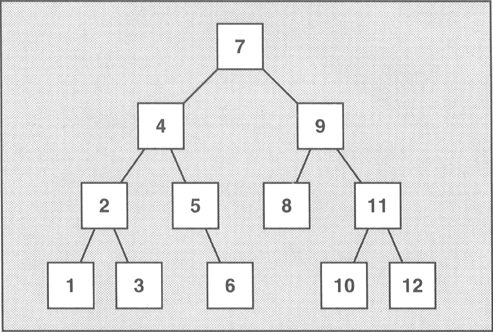
在搜索元素时具有良好的性能。

**限制：**

你不能直接改变元素值，因为这样会打乱原本正确的顺序，因此要改变元素值，必须先删除旧元素，再插入新元素。



Set 和 multiset 通常以平衡二叉树来实现，如下图：



**二. 排序准则**

Set<Elem, op>，其中op为排序准则，缺省准则是less，即从小到大排序。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7312433)

1. std::set<**int**, std::greater<**int**> > coll; //从大到小
2. std::set<**int**, std::less<**int**> > coll; //从小到大
3. std::set<**int**> coll; //默认less<>，也是从小到大

std::set<int, std::greater<int> > coll; //从大到小

std::set<int, std::less<int> > coll; //从小到大

std::set<int> coll; //默认less<>，也是从小到大

**三. 搜索函数**

由于 set 和 multiset 在元素快速搜索方面有优化设计，所以提供了特殊的搜索函数：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7312433)

1. //返回元素值为elem的元素个数
2. s.count(elem)
3. //返回第一个元素值为 elem 的位置，找不到的话返回end()
4. s.find(elem)
5. //返回 elem 的第一个可以安插的位置，也就是 (元素值 >=elem)的位置
6. s.lower\_bound(elem)
7. //返回 elem 的最后一个可以安插的位置，也就是(元素值 > elem)的位置
8. s.upper\_bound(elem)
9. //返回 elem 可以安插的第一个位置和最后一个位置，也就是(元素值 == elem) 的元素区间
10. s.equal\_range(elem)

//返回元素值为elem的元素个数

s.count(elem)

//返回第一个元素值为 elem 的位置，找不到的话返回end()

s.find(elem)

//返回 elem 的第一个可以安插的位置，也就是 (元素值 >=elem)的位置

s.lower\_bound(elem)

//返回 elem 的最后一个可以安插的位置，也就是(元素值 > elem)的位置

s.upper\_bound(elem)

//返回 elem 可以安插的第一个位置和最后一个位置，也就是(元素值 == elem) 的元素区间

s.equal\_range(elem)

**四. 插入和删除函数**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7312433)

1. //插入elem元素，返回新元素的位置，pos 指出插入操作的搜寻起点，如果 pos 恰当可以加快速度
2. c.insert(pos,elem)
3. c.insert(elem)
4. //删除与elem相等的元素，返回删除元素的个数
5. c.erase(elem)
6. c.erase(pos)
7. c.erase(beg, end)

//插入elem元素，返回新元素的位置，pos 指出插入操作的搜寻起点，如果 pos 恰当可以加快速度

c.insert(pos,elem)

c.insert(elem)

//删除与elem相等的元素，返回删除元素的个数

c.erase(elem)

c.erase(pos)

c.erase(beg, end)

**注意插入操作的返回值：**

因为 set 不允许重复，面 multiset 允许重复，所以他们的 insert 操作有不有同的返回值。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7312433)

1. //set 提供的接口
2. pair<iterator, **bool**> insert(const value\_type& elem); //返回 pair<>

//set 提供的接口

pair<iterator, bool> insert(const value\_type& elem); //返回 pair<>

因为 set 不允许元素重复，所以如果插入相同的元素，将会返回失败。

Pari 的 **secode** 成员表示插入是否成功。

Pair 的 **first** 成员返回新元素的位置。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7312433)

1. //multiset 提供的接口
2. iterator insert(const value\_type& elem); //返回新元素的位置

//multiset 提供的接口

iterator insert(const value\_type& elem); //返回新元素的位置

**注意删除操作：**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7312433)

1. //删除与 elem 相等的元素，返回删除元素的个数
2. c.erase(elem)

//删除与 elem 相等的元素，返回删除元素的个数

c.erase(elem)

我们可以看到，其它容器的 erase() 操作参数都是位置。而set 除了其它几个erase()，还有这个传入值的 erase() 函数操作。这个操作在 list 里应该对应的是，c.remove()。但是这里确没有一至，可能是历史原因吧，我也不知道，但是这个要注意。反正我们记住，在 set 里有一个 erase(elem)，在 list 里有一个 remove() 操作就行了。

如果 multisets 内含有重复元素，你不能使用 erase()，来删除这些重复元素中的第一个，你可以这么做：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7312433)

1. std::multiset<Elem> coll;
2. ...
3. //remove first element with passed value
4. std::multiset<Elem>::iterator pos;
5. pos = coll.find (elem);
6. if (pos != coll.end())
7. {
8. coll.erase(pos);
9. }

std::multiset<Elem> coll;

...

//remove first element with passed value

std::multiset<Elem>::iterator pos;

pos = coll.find (elem);

if (pos != coll.end())

{

coll.erase(pos);

}

**五. 示例代码**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7312433)

1. #include <iostream>
2. #include <set>
3. using namespace std;
4. **int** main()
5. {
6. /\*type of the collection:
7. \*-no duplicates
8. \*-elements are integral values
9. \*-descending order
10. \*/
11. typedef set<**int**,greater<**int**> > IntSet;
12. IntSet coll1; // empty set container
13. //insert elements in random order
14. coll1.insert(4);
15. coll1.insert(3);
16. coll1.insert(5);
17. coll1.insert(1);
18. coll1.insert(6);
19. coll1.insert(2);
20. coll1.insert(5);
21. //iterate over all elements and print them
22. IntSet::iterator pos;
23. for (pos = coll1.begin(); pos != coll1.end(); ++pos) {
24. cout << \*pos << ' ';
25. }
26. cout << endl;
27. //再次插入 4
28. pair<IntSet::iterator, **bool**> status = coll1.insert(4);
29. if (status.second) {
30. cout << "4 inserted as element "
31. << distance (coll1.begin(),status. first) + 1
32. << endl;
33. }
34. else {
35. cout << "4 already exists" << endl;
36. }
37. //assign elements to another set with ascending order
38. set<**int**> coll2(coll1.begin(),
39. coll1.end());
40. //print all elements of the copy
41. copy (coll2.begin(), coll2.end(),
42. ostream\_iterator<**int**>(cout," "));
43. cout << endl;
44. //remove all elements up to element with value 3
45. coll2.erase (coll2.begin(), coll2.find(3));
46. //remove all elements with value 5
47. **int** num;
48. num = coll2.erase (5);
49. cout << num << " element(s) removed" << endl;
50. //print all elements
51. copy (coll2.begin(), coll2.end(),
52. ostream\_iterator<**int**>(cout," "));
53. cout << endl;
54. return 0;
55. }