```
#include <map>

int main ()
{
  typedef multimap<char, int, less<char> > mmap;
  mmap m;
  cout << "count ('X') = " << m.count ('X') << endl;
  m.insert (pair<const char, int> ('X', 10));
  cout << "count ('X') = " << m.count ('X') << endl;
  m.insert (pair<const char, int> ('X', 20));
  cout << "count ('X') = " << m.count ('X') << endl;
  m.insert (pair<const char, int> ('Y', 32));
  mmap::iterator i = m.find ('X'); // Find first match.
  while (i != m.end ()) // Loop until end is reached.
  {
    cout << (*i).first << " -> " << (*i).second << endl;
    i++;
  }
  int count = m.erase ('X');
  cout << "Erased " << count << " items" << endl;
  return 0;
}

/*
count ('X') = 0
count ('X') = 1
count ('X') = 2
X -> 10
X -> 20
Y -> 32
Erased 2 items
*/
```

```
#include <map>

int main ()
{
  typedef map<char, int, less<char> > maptype;
  maptype m;
  // Store mappings between roman numerals and decimals.
  m['l'] = 50;
  m['x'] = 20; // Deliberate mistake.
  m['v'] = 5;
  m['i'] = 1;
  cout << "m['x'] = " << m['x'] << endl;
  m['x'] = 10; // Correct mistake.
  cout << "m['x'] = " << m['x'] << endl;
  cout << "m['z'] = " << m['z'] << endl; // Note default value is added.
  cout << "m.count ('z') = " << m.count ('z') << endl;
  pair<maptype::iterator, bool> p;
  p = m.insert (pair<const char, int> ('c', 100));
  if (p.second)
    cout << "First insertion successful" << endl;
  p = m.insert (pair<const char, int> ('c', 100));
  if (p.second)
    cout << "Second insertion successful" << endl;
  else
    cout << "Existing pair " << (*(p.first)).first
         << " -> " << (*(p.first)).second << endl;
  return 0;
}

/*
m['x'] = 20
m['x'] = 10
m['z'] = 0
m.count ('z') = 1
First insertion successful
Existing pair c -> 100
*/
```

```
#include <map>

typedef multimap<int, char, less<int> > mmap;
typedef pair<const int, char> pair_type;

pair_type p1 (3, 'c');
pair_type p2 (6, 'f');
pair_type p3 (1, 'a');
pair_type p4 (2, 'b');
pair_type p5 (3, 'x');
pair_type p6 (6, 'f');

pair_type array [] = {p1,p2,p3,p4,p5,p6};
```

```
int main ()
{
  mmap m (array, array + 7);
  mmap::iterator i;
  // Return location of first element that is not less than 3
  i = m.lower_bound (3);
  cout << "lower bound:" << endl;
  cout << (*i).first << " -> " << (*i).second << endl;
  // Return location of first element that is greater than 3
  i = m.upper_bound (3);
  cout << "upper bound:" << endl;
  cout << (*i).first << " -> " << (*i).second << endl;
  return 0;
}

/*
lower bound:
3 -> c
upper bound:
6 -> f
*/
```

```
#include <set>
int main ()
{
  set<int, less<int> > s;
  pair<set<int, less<int> >::const_iterator, bool> p;
  p = s.insert (42);
  if (p.second)
   cout << "1Inserted new element " << *(p.first) << endl;
  else
   cout << "1Existing element = " << *(p.first) << endl;
  p = s.insert (42);
  if (p.second)
   cout << "2Inserted new element " << *(p.first) << endl;
  else
   cout << "2Existing element = " << *(p.first) << endl;
  return 0;
}

/*
1Inserted new element 42
2Existing element = 42
*/
```

```
#include <set>
int main ()
{
  typedef multiset<int, less<int> > mset;
  mset s;
  cout << "count (42) = " << s.count (42) << endl;
  s.insert (42);
  cout << "count (42) = " << s.count (42) << endl;
  s.insert (42);
  cout << "count (42) = " << s.count (42) << endl;

  mset::iterator i = s.find (40);

  if (i == s.end ())
    cout << "40 Not found" << endl;
  else
    cout << "Found " << *i << endl;
  i = s.find (42);
  if (i == s.end ())
    cout << "Not found" << endl;
  else
    cout << "Found " << *i << endl;
  int count = s.erase (42);
  cout << "Erased " << count << " instances" << endl;
  return 0;
}

/*
count (42) = 0
count (42) = 1
count (42) = 2
40 Not found
Found 42
Erased 2 instances
*/
```

```
#include <set>

int array [] = { 3, 6, 1, 2, 3, 2, 6, 7, 9 };

int main ()
{
  multiset<int, less<int> > s (array, array + 9);
  multiset<int, less<int> >::iterator i;
  // Return location of first element that is not less than 3
  i = s.lower_bound (3);
  cout << "lower bound = " << *i << endl;
  // Return location of first element that is greater than 3
  i = s.upper_bound (3);
  cout << "upper bound = " << *i << endl;
  return 0;
}

/*
lower bound = 3
upper bound = 6
*/
```

```
#include <set>

int array [] = { 3, 6, 1, 2, 3, 2, 6, 7, 9 };

int main ()
{
  typedef multiset<int, less<int> > mset;
  mset s (array, array + 9);
  pair<mset::const_iterator, mset::const_iterator> p = s.equal_range (3);
  cout << "lower bound = " << *(p.first) << endl;
  cout << "upper bound = " << *(p.second) << endl;
  return 0;
}

/*
lower bound = 3
upper bound = 6
*/
```

```
#include <set>

int array [] = { 3, 6, 1, 9 };

int main ()
{
  typedef multiset<int> mset1;
  typedef multiset<int, greater<int> > mset2;

  mset1 s1 (array, array + 4);
  mset1::const_iterator i1 = s1.begin ();
  cout << "Using less_than: " << endl;
  while (i1 != s1.end ())
    cout << *i1++ << endl;
  mset2 s2 (array, array + 4);
  mset2::const_iterator i2 = s2.begin ();
  i2 = s2.begin ();
  cout << "Using greater_than: " << endl;
  while (i2 != s2.end ())
    cout << *i2++ << endl;
  return 0;
}

/*
Using less_than:
1
3
6
9
Using greater_than:
9
6
3
1
*/
```