```
1    // Demonstrate the STL "sorted associative containers, map, set
2    // multimap and multiset.
3    // George F. Riley, Georgia Tech, Fall 2011
4
5    #include <iostream>
6    #include <map>
7    #include <set>
8
9    using namespace std;
10
11   // Generic subroutine to print a container
12   template <class ForwardIterator>
13   void Print(ForwardIterator b, ForwardIterator e, bool addEndl = true)
14   {
15     while(b != e)
16       {
17         cout << (*b++);
18         if (addEndl) cout << endl;
19       }
20   }
21
22
23   // Simple "A" object for demonstration
24   class A {
25   public:
26     A(int i) : a(i) {}
27   public:
28     int a;
29   };
30
31   // Define a less than operator for objects of "A"
32   bool    operator<(const A& a1, const A& a2) { return a1.a < a2.a; }
33   // Define an output operator
34   ostream& operator<<(ostream& os, const A& a) { os << a.a; return os; }
35
36   // Define a "map" type, described below
37   typedef map<string, int> StrIntMap_t;
38   // The "value_type" of a map container is a "pair", with
39   // "first" being the key, and "second" is the element
40   typedef StrIntMap_t::value_type StrIntPair_t;
41
42   // Define an output operator for the StrIntPair_t
43   ostream& operator<<(ostream& os, const StrIntPair_t& sip)
44     { cout << "Name " << sip.first << " cost " << sip.second; }
45
46   int main()
47   {
48     // The "set" container simply maintains the object in the
49     // container in sorted order.  This of course implies the
50     // existence of a way to compare two values of set elements
51     // for "less than".
52     typedef set<int> IntSet_t;
53     IntSet_t s;
54     s.insert(1);
55     s.insert(0);
56     s.insert(999);
```

Program map-set.cc

1

```
57    s.insert(888);
58    s.insert(888);
59    s.insert(888);
60    s.insert(2);
61    Print(s.begin(), s.end());
62    // We cannot "push_back" a sorted container (that makes no sense)
63    // nor can we pop_front() or pop_back(), but similar behavior
64    // is easy
65    if (!s.empty())
66      {
67        IntSet_t::iterator last = --s.end();
68        cout << "front " << *s.begin() << " back " << *last << endl;
69        // Remove front and back
70        s.erase(s.begin()); s.erase(last);
71        if (!s.empty())
72          { // Need to check not empty, as an empty container cannot
73            // decrement "end()"
74            last = --s.end();
75            cout << "front " << *(s.begin()) << " back " << *last << endl;
76          }
77      }
78    // A multiset is similar, but allows duplicate values it the set
79    typedef multiset<int> MultiInt_t;
80    MultiInt_t m;
81    m.insert(1);
82    m.insert(0);
83    m.insert(999);
84    m.insert(888);
85    m.insert(888);
86    m.insert(888);
87    m.insert(2);
88    Print(m.begin(), m.end());
89    // Demostrate the standard object "pair".  In this case it is the
90    // return value from "equal_range"
91    pair<MultiInt_t::iterator, MultiInt_t::iterator> p = m.equal_range(888);
92    cout << "Result from equal_range on the  multiset" << endl;
93    // pair objects have two subfields, "first" and "second"
94    Print(p.first, p.second);
95
96    typedef set<A> ASet_t;
97    ASet_t a;
98    a.insert(A(0));
99    a.insert(A(100));
100   a.insert(A(50));
101   a.insert(A(80));
102   a.insert(A(75));
103   cout << "Set of A objects" << endl;
104   Print(a.begin(), a.end());
105
106   // Demonstrate the "map" container.  Similar to set, except that
107   // the sort key is separate from the objects in the container.
108   // Map's have two parts, the "key" and the "element".
109   // For this example, the key is a string and the element is a
110   // cost (integer).
111   typedef map<string, int> StrIntMap_t;
112   StrIntMap_t sim;
```

Program map-set.cc (continued)

```
113    // The "value_type" of a map container is a "pair", with
114    // "first" being the key, and "second" is the element
115    typedef StrIntMap_t::value_type StrIntPair_t;
116    // We can insert object object with "insert"
117    sim.insert(StrIntPair_t("Yugo", 5000));
118    sim.insert(StrIntPair_t("Ford", 10000));
119    cout << "First map print" << endl;
120    Print(sim.begin(), sim.end());
121    // We can also use the indexing operator [] to access a map
122    cout << "Cost of Ford is " << sim["Ford"] << endl;
123    // And we can add an element with the indexing operator
124    sim["Ferrari"] = 200000;
125    // What if the element does not exist?
126    cout << "Cost of Toyota " << sim["Toyota"] << endl;
127    cout << "Final map print" << endl;
128    Print(sim.begin(), sim.end());
129
130    // Multimap is similar, but allow duplicate keys
131    typedef multimap<string, int> StrIntMultiMap_t;
132    StrIntMultiMap_t simm;
133    simm.insert(StrIntPair_t("Yugo", 5000));
134    simm.insert(StrIntPair_t("Ford", 10000));
135    simm.insert(StrIntPair_t("Ferrari", 100000));
136    simm.insert(StrIntPair_t("Ferrari", 300000));
137    simm.insert(StrIntPair_t("Ferrari", 200000));
138    cout << "Final multimap print" << endl;
139    Print(simm.begin(), simm.end());
140
141    // Demonstrate use of "Find" and iterator "first" and "second"
142    StrIntMultiMap_t::iterator mmit = simm.find("Ferrari");
143    cout << "After the \"find()\" call on the StrIntMultiMap" << endl;
144    if (mmit == simm.end()) cout << "HuH?  No Ferraris?" << endl;
145    else                    cout << mmit->first << " " << mmit->second << endl;
146 }
147
148
149
```

Program map-set.cc (continued)