Lab2

Purpose:  Use Regular expressions, elementary STL data structures Vector, Stack, Queue

Part1:  
Write a generic XML processor.  Algorithm to process an XML node:  
1. Write a generic XML node class that can handle any XML data.  
2. Process the XML Header  
3. Read and process line of text using Regular Expressions.  This is the master <Node>.  
4. For the remaining lines in the XML file  
 a. Push the node data onto a stack and a Queue  
 b. Read the next line and extract the node data  
  i. If the line contains both the begin and end tag, push\_back the data into a queue  
  ii. If the line is a node terminator, check the top node of the stack.    
  iii. If it has the same tag data, terminate the node and push the node into the vector  
 c. Otherwise push the node data on the node stack    
Example:  
<Airport>                 // tag: push <Airport> on stack  
 <Location>                // tag: push <Location> on stack  
  <Code>ABE</Code>    // element:  push\_back on element queue  
  <City>Allentown</City>   // element:  push\_back on element queue  
  <State>PA</State>        // element:  puch\_back on element queue  
 </Location>               // terminator:  pop stack and complete node in queue  
 <Coordinates>              // tag:  push <Coordinates> on stack  
  <Latitude>40.65</Latitude>   // element:  push\_back on element queue  
  <Longitude>75.43</Longitude> // element:  push\_back on element queue  
  </Coordinates>             // terminator:  pop stack and complete node in queue  
</Airport>                  // terminator:  pop stack and complete node in queue  
     
Part2:  
1. Store the XML nodes an a Linked List.    
2. After processing all the nodes, sort the data using the STL List Sort routine.    
3. Search the list for two specific Airport Codes using a Binary Search algorithm (see Pseudo-code below)  
4. Calculate the distance between the two Airports using the Haversine formula:  
http://www.codecodex.com/wiki/Calculate\_Distance\_Between\_Two\_Points\_on\_a\_Globe

  Algorithm BINARY\_SEARCH(Array,Left,Right,Key)     
   1  if L > R return -1  
   2  M = (L + R) / 2  
   3  if A[M] == K return M  
   4  if A[M] > K return BINARY\_SEARCH(A, L, M-1, K)  
   5  return BINARY\_SEARCH(A, M+1, R, K)