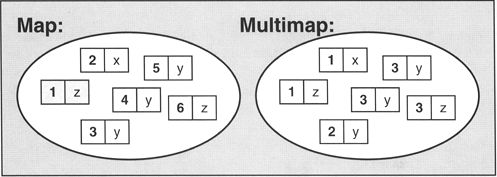
**一. Map与 Multimap 特点**

Map 和 Multimap 将(key/value) 一组当做元素，它们可以根据 key 的排序准则自动将元素排序。



Multimap允许重复元素，map不允许重复。

类型定义：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7313204)

1. typedef std::map<std::string, **float**, std::greater<string> > StringFloatMap;
2. ...
3. StringFloatMap coll;

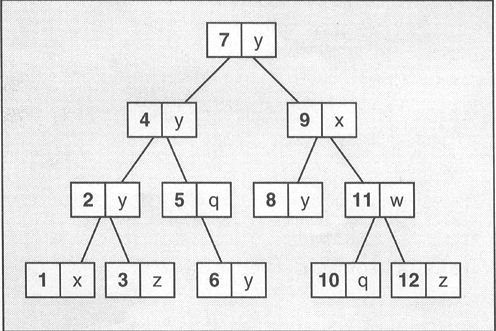
typedef std::map<std::string, float, std::greater<string> > StringFloatMap;

...

StringFloatMap coll;

注意，两个>之间需加上一个空格，因为>>会被编译器视为移位操作符。

Map 和 multimap 通常以平衡二叉树来实现，map 和 multimap 拥有 set 和 multiset 所有能为和所有操作函数。



通常，**不能直接改变key的值**，value 的值是可以直接修改的，因此可以想像元素的实质类型是 pair<**const** key, T>。

二. **搜索函数**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7313204)

1. //返回键值等于 key 的元素个数
2. m.count(key)
3. //返回键值等于 key 的第一个元素，找不到就返回 end()
4. m.find(key)
5. //返回 键值 >= key 的第一个元素位置
6. m.lower\_bound(key)
7. //返回 键值 >= key 的第一个元素位置
8. m.upper\_bound(key)
9. //返回 键值 == key 的元素区间
10. m.equal\_range(key)

//返回键值等于 key 的元素个数

m.count(key)

//返回键值等于 key 的第一个元素，找不到就返回 end()

m.find(key)

//返回 键值 >= key 的第一个元素位置

m.lower\_bound(key)

//返回 键值 >= key 的第一个元素位置

m.upper\_bound(key)

//返回 键值 == key 的元素区间

m.equal\_range(key)

**注意：**

所有的搜索函数，**参数是key**，而不是value。这样你就不能以 find() 搜寻拥有某特定 value 的元素。

**三. 插入和删除函数**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7313204)

1. //插入
2. c.insert(elem)
3. c.insert(pos,elem)
4. c.insert(beg,end)
5. //删除
6. c.erase(elem)
7. c.erase(pos)
8. c.erase(beg,end)

//插入

c.insert(elem)

c.insert(pos,elem)

c.insert(beg,end)

//删除

c.erase(elem)

c.erase(pos)

c.erase(beg,end)

操作和 set 一样，不同的是**参数是value**，而不是key。这里要区别于上面搜索函数的参数。

**注意：**

删除元素时，会使 pos 无效，如下

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7313204)

1. //使迭代器失效，注意！！！！
2. typedef std::map<std::string,**float**> StringFloatMap;
3. StringFloatMap coll;
4. StringFloatMap::iterator pos;
5. ...
6. for (pos = coll.begin(); pos != coll.end(); ++pos)
7. {
8. if (pos->second == value) {
9. coll. erase (pos); // 出错 !!!
10. }
11. }
12. //正确处理迭代器所指元素的方法
13. typedef std::map<std::string,**float**> StringFloatMap;
14. StringFloatMap coll;
15. StringFloatMap::iterator pos, tmp\_pos;
16. ...
17. //remove all elements having a certain value
18. for (pos = c.begin(); pos != c.end(); )
19. {
20. if (pos->second == value) {
21. c.erase(pos++); // Make clear!!
22. }
23. else {
24. ++pos;
25. }
26. }

//使迭代器失效，注意！！！！

typedef std::map<std::string,float> StringFloatMap;

StringFloatMap coll;

StringFloatMap::iterator pos;

...

for (pos = coll.begin(); pos != coll.end(); ++pos)

{

if (pos->second == value) {

coll. erase (pos); // 出错 !!!

}

}

//正确处理迭代器所指元素的方法

typedef std::map<std::string,float> StringFloatMap;

StringFloatMap coll;

StringFloatMap::iterator pos, tmp\_pos;

...

//remove all elements having a certain value

for (pos = c.begin(); pos != c.end(); )

{

if (pos->second == value) {

c.erase(pos++); // Make clear!!

}

else {

++pos;

}

}

pos++，是这样执行的，首先pos会移向下一个元素，但pos++返回的还是原来的pos的位置。

**四. 将 map 视为关联数组**

m[key]，这里下标是元素**key**，索引可以是任意类型，而非局限为整型。

**注意：**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7313204)

1. coll["otto"] = 7.7;

coll["otto"] = 7.7;

处理 coll["otto"] 时，如果存在键值 "otto"，则返回元素的引用，如果不存在键值 "otto"，则会自动安插一个新元素，键值为 "otto"，value以default实现。

不小心会引起误操作，如：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7313204)

1. std::cout << coll ["ottto"]; // otto拼写错了

std::cout << coll ["ottto"]; // otto拼写错了

不会有错误提示，反而会插入一个新的元素。

**五. 将 value 值传入 map**

**1. 运用 value\_type，value\_type 是容器本身提供的类型定义。**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7313204)

1. std::map<std::string,**float**> coll;
2. ...
3. coll.insert(std::map<std::string, **float**>::value\_type("otto", 22.3));

std::map<std::string,float> coll;

...

coll.insert(std::map<std::string, float>::value\_type("otto", 22.3));

注，系统中 value\_type 的定义如下：

typedef pair<const Key, T> value\_type;

**2. 运用 pair< >**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7313204)

1. std::map<std::string, **float**> coll;
2. ...
3. //隐式转换为const:
4. coll.insert(std::pair<std::string, **float**>("otto",22.3));
5. //显示转换:
6. coll.insert(std::pair<<strong>const</strong> std::string, **float**>("otto",22.3));

std::map<std::string, float> coll;

...

//隐式转换为const:

coll.insert(std::pair<std::string, float>("otto",22.3));

//显示转换:

coll.insert(std::pair<**const** std::string, float>("otto",22.3));

**3. make\_pair( )**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7313204)

1. std::map<std::string,**float**> coll;
2. ...
3. coll.insert(std::make\_pair("otto",22.3));

std::map<std::string,float> coll;

...

coll.insert(std::make\_pair("otto",22.3));

注：以上三种方法，在实际开发中可以试着用，因为有些编译器并不能很好的支持其中方法，比如 VC6.0 就不能很好的使用 make\_pair()，不过改用value\_type是可以的。