STL For Beginners

Dr. Ganesh R. Baliga
Computer Science Department

Introduction

- STL: Standard Template Library
- Incorporated into the ANSI/ISO standard for the C++ language
- Collection of classes and functions
  - Frequently used
  - Generic (template-based)
  - Robust
  - Efficient

Collections

- STL container classes: deque, vector, list, queue, set, stack, map, multiset, multimap, priority_queue.
- Examples
  - Customers lined at a store's checkout: queue (of customers)
  - Words present in a dictionary: set (of strings)
  - Most recently visited web sites in a browser: stack (of URLs)

Collections

- Phone book: map (person mapped to phone number)
- Index at the end of a text book: multimap (word mapped to one or more page numbers)

Collections

- A resizable array: vector
  - Allows fast access to an item, given its position
  - Size can be increased
- A position based *sequence*: list
  - Duplicates allowed
  - Allows insertion and removal at arbitrary positions
- A double ended queue: deque
  - Queue that permits insertion and deletion of elements at both ends.

Background: The C++ Class

```
class Rational
{
  public:

  // Constructors
  Rational ( int numerator, int denominator = 1 );
  Rational ( );  // default constructor

  // Accessors; to access the private attributes

  int getNumerator ( ) const;
  int getDenominator ( ) const;
```

```
// Mutators; to change the private attributes

  void setNumerator ( int value );
  void setDenominator ( int value );

  private:
  int n_, d_;  // privately held attributes
};

// returns the sum of two Rationals
Rational operator + ( const Rational & f, const Rational & s );

// > operator for Rationals
bool operator > (const Rational & f, const Rational & s );

// stream insertion operator for Rationals
ostream & operator << (ostream & out, const Rational & r);
```

- Overloading the operator > for the class Rational

```
// Comparing two Rational numbers
bool operator > (const Rational & f,     // first rational
                 const Rational & s )  // second rational
{
    bool result;

    double frac1 = f.getNumerator( ) / (double) f.getDenominator( );
    double frac2 = g.getNumerator( ) / (double) g.getDenominator( );

    if ( frac1 > frac2)
        result = true;
    else
        result = false;

    return result;
} // end operator >
```

- This class definition allows you to:
  - Rational r1, r2 (4, 5);
  - if (r1 > r2) …// calls operator > (r1, r2)
  - cout << r1; // calls operator << (cout, r1)
- It does not allow you to:
  - if (r1 < r2) …
  - cin >> r1;
- C++ permits you to overload all operators such as the arithmetic operators (+, -, /, *), the comparison operators (<, >, !=, ==, <=, >=), etc.

Background: Templates

```
template <class T>
void swap ( T & first, T & second ) {
    T temp = first;
    first = second;
    second = temp;
}
```

- Now can use it to swap two ints, two Rational objects, etc.
- C++ class templates allow creation of generic classes.

C++ Class Templates

```
template <class A, class B>
class pair {  // Useful standard STL class
public:
  A first;
  B second;
  pair ( A a, B b) : first ( a ), second ( b ) { // Constructor
  }
};
```

Now you can declare:

```
pair<string,int> p1 ( “Homer Simpson”, 55 );
pair<string, string> aPhoneBookEntry ( “Charlie Brown”, “853-1234”);
```

STL Container: vector

- Recall that vector is essentially a resizable array
- To use it, you must

```
#include <vector>
using namespace std;
```

- Declaring vector objects

```
vector<int> veci1, veci2; // Creates two empty vectors of int
vector<double> vecd1 (10); // Creates a vector of double of size 10

// vector of 15 Rational initialized to 3/4
vector<Rational> vecRat1 ( 15, Rational ( 3, 4 ));
// Initialize vecRat2 to be identical to vecRat1
vector<Rational> vecRat2 ( vecRat1 );
```

STL Container: vector

Common Usage

- veci1 = veci2; // Assignment operation
- int sz = vecd1.size ( ); // Size of the vector
- if (vecRat1 == vecRat2) …

  // Equality test; will only work if operator ==

  // is defined for Rational objects
- veci1.push_back (10); // It grows
- int value = veci1.pop_back ( ); // And shrinks
- vecRat1[15] = Rational(4, 5); // And it is array-like.

Container Class: queue

Need to #include <queue>

Operations

queue<Rational> qrat, qrat1; // Declare queues of Rational

Rational r1 (5, 6);

qrat.push (r1); // One more item in the queue

cout << qrat.front ( ); // Print the front of the queue;

cout << qrat.size ( ); // Prints the number of items in the queue

Rational r2 = qrat.pop ( ); // Removes the front of the queue

If (qrat == qrat1) … // True if the two queues are equal

STL Container: stack

We are implementing the Back button on a web browser

```
#include <stack>
…
stack<URL> urlStack; // Declare stack of URL objects
…
// When the user goes to a new URL…
// One more URL on the stack
urlStack.push ( newURL );
…
// When the user hits the back button …
// Removes the top of the stack
urlStack.pop ( );
URL currentURL = urlStack.top ( ); // And now, go there!
```

Aggregate Computations

- Need to process the items stored in a container.
- For a vector, this can be done as follows:

```
vector<int> vec;
…
for (int i = 0; i < vec.size( ) ; i++)
    ProcessIt ( vec[ i ] );
```

- Problem: Not all containers provide indexed access to their items.

STL Iterators

- An iterator refers (points) to a position within the container.
- Iterators are container specific, for example, vector<int>::iterator, queue<Rational>::iterator
- Two special positions:
  - The beginning (the zeroth item)
  - The end (***after*** the last item)
- Iterators can be moved around to point at different positions in the container.

STL Iterators

- Finding the sum of student grades

```
vector<double> grades;
…
vector<double>::iterator it; // Declare an iterator object
for ( it = grades.begin ( );  // Initialize iterator to point to zeroth item
      it != grades.end ( );   // Loop as long as not at the end
      it++)                   // Advance the iterator to the next item
  sum = sum + *it;            // Operator * overloaded for iterators
```

- Almost identical code would work for other STL containers

STL Iterators

- Iterators can be used to modify a container at the position that they are pointing to.
  - insert (iterator, item) : inserts the item at the given iterator position
  - erase (iterator): removes the item at the given iterator position
- Example:

```
listOfDbl.insert ( iter, 53.75 ); // Inserts 53.75 at the position “iter”
listOfDbl.erase ( iter ); // Erases the item at the position “iter”
```

STL Container: map

- Simple application: Phone book

Need to #include <map>

```
// A phone book maps names (string) to phone numbers (string)
map<string,string> phoneBook;
phoneBook [ “Homer Simpson” ] = “555-1213”;
string bartsNum = phoneBook [ “Bart Simpson” ];
…
// Print the phone book
map<string,string>::iterator iter;
for (iter = phoneBook.begin ( ); iter != phoneBook.end( ); iter++)
    cout << (*iter).first << “ “ << (*iter).second << endl;
```

STL container: map

- Another application: Word count

```
map<string,int> wordsWithCount; // map each word to its count
string nextWord; // stores the word that is read from the input
…
// Extracted nextWord from the input. First see if it is in the map
map<string,int>::iterator iter;
iter = wordsWithCount.find ( nextWord ); // Try to find it …

if (iter != wordsWithCount.end ( )) // Found! Increment its count
    wordsWithCount[nextWord] ++;
else  // First insertion of this word, so start its count at 1
    wordsWithCount[nextWord] = 1;
```

STL Container: map

- Another example: Creating a text book index

Two possibilities for storing the index.

map<string, set<int> > index;

multimap<string,int> index;

STL Algorithms

- Large collection of generic algorithms

Must #include <algorithm>

```
vector<int> vec;
// Sort in increasing order (default meaning for <)
sort ( vec.begin( ), vec.end ( ) );
…
// Sort in decreasing order, thanks to function decreasing below
sort ( vec.begin( ), vec.end ( ), decreasing );
….
bool decreasing ( int a, int b ) { // User defined function
    return a > b;
}
```

STL Algorithms

- Example: To print a phone book
- Recall that a phone book is declared as:

```
map<string,string> phoneBook;
…
// Function printEntry outputs one entry in the phone book
void printEntry ( const map<string,string>::value_type &
                        phoneEntry)  {
    cout << phoneEntry.first << " " << phoneEntry.second << endl;
}
…
// Use STL's for_each algorithm
for_each ( phoneBook.begin( ), phoneBook.end ( ), printEntry );
```

STL Algorithms

```
list<int> listi;
…
// Example: search for value 76 using find algorithm
list<int>::iterator iter;
iter = find ( listi.begin( ), listi.end ( ), 76 );
if (iter != listi.end ( )) // Found it!
…
// Example: using algo. binary_search for sorted vectors
vector<double> vecd;
if (binary_search ( vecd.begin ( ), vecd.end( ), 76 ))
```

References

- Dave Musser, Gillmer Derge and Atul Saini, “STL Tutorial and Reference Guide”
- STL online:
  - At Silicon Graphics www.sgi.com/tech/stl