The Standard Template Library

Kafi Rahman

Assistant Professor @CS
Truman State University

Storing Objects Of Your Own Classes as Keys in a map

- If you want to store an object as a key in a map, there is one requirement for that object's class:
  - It must overload the < operator.
- Consider the following Customer class…

Storing Objects Of Your Own Classes as Keys in a map

```
#ifndef CUSTOMER_H
#define CUSTOMER_H
#include<string>
using namespace std;

class Customer
{
private:
     int custNumber;
     string name;
public:
     Customer(int cn, string n)
     {   custNumber = cn;
         name = n; }

     void setCustNumber(int cn)
     {   custNumber = cn; }

     void setName(string n)
     {   name = n; }

     int getCustNumber() const
     {   return custNumber; }

     string getName() const
     {   return name; }

     bool operator < (const Customer &right) const
     {   bool status = false;

         if (custNumber < right.custNumber)
            status = true;

         return status; }
};
#endif
```

Program 17-17

```
#include <iostream>
#include <string>
#include <map>
#include "Customer.h"
using namespace std;

int main()
{
   // Create some Customer objects.
   Customer customer1(1001, "Sarah Scott");
   Customer customer2(1002, "Austin Hill");
   Customer customer3(1003, "Megan Cruz");

   // Create a map to hold the seat assignments.
   map<Customer, string> assignments;

   // Use the map to store the seat assignments.
   assignments[customer1] = "1A";
   assignments[customer2] = "2B";
   assignments[customer3] = "3C";

   // Display all objects in the map.
   for (auto element : assignments)
   {
      cout << element.first.getName() << "\t"
           << element.second << endl;
   }

   return 0;
}
```

Program Output

```
Sarah Scott     1A
Austin Hill     2B
Megan Cruz      3C
```

This program assigns seats in a theater to customers. The map uses Customer objects as keys, and seat numbers as values.

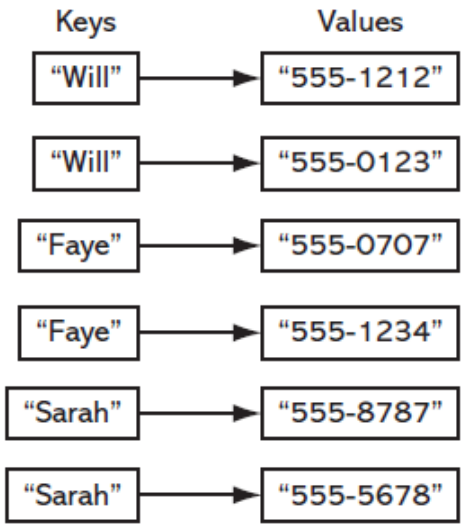
The unordered_map Class

- The unordered_map class is similar to the map class, except in two regards:
  - The keys in an unordered_map are not sorted
  - The unordered_map class has better performance
- You should use the unordered_map class instead of the map class if:
  - You will be making a lot of searches on a large number of elements
  - You are not concerned with retrieving them in key order
- The unordered_map class is declared in the <unordered_map> header file

The multimap Class

- The mulitmap class is a map that allows duplicate keys
- The mulitmap class has most of the same member functions as the map class (see Table 17-11 in your textbook)
- The multimap class is declared in the <map> header file

The multimap Class



- Consider a phonebook application where the key is a person's name and the value is that person's phone number.
- A multimap container would allow each person to have multiple phone numbers

The multimap Class

Program 17-19

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

int main()
{
   // Define a phonebook multimap.
   multimap<string, string> phonebook =
      { {"Will", "555-1212"},  {"Will", "555-0123"},
        {"Faye", "555-0707"},  {"Faye", "555-1234"},
        {"Sarah", "555-8787"}, {"Sarah", "555-5678"} };

   // Display the elements in the multimap.
   for (auto element : phonebook)
   {
      cout << element.first << "\t"
           << element.second << endl;
   }
   return 0;
}
```

Program Output

```
Faye     555-0707
Faye     555-1234
Sarah    555-8787
Sarah    555-5678
Will     555-1212
Will     555-0123
```

Adding Elements to a multimap

- The multimap class does not overload the `[]` operator.
  - So, you cannot use an assignment statement to add a new element to a multimap.
- Instead, you will use either the `emplace()` or the `insert()` member functions.

Adding Elements to a multimap

```
multimap<string, string> phonebook;
phonebook.emplace("Will", "555-1212");
phonebook.emplace("Will", "555-0123");
phonebook.emplace("Faye", "555-0707");
phonebook.emplace("Faye", "555-1234");
phonebook.emplace("Sarah", "555-8787");
phonebook.emplace("Sarah", "555-5678");
```

Adding Elements to a multimap

```
multimap<string, string> phonebook;
phonebook.insert(make_pair("Will", "555-1212"));
phonebook.insert(make_pair("Will", "555-0123"));
phonebook.insert(make_pair("Faye", "555-0707"));
phonebook.insert(make_pair("Faye", "555-1234"));
phonebook.insert(make_pair("Sarah", "555-8787"));
phonebook.insert(make_pair("Sarah", "555-5678"));
```

Getting the Number of Elements With a Specified Key

Program 17-20

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

int main()
{
   // Define a phonebook multimap.
   multimap<string, string> phonebook =
      { {"Will", "555-1212"},  {"Will", "555-0123"},
        {"Faye", "555-0707"},  {"Faye", "555-1234"},
        {"Sarah", "555-8787"}, {"Sarah", "555-5678"} };

   // Display the number of elements that match "Faye".
   cout << "Faye has " << phonebook.count("Faye") << " elements.\n";
   return 0;
}
```

Program Output

```
Faye has 2 elements.
```

- The multimap class's count() member function accepts a key as its argument, and returns the number of elements that match the specified key.

Displaying the Elements With a Specified Key

```
int main()
{
     multimap <int, int> map;

     // insert the values in multimap
     map.insert(make_pair(1, 10));
     map.insert(make_pair(2, 20));
     map.insert(make_pair(2, 30));
     map.insert(make_pair(2, 40));
     map.insert(make_pair(3, 50));
     map.insert(make_pair(4, 60));
     map.insert(make_pair(4, 70));

     int key = 2;
     for (auto itr = map.begin(); itr != map.end(); itr++)
          if (itr -> first == key)
               cout << itr -> first << "  "
                    << itr -> second << endl;

     return 0;
}
```

Displaying the Elements With a Specified Key

```
int main()
{
      multimap <int, int> map;

      // insert the values in multimap
      map.insert(make_pair(1, 10));
      map.insert(make_pair(2, 20));
      map.insert(make_pair(2, 30));
      map.insert(make_pair(2, 240));
      map.insert(make_pair(3, 50));
      map.insert(make_pair(4, 60));
      map.insert(make_pair(4, 70));

      int key = 2;
      auto start_itr = map.lower_bound(key);
      auto end_itr = map.upper_bound(key);

      while (start_itr != end_itr)
      {
            if (start_itr -> first == key)
               cout << start_itr -> first << "  "
                        << start_itr -> second << endl;
            start_itr++;
      }

      return 0;
}
```

Deleting Elements with a Specified Key

```
// Define a phonebook multimap.
multimap<string, string> phonebook =
    { {"Will", "555-1212"},  {"Will", "555-0123"},
      {"Faye", "555-0707"},  {"Faye", "555-1234"},
      {"Sarah", "555-8787"}, {"Sarah", "555-5678"} };

// Delete Will's phone numbers from the multimap.
phonebook.erase("Will");
```

- To delete all elements matching a specified key, use the erase() member function.

17.5

The set, multiset, and unordered_set Classes

Sets

- A set is an associative container that is similar to a mathematical set.
- You can use the STL set class to create a set container.
- All the elements in a set must be unique. No two elements can have the same value.
- The elements in a set are automatically sorted in ascending order.
- The set class is declared in the <set> header file.

set Class Constructors

Default Constructor	set<dataType> name; Creates an empty set.
Range Constructor	set<dataType> name(iterator1, iterator2); Creates a set that is initialized with a range of values. iterator1 marks the beginning of the range and iterator2 marks the end.
Copy Constructor	set<dataType> name(set2); Creates a set that is a copy of set2.

The set Class

- Example: defining a set container to hold integers:

```
set<int> numbers;
```

- Example: defining and initializing a set container to hold integers:

```
set<int> numbers = {1, 2, 3, 4, 5};
```

The set Class

- A set cannot contain duplicate items.
- If the same value appears more than once in an initialization list, it will be added to the set only one time.
- For example, the following set will contain the values 1, 2, 3, 4, and 5:

```
set<int> numbers = {1, 1, 2, 2, 3,
4, 5, 5, 5};
```

Adding New Elements to a set

- The insert() member function adds a new element to a set:

```
set<int> numbers;
numbers.insert(10);
numbers.insert(20);
numbers.insert(30);
```

Stepping Through a set With the for Loop

```
// Create a set containing names.
set<string> names = {"Joe", "Karen", "Lisa", "Jackie"};

// Display each element.
for (string element : names)
{
   cout << element << endl;
}
```

Using an Iterator With a set

- The begin() and end() member functions return a bidirectional iterator of the iterator type
- The cbegin() and cend() member functions return a bidirectional iterator of the const_iterator type
- The rbegin() and rend() member functions return a reverse bidirectional iterator of the reverse_iterator type
- The crbegin() and crend() member functions return a reverse bidirectional iterator of the const_reverse_iterator type

Using an Iterator With a set

```
// Create a set containing names.
set<string> names = {"Joe", "Karen", "Lisa", "Jackie"};

// Create an iterator.
set<string>::iterator iter;

// Use the iterator to display each element in the set.
for (iter = names.begin(); iter != names.end(); iter++)
{
   cout << *iter << endl;
}
```

Determining Whether an Element Exists

```
set<string> names = {"Joe", "Karen", "Lisa", "Jackie"};
if (names.count("Lisa"))
   cout << "Lisa was found in the set.\n";
else
   cout << "Lisa was not found.\n";
```

- The set class's count() member function accepts a value as its argument, and returns 1 if that value exists in the set. The function returns 0 otherwise.

Retrieving an Element

- The set class has a find() member function that searches for an element with a specified value.
- The find() function returns an iterator to the element matching it.
- If the element is not found, the find() function returns an iterator to the end of the set.

Retrieving an Element

```
// Create a set containing names.
set<string> names = {"Joe", "Karen", "Lisa", "Jackie"};

// Create an iterator.
set<string>::iterator iter;

// Find "Karen".
iter = names.find("Karen");

// Display the result.
if (iter != names.end())
{
   cout << *iter << " was found.\n";
}
else
{
   cout << "Karen was not found.\n";
}
```

Storing Objects Of Your Own Classes in a set

- If you want to store an object in a set, there is one requirement for that object's class:

  It must overload the < operator.

- Consider the following Customer class…

Storing Objects Of Your Own Classes in a set

```
#ifndef CUSTOMER_H
#define CUSTOMER_H
#include<string>
using namespace std;

class Customer
{
private:
     int custNumber;
     string name;
public:
     Customer(int cn, string n)
     {   custNumber = cn;
         name = n; }

     void setCustNumber(int cn)
     {   custNumber = cn; }

     void setName(string n)
     {   name = n; }

     int getCustNumber() const
     {   return custNumber; }

     string getName() const
     {   return name; }

     bool operator < (const Customer &right) const
     {   bool status = false;

         if (custNumber < right.custNumber)
            status = true;

         return status; }
};
#endif
```

Storing Objects Of Your Own Classes in a set

Program 17-22

```
#include <iostream>
#include <set>
#include "Customer.h"
using namespace std;

int main()
{
   // Create a set of Customer objects.
   set<Customer> customerset =
     { Customer(1003, "Megan Cruz"),
       Customer(1002, "Austin Hill"),
       Customer(1001, "Sarah Scott")
     };

   // Try to insert a duplicate customer number.
   customerset.emplace(1001, "Evan Smith");

   // Display the set elements
   cout << "List of customers:\n";
   for (auto element : customerset)
   {
      cout << element.getCustNumber() << " "
           << element.getName() << endl;
   }

```

Continued…

Storing Objects Of Your Own Classes in a set

```
    // Search for customer number 1002.
    cout << "\nSearching for Customer Number 1002:\n";
    auto it = customerset.find(Customer(1002, ""));

    if (it != customerset.end())
        cout << "Found: " << it->getName() << endl;
    else
        cout << "Not found.\n";

    return 0;
}
```

Program Output

```
List of customers:
1001 Sarah Scott
1002 Austin Hill
1003 Megan Cruz

Searching for Customer Number 1002:
Found: Austin Hill
```

The multiset Class

- The mulitset class is a set that allows duplicate items.
- The mulitset class has the same member functions as the set class (see Table 17-13 in your textbook).
- The multiset class is declared in the <set> header file.

The multiset Class

- In the set class, the count() member function returns either 0 or 1. In the multiset class, the count() member function can return values greater than 1.

- In the set class, the equal_range() member function returns a range with, at most, one element. In the multiset class, the equal_range() member function can return a range with multiple elements.

The unordered_set Class

- The unordered_set class is similar to the set class, except in two regards:
  - The values in an unordered_set are not sorted
  - The unordered_set class has better performance
- You should use the unordered_set class instead of the set class if:
  - You will be making a lot of searches on a large number of elements
  - You are not concerned with retrieving them in ascending order
- The unordered_set class is declared in the <unordered_set> header file

The unordered_multiset Class

- The unordered_multiset class is similar to the multiset class, except in two regards:
  - The values in an unordered_multiset are not sorted
  - The unordered_multiset class has better performance
- You should use the unordered_multiset class instead of the multiset class if:
  - You will be making a lot of searches on a large number of elements
  - You are not concerned with retrieving them in ascending order
- The unordered_multiset class is declared in the <unordered_set> header file

Thank you