C++ STL中Map的按Key排序和按Value排序

map是用来存放<key, value>键值对的**数据结构**，可以很方便快速的根据key查到相应的value。假如存储学生和其成绩（假定不存在重名，当然可以对重名加以区分），我们用map来进行存储就是个不错的选择。 我们这样定义，map<string, int>，其中学生姓名用string类型，作为Key；该学生的成绩用int类型，作为value。这样一来，我们可以根据学生姓名快速的查找到他的成绩。

但是，我们除了希望能够查询某个学生的成绩，或许还想看看整体的情况。我们想把所有同学和他相应的成绩都输出来，并且按照我们想要的顺序进行输出：比如按照学生姓名的顺序进行输出，或者按照学生成绩的高低进行输出。换句话说，我们希望能够对map进行按Key排序或按Value排序，然后按序输出其键值对的内容。

一、C++ STL中Map的按Key排序

其实，为了实现快速查找，map内部本身就是按序存储的（比如红黑树）。在我们插入<key, value>键值对时，就会按照key的大小顺序进行存储。这也是作为key的类型必须能够进行<运算比较的原因。现在我们用string类型作为key，因此，我们的存储就是按学生姓名的字典排序储存的。

【参考代码】

```
#include<map>
#include<string>
#include<iostream>
using namespace std;

typedef pair<string, int> PAIR;

ostream& operator<<(ostream& out, const PAIR& p) {
  return out << p.first << "\t" << p.second;
}

int main() {
  map<string, int> name_score_map;
  name_score_map["LiMin"] = 90;
  name_score_map["ZiLinMi"] = 79;
  name_score_map["BoB"] = 92;
  name_score_map.insert(make_pair("Bing",99));
  name_score_map.insert(make_pair("Albert",86));
  for (map<string, int>::iterator iter = name_score_map.begin();
       iter != name_score_map.end();
       ++iter) {
    cout << *iter << endl;
  }
  return 0;
}
```

【运行结果】

```
Albert  86
Bing    99
BoB     92
LiMin   90
ZiLinMi 79
```

大家都知道map是stl里面的一个模板类，现在我们来看下map的定义：

```
template < class Key, class T, class Compare = less<Key>,
        class Allocator = allocator<pair<const Key,T> > > class map;
```

它有四个参数，其中我们比较熟悉的有两个: Key 和 Value。第四个是 Allocator，用来定义存储分配模型的，此处我们不作介绍。

现在我们重点看下第三个参数： class Compare = less<Key>

这也是一个class类型的，而且提供了默认值 less<Key>。 less是stl里面的一个函数对象，那么什么是函数对象呢？

所谓的函数对象：即调用操作符的类，其对象常称为函数对象（function object），它们是行为类似函数的对象。表现出一个函数的特征，就是通过“对象名+(参数列表)”的方式使用一个 类，其实质是对operator()操作符的重载。

现在我们来看一下less的实现：

```
template <class T> struct less : binary_function <T,T,bool> {
  bool operator() (const T& x, const T& y) const
    {return x<y;}
};
```

它是一个带模板的struct，里面仅仅对()运算符进行了重载，实现很简单，但用起来很方便，这就是函数对象的优点所在。stl中还为四则运算等常见运算定义了这样的函数对象，与less相对的还有greater：

```
template <class T> struct greater : binary_function <T,T,bool> {
  bool operator() (const T& x, const T& y) const
    {return x>y;}
};
```

map这里指定less作为其默认比较函数(对象)，所以我们通常如果不自己指定Compare，map中键值对就会按照Key的less顺序进行组织存储，因此我们就看到了上面代码输出结果是按照学生姓名的字典顺序输出的，即string的less序列。

我们可以在定义map的时候，指定它的第三个参数Compare，比如我们把默认的less指定为greater：

【参考代码】

```
#include<map>
#include<string>
#include<iostream>
using namespace std;

typedef pair<string, int> PAIR;

ostream& operator<<(ostream& out, const PAIR& p) {
```

```
9    return out << p.first << "\t" << p.second;
10 }
11
12 int main() {
13   map<string, int, greater<string> > name_score_map;
14   name_score_map["LiMin"] = 90;
15   name_score_map["ZiLinMi"] = 79;
16   name_score_map["BoB"] = 92;
17   name_score_map.insert(make_pair("Bing",99));
18   name_score_map.insert(make_pair("Albert",86));
19   for (map<string, int>::iterator iter = name_score_map.begin();
20        iter != name_score_map.end();
21        ++iter) {
22     cout << *iter << endl;
23   }
24   return 0;
25 }
```

【运行结果】

```
ZiLinMi 79
LiMin   90
BoB     92
Bing    99
Albert  86
```

现在知道如何为map指定Compare类了，如果我们想自己写一个compare的类，让map按照我们想要的顺序来存储，比如，按照学生姓名的长短排序进行存储，那该怎么做呢？

其实很简单，只要我们自己写一个函数对象，实现想要的逻辑，定义map的时候把Compare指定为我们自己编写的这个就ok啦。

```
1  struct CmpByKeyLength {
2    bool operator()(const string& k1, const string& k2) {
3      return k1.length() < k2.length();
4    }
5  };
```

是不是很简单！这里我们不用把它定义为模板，直接指定它的参数为string类型就可以了。

【参考代码】

```
1  int main() {
2    map<string, int, CmpByKeyLength> name_score_map;
3    name_score_map["LiMin"] = 90;
4    name_score_map["ZiLinMi"] = 79;
5    name_score_map["BoB"] = 92;
6    name_score_map.insert(make_pair("Bing",99));
7    name_score_map.insert(make_pair("Albert",86));
8    for (map<string, int>::iterator iter = name_score_map.begin();
9         iter != name_score_map.end();
10        ++iter) {
11     cout << *iter << endl;
12   }
13   return 0;
14 }
```

【运行结果】

```
BoB     92
Bing    99
LiMin   90
Albert  86
ZiLinMi 79
```

二、C++ STL中Map的按Value排序

在第一部分中，我们借助map提供的参数接口，为它指定相应Compare类，就可以实现对map按Key排序，是在创建map并不断的向其中添加元素的过程中就会完成排序。

现在我们想要从map中得到学生按成绩的从低到高的次序输出，该如何实现呢？换句话说，该如何实现Map的按Value排序呢？

第一反应是利用stl中提供的sort算法实现，这个想法是好的，不幸的是，sort算法有个限制，利用sort算法只能对序列容器进行排序，就是线性的（如vector，list，deque）。map也是一个集合容器，它里面存储的元素是pair，但是它不是线性存储的（前面提过，像红黑树），所以利用sort不能直接和map结合进行排序。

虽然不能直接用sort对map进行排序，那么我们可不可以迂回一下，把map中的元素放到序列容器（如vector）中，然后再对这些元素进行排序呢？这个想法看似是可行的。要对序列容器中的元素进行排序，也有个必要条件：就是容器中的元素必须是可比较的，也就是实现了<操作的。那么我们现在就来看下map中的元素满足这个条件么？

我们知道map中的元素类型为pair，具体定义如下：

```
1  template <class T1, class T2> struct pair
2  {
3    typedef T1 first_type;
4    typedef T2 second_type;
5
6    T1 first;
7    T2 second;
8    pair() : first(T1()), second(T2()) {}
9    pair(const T1& x, const T2& y) : first(x), second(y) {}
10   template <class U, class V>
11     pair (const pair<U,V> &p) : first(p.first), second(p.second) { }
12 }
```

pair也是一个模板类，这样就实现了良好的通用性。它仅有两个数据成员first 和 second，即 key 和 value，而且在<utility>头文件中，还为pair重载了 < 运算符， 具体实现如下：

```
1  template<class _T1, class _T2>
2    inline bool
3    operator<(const pair<_T1, _T2>& __x, const pair<_T1, _T2>& __y)
4    { return __x.first < __y.first
5           || (!(__y.first < __x.first) && __x.second < __y.second); }
```

重点看下其实现：

```
1  __x.first < __y.first || (!(__y.first < __x.first) && __x.second < __y.second)
```

这个less在两种情况下返回true，第一种情况：__x.first < __y.first 这个好理解，就是比较key，如果__x的key 小于 __y的key 则返回true。

第二种情况有点费解： !(__y.first < __x.first) && __x.second < __y.second

当然由于||运算具有短路作用，即当前面的条件不满足是，才进行第二种情况的判断 。第一种情况__x.first < __y.first 不成立，即__x.first >= __y.first 成立，在这个条件下，我们来分析下 !(__y.first < __x.first) && __x.second < __y.second

!(__y.first < __x.first) ，看清出，这里是y的key不小于x的key ，结合前提条件，__x.first < __y.first 不成立，即x的key不小于y的key

即： !(__y.first < __x.first) && !(__x.first < __y.first) 等价于 __x.first == __y.first ,也就是说，第二种情况是在key相等的情况下，比较两者的value（second）。

这里比较令人费解的地方就是，为什么不直接写 __x.first == __y.first 呢？ 这么写看似费解，但其实也不无道理：前面讲过，作为map的key必须实现<操作符的重载，但是并不保证==符也被重载了，如果key没有提供==，那么 ，__x.first == __y.first 这样写就错了。由此可见，stl中的代码是相当严谨的，值得我们好好研读。

现在我们知道了pair类重载了<符，但是它并不是按照value进行比较的，而是先对key进行比较，key相等时候才对value进行比较。显然不能满足我们按value进行排序的要求。

而且，既然pair已经重载了<符，而且我们不能修改其实现，又不能在外部重复实现重载<符。

```
typedef pair<string, int> PAIR;
bool operator< (const PAIR& lhs, const PAIR& rhs) {
  return lhs.second < rhs.second;
}
```

如果pair类本身没有重载<符，那么我们按照上面的代码重载<符，是可以实现对pair的按value比较的。现在这样做不行了，甚至会出错（编译器不同，严格的就报错）。

那么我们如何实现对pair按value进行比较呢？ 第一种：是最原始的方法，写一个比较函数； 第二种：刚才用到了，写一个函数对象。这两种方式实现起来都比较简单。

```
typedef pair<string, int> PAIR;

bool cmp_by_value(const PAIR& lhs, const PAIR& rhs) {
  return lhs.second < rhs.second;
}

struct CmpByValue {
  bool operator()(const PAIR& lhs, const PAIR& rhs) {
    return lhs.second < rhs.second;
  }
};
```

接下来，我们看下sort算法，是不是也像map一样，可以让我们自己指定元素间如何进行比较呢？

```
template <class RandomAccessIterator>
  void sort ( RandomAccessIterator first, RandomAccessIterator last );

template <class RandomAccessIterator, class Compare>
  void sort ( RandomAccessIterator first, RandomAccessIterator last, Compare comp );
```

我们看到，令人兴奋的是，sort算法和map一样，也可以让我们指定元素间如何进行比较，即指定Compare。需要注意的是，map是在定义时指定的，所以传参的时候直接传入函数对象的类名，就像指定key和value时指定的类型名一样；sort算法是在调用时指定的，需要传入一个对象，当然这个也简单，类名()就会调用构造函数生成对象。

这里也可以传入一个函数指针，就是把上面说的第一种方法的函数名传过来。（应该是存在函数指针到函数对象的转换，或者两者调用形式上是一致的，具体确切原因还不明白，希望知道的朋友给讲下，先谢谢了。）

【参考代码】

```
int main() {
  map<string, int> name_score_map;
  name_score_map["LiMin"] = 90;
  name_score_map["ZiLinMi"] = 79;
  name_score_map["BoB"] = 92;
  name_score_map.insert(make_pair("Bing",99));
  name_score_map.insert(make_pair("Albert",86));
 //把map中元素转存到vector中
  vector<PAIR> name_score_vec(name_score_map.begin(), name_score_map.end());
 sort(name_score_vec.begin(), name_score_vec.end(), CmpByValue());
// sort(name_score_vec.begin(), name_score_vec.end(), cmp_by_value);
 for (int i = 0; i != name_score_vec.size(); ++i) {
   cout << name_score_vec[i] << endl;
 }
 return 0;
}
```

【运行结果】