CSCI 3110

STL: Associative containers (Elements in associative containers are referenced by their *key* and not by their absolute position in the container.)

Set is an ordered collection of unique keys

- Ex. A set of student M numbers

```
#include <set>
#include <iostream>
using namespace std;

void PrintSet(set<int> &theSet);

int main()   {
   set < int > s;

   // inserting elements in random order .
   s.insert( 60 ) ;
   s.insert( 10 ) ;
   s.insert( 20 ) ;
   s.insert( 20 ) ;
   s.insert( 40 ) ;
   s.insert( 50 ) ;

   PrintSet(s);

   cout<< "The size of set : " << s.size() <<endl ;

   set<int >::iterator p;
   p=s.find(40);
   s.erase(p);

   PrintSet(s);

   return 0 ;
}

// printing set s
//initializing the iterator, iterating to the beginning of the set.
void PrintSet(set<int> &theSet) {
   set<int >::iterator it ;
   cout << "The element of set s are : \n";
   for (it = theSet.begin() ; it != theSet.end() ; it++ )    {
      cout << *it<< " ";
   }
   cout << endl;
}
```

When to use Set?

- When there is a need of storing the elements in a sorted manner
- When there is a need to search for elements by value efficiently

STL MultiSet: similar to set except keys are not unique

STL map

Associate a value with a unique key value : (key, value) pairs

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

int main()  {
   // Create a map of strings to integers
   map<string, int> mp;

   // Insert some values into the map
   mp["one"] = 1;
   mp["two"] = 2;
   mp["three"] = 3;

   // Get an iterator pointing to the first element in the
   // map
   map<string, int>::iterator it = mp.begin();

   // Iterate through the map and print the elements
   while (it != mp.end())
   {
      cout << "Key: " << it->first
          << ", Value: " << it->second << endl;
      ++it;
   }

   return 0;
}
```

Output:

```
Key: one, Value: 1
Key: three, Value: 3
Key: two, Value: 2
```

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

int main() {
 // empty map container
    map<int, int> gquiz1;

    // insert elements in random order
    gquiz1.insert(pair<int, int>(1, 40));
    gquiz1.insert(pair<int, int>(2, 30));
    gquiz1.insert(pair<int, int>(3, 60));
    gquiz1.insert(pair<int, int>(4, 20));
    gquiz1.insert(pair<int, int>(5, 50));
    gquiz1.insert(pair<int, int>(6, 50));

    // another way of inserting a value in a map
    gquiz1[7] = 10;

    // printing map gquiz1
    map<int, int>::iterator itr;
    cout << "\nThe map gquiz1 is : \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz1.begin(); itr != gquiz1.end(); ++itr) {
        cout << '\t' << itr->first << '\t' << itr->second
             << '\n';
    }
    cout << endl;

    // assigning the elements from gquiz1 to gquiz2
    map<int, int> gquiz2(gquiz1.begin(), gquiz1.end());

    // print all elements of the map gquiz2
    cout << "\nThe map gquiz2 after"
         << " assign from gquiz1 is : \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {
        cout << '\t' << itr->first << '\t' << itr->second
             << '\n';
    }
    cout << endl;
    // remove all elements up to
    // element with key=3 in gquiz2
    cout << "\ngquiz2 after removal of"
            " elements less than key=3 : \n";
    cout << "\tKEY\tELEMENT\n";
    gquiz2.erase(gquiz2.begin(), gquiz2.find(3));
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {
        cout << '\t' << itr->first << '\t' << itr->second
             << '\n';
    }

    // remove all elements with key = 4
    int num;
    num = gquiz2.erase(4);
    cout << "\ngquiz2.erase(4) : ";
    cout << num << " removed \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {
        cout << '\t' << itr->first << '\t' << itr->second
             << '\n';
    }

    cout << endl;

    return 0;
}
```

```
Output:
Key: one, Value: 1
Key: three, Value: 3
Key: two, Value: 2

The map gquiz1 is :
        KEY     ELEMENT
        1       40
        2       30
        3       60
        4       20
        5       50
        6       50
        7       10


The map gquiz2 after assign from gquiz1 is :
        KEY     ELEMENT
        1       40
        2       30
        3       60
        4       20
        5       50
        6       50
        7       10


gquiz2 after removal of elements less than key=3 :
        KEY     ELEMENT
        3       60
        4       20
        5       50
        6       50
        7       10

gquiz2.erase(4) : 1 removed
        KEY     ELEMENT
        3       60
        5       50
        6       50
        7       10
```

When to use map?

- When we want to have fast access to a value via its key, useful when building any kind of index or references
- When we need to keep the keys unique across the entire data structure, i.e., no duplicates.

Multimap: similar to map except it permits multiple entries to have the same key