Lecture 11

Standard Template Library

Lists

Iterators

Sets

Maps

Lists

- #include <list>
- Doubly-linked list
- Allowing insertion, deletion and modification of elements at the front, the back and in the middle
- **size, push_back, push_front, pop_back and pop_front.**
- Mechanism of ***iterators*** is needed

Iterators

- Vectors, lists, queues, etc. are all examples of collections. In fact, they are all subclasses of a **Container** class in STL
- An iterator is like a reference (or pointer) to one of the elements of a collection.
- Supports at least two operations:

  1) ++ operator: advance forward one position

  2) unary * operator : return the element being referenced

Lists & Iterators: Example

```
list< int > c;                    //Creates an empty list of integers

c.push_back( 3 );
c.push_back( 5 );
c.push_front( 6 );                //Add 3 elements to the list, 6, 3, 5

for( list< int >::iterator i = c.begin(); i != c.end(); ++i )
{
        cout << " " << *i;
}
```

1. list< int >::iterator: class name for an iterator over a list of integers.
2. i: iterator, initialized to c.begin()

Lists & Iterators: Example

```
list< int > c;                    //Creates an empty list of integers

c.push_back( 3 );
c.push_back( 5 );
c.push_front( 6 );                //Add 3 elements to the list, 6, 5, 3

for( list< int >::iterator i = c.begin(); i != c.end(); ++i )
{
        cout << " " << *i;
}
```

3. The **begin()** method of every collection returns an iterator pointing to the first element of the collection.
4. The **end()** method returns an iterator pointing to the position in the collection that is one **after** the last element. In other words, when i is equal to c.end(), i has run off the end of the list and is pointing to a nonexistent position – the loop should then stop.
5. *i – the value that i is referencing.

Result: _6_3_5

More Example

1. Same as using iterators with Java collections, but with a very different syntax.
2. An important difference is the pair of methods, **begin() and end().** Think of them as defining an interval [begin, end) that spans the entire collection. The **left** endpoint is **included** in the range, and the **right** endpoint is **excluded**.
3. If begin() is equal to end()?
4. You can also pass a range to the constructor of most collections as follows.

http://www.cplusplus.com/reference/vector/vector/vector/

```
vector< double > v( 10, 7.0 );    //creates a vector of 10 elements, each is 7

list< double > w( v.begin(), v.end() );  //creates a list containing copy

vector< double > x( ++w.begin(), w.end() ); //creates another vector, 9 elements
```

Sets

- #include <set>
- It is a sorted collection of elements, stored as a balanced **binary search tree.**
- No push_back or push_front **, insert()** instead**.**(sorted)
- To store elements of type C in a **set, the < operator must be defined** for type C. (Opertator Overloading)

(You can **safely** declare sets of int, double, char or any othe primitive types because you can use the < operator to compare two ints, two doubles or two chars – the operator is already defined in C++.) http://www.cplusplus.com/reference/set/set/

Sets: Example

```
set< string > ss;     //creates an empty set of strings, #include <string>

ss.insert( "Sheridan" );
ss.insert( "Delenn" );
ss.insert( "Lennier" );    //add elements to the set

for( set< string >::iterator j = ss.begin(); j != ss.end(); ++j )
{
        cout << " " << *j;
}
```

1. set< string >::iterator : class name for an iterator over a set of strings
2. j: iterator, initialized to ss.begin()
3. ss.begin(): smallest element; ss.end(): one after the last element of the set
4. ++j increments the iterator. *j returns the string being referenced by j.

Result: _Delenn_Lennier_Sheridan

More Examples

```
set< int > s( v.begin(), v.end() );
 //Create a set s by passing it a range containing all of the elements  of v.
Time:  O(n*log(n))

vector< int > w( s.begin(), s.end() );
//Create another vector, w, containing all of the elements of s. Time:  Linear
```

```
set< int > s;
for( int i = 1900; i <= 3000; i += 4 )
{
      s.insert( i );
}
for( int i = 1900; i <= 3000; i += 100 )
{
      if( i % 400 != 0 )
     {
          if( s.count( i ) == 1 ) s.erase( i );
      }
}
```

1. Erase(): remove an element
2. Count(): check whether an element is in the set

Maps

- #include <map>
- store key-value pairs rather than collection of elements.
- maps are sorted by the keys.
- Implemented using balanced binary search trees. insertion, deletion and update operations require **logarithmic** time in the size of the map.
- The map class has 2 template parameters – key type and value type. (using [] to access entries)

Maps: Example

```
vector< string > i2s( 7 );
i2s[0] = "Sunday"; i2s[1] = "Monday"; i2s[2] = "Tuesday"; i2s[3] =
"Wednesday“; i2s[4] = "Thursday"; i2s[5] = "Friday"; i2s[6] =
"Saturday";

map< string, int > s2i;

for( int i = 0; i < 7; i++ )
{
        s2i[i2s[i]] = i;
}
```

Maps: More Example

```
string word;
map< string, int > freq;

while( cin >> word )
{
        freq[word]++;
}

for( map< string, int >::iterator i = freq.begin(); i != freq.end(); ++i )
{
        cout << (*i).first << ": " << (*i).second << endl;
}  //pair classes?
```

STL – Pair in Map

- the STL defines a class pair that simply stores 2 things.
- pair has 2 member variables (or fields): first is of type A and second is of type B. To create a pair, you need to manually specify what A and B are, like this.
- pair< string, double > p("pi", 3.14); //make_pair("pi", 3.14); p.second = 3.14159;
- The first line calls the constructor.
- To use pair, you need to #include <map>. (map: like Java's TreeMap - basically, a set of key-value pairs, sorted by the key.)

```
int min( int a, int b ) {
return a < b ? a : b;
}
(templates, chars, doubles,
strings, and anything else that has the
'<' operator defined on it. )

template< class C >
C min( C a, C b ) {
return a < b ? a : b;
}
```

```
template< class A, class B >
class pair {
public:
        A first;
        B second;
        pair( A a, B b ) {
            first = a;
            second = b; }
};
```

Maps: More Example

```
map< string, int > freq;
vector< string > commonwords;

commonwords.push_back( "and" );

for ( int i=0; i < commonwords.size(); i++ )
{
      if ( freq.count( commonwords[i] ) ) // Check if this common word is
in the map
     freq.erase( commonwords[i] ); // Erase it so our map don't count it
}
```

- count(): check whether a given key exists in a map.
- erase(k): erase the key-value pair whose key is equal to k.
- size(): return the size of the map.