```
// Demonstrate the STL "sorted associative containers, map, set
// multimap and multiset.
// George F. Riley, Georgia Tech, Summer 2006

#include <iostream>
#include <map>
#include <set>

using namespace std;

// Generic subroutine to print a container
template <class ForwardIterator>
void Print(ForwardIterator b, ForwardIterator e, bool addEndl = true)
{
  while(b != e)
    {
      cout << (*b++);
      if (addEndl) cout << endl;
    }
}


// Simple "A" object for demonstration
class A {
public:
  A(int i) : a(i) {}
public:
  int a;
};

// Define a less than operator for objects of "A"
bool     operator<(const A& a1, const A& a2) { return a1.a < a2.a; }
// Define an output operator
ostream& operator<<(ostream& os, const A& a) { os << a.a; return os; }

// Define a "map" type, described below
typedef map<string, int> StrIntMap_t;
// The "value_type" of a map container is a "pair", with
// "first" being the key, and "second" is the element
typedef StrIntMap_t::value_type StrIntPair_t;

// Define an output operator for the StrIntPair_t
ostream& operator<<(ostream& os, const StrIntPair_t& sip)
  { cout << "Name " << sip.first << " cost " << sip.second; }

int main()
{
  // The "set" container simply maintains the object in the
  // container in sorted order.  This of course implies the
  // existence of a way to compare two values of set elements
  // for "less than".
  typedef set<int> IntSet_t;
  IntSet_t s;
  s.insert(1);
  s.insert(0);
  s.insert(999);
```

Program map-set.cc

```
s.insert(888);
s.insert(888);
s.insert(888);
s.insert(2);
Print(s.begin(), s.end());
// We cannot "push_back" a sorted container (that makes no sense)
// nor can we pop_front() or pop_back(), but similar behavior
// is easy
if (!s.empty())
  {
    IntSet_t::iterator last = --s.end();
    cout << "front " << *s.begin() << " back " << *last << endl;
    // Remove front and back
    s.erase(s.begin()); s.erase(last);
    if (!s.empty())
      { // Need to check not empty, as an empty container cannot
        // decrement "end()"
        last = --s.end();
        cout << "front " << *(s.begin()) << " back " << *last << endl;
      }
  }
// A multiset is similar, but allows duplicate values it the set
typedef multiset<int> MultiInt_t;
MultiInt_t m;
m.insert(1);
m.insert(0);
m.insert(999);
m.insert(888);
m.insert(888);
m.insert(888);
m.insert(2);
Print(m.begin(), m.end());
// Demostrate the standard object "pair".  In this case it is the
// return value from "equal_range"
pair<MultiInt_t::iterator, MultiInt_t::iterator> p = m.equal_range(888);
cout << "Result from equal_range on the  multiset" << endl;
// pair objects have two subfields, "first" and "second"
Print(p.first, p.second);

typedef set<A> ASet_t;
ASet_t a;
a.insert(A(0));
a.insert(A(100));
a.insert(A(50));
a.insert(A(80));
a.insert(A(75));
cout << "Set of A objects" << endl;
Print(a.begin(), a.end());

// Demonstrate the "map" container.  Similar to set, except that
// the sort key is separate from the objects in the container.
// Map’s have two parts, the "key" and the "element".
// For this example, the key is a string and the element is a
// cost (integer).
typedef map<string, int> StrIntMap_t;
StrIntMap_t sim;
```

Program map-set.cc (continued)

```
  // The "value_type" of a map container is a "pair", with
  // "first" being the key, and "second" is the element
  typedef StrIntMap_t::value_type StrIntPair_t;
  // We can insert object object with "insert"
  sim.insert(StrIntPair_t("Yugo", 5000));
  sim.insert(StrIntPair_t("Ford", 10000));
  cout << "First map print" << endl;
  Print(sim.begin(), sim.end());
  // We can also use the indexing operator [] to access a map
  cout << "Cost of Ford is " << sim["Ford"] << endl;
  // And we can add an element with the indexing operator
  sim["Ferrari"] = 200000;
  // What if the element does not exist?
  cout << "Cost of Toyota " << sim["Toyota"] << endl;
  cout << "Final map print" << endl;
  Print(sim.begin(), sim.end());

  // Multimap is similar, but allow duplicate keys
  typedef multimap<string, int> StrIntMultiMap_t;
  StrIntMultiMap_t simm;
  simm.insert(StrIntPair_t("Yugo", 5000));
  simm.insert(StrIntPair_t("Ford", 10000));
  simm.insert(StrIntPair_t("Ferrari", 100000));
  simm.insert(StrIntPair_t("Ferrari", 300000));
  simm.insert(StrIntPair_t("Ferrari", 200000));
  cout << "Final multimap print" << endl;
  Print(simm.begin(), simm.end());

  // Demonstrate use of "Find" and iterator "first" and "second"
  StrIntMultiMap_t::iterator mmit = simm.find("Ferrari");
  cout << "After the \"find()\" call on the StrIntMultiMap" << endl;
  if (mmit == simm.end()) cout << "HuH?  No Ferraris?" << endl;
  else                    cout << mmit->first << " " << mmit->second << endl;
}
```

Program map-set.cc (continued)