1. Implement a doubly linked list. Use (at least) two class. One for the linked list and one for a nodes in the linked list. Use templates classes for the LinkedList and node classes. The linked list needs to support seven operations: append, prepend, [], contains, isEmpty, iterator, and remove.

   append(x) will place the value x into a node and add the node at the end of the linked list.

   prepend(x) will place the value x into a node and add the node at the front of the linked list.

   contains(x) returns true if the linked list contains the value x in a node, otherwise it returns false.

   isEmpty() returns true if the list is empty, otherwise it returns false.

   remove(x) removes all nodes in the list that contain the value x.

   [n] allows us to access and change the value at the n location in the list (see below). If the list does not have an n'th location an out_of_range exception is thrown. Being C based the first element has index 0.

   iterator() returns an iterator that iterates through the elements of the list from front to back. One can access the elements in the list and change elements in the list using the iterator.

   Here are some sample operations.

```
LinkedList<int> test;
test.isEmpty();         //returns true
test.append(1);
test.append(2);
test.prepend(3);
test.contains(3);       //returns true
int x = test[0];        //x now is 3
test[0] = 10;
int y = test[0];        //y is now 10
ListIterator elements = list.iterator();
int z = *elements;      //z is now 10 as elements refers to the first element of the list
++elements;             // elements now refers to the second element of the list
elements = 20;          // The second element of the list is now 20
elements.atEnd();       // returns false as elements is not at the end of the list.
```

   The ListIterator class needs four operations: *, ++, = and atEnd(). A ListIterator object refers to one element of the list at a time. When first created it refers to the first element of the list. The ++ operator advanced the iterator to the next element of the list. The ++ operator does nothing if the list is already at the end of the list. If the iterator is at the end of the list atEnd() returns true otherwise it returns false. The * operator returns the value of the current element of the list. The = operator is used to change the value of an element of the list.

The iterator does not make a copy of the linked list. It also does not store the elements of the list into an array or any other collection structure.

The Node class needs to support at least the operations append, prepend, delete and contains. If aNode is a node in the linked list then aNode.append(x) will create a new node (call it Z) to hold the value x and attach the new node after aNode. If aNode is not the end of the list then all the nodes after aNode will now be after the node Z. aNode.prepend(x) does the same thing but places the new node before aNode. aNode.delete() removes aNode from the linked list. It will update necessary pointers to maintain the list. It will also delete pointers as needed. The LinkedList class uses the node operations when performing the obvious related operations.

The C++ standard library has a number of classes that provide similar functionality to the above linked list class. You are not to use the classes and iterators from the C++ library in this assignment. In this problem you are also not allowed to use arrays.

2. Create a subclass OrderedStringList of your linked list class. The OrderedStringList will only contain strings. The OrderedStringList always keeps the strings in sorted (lexicographical) ordered. That is append and prepend always place the new element in its correct sorted location. The OrderedStringList also have a function member oddIterator(). It returns an iterator that only iterates over the elements (strings) of the list that are of odd length. That is if the list contains the strings "a", "bb", "ccc" and "dd" the iterator will only consider the strings "a" and "ccc" to be in the list. This iterator does not make a separate list or copy the original list in any way.

3. Grader Program. We have a grade file that contains a name of a student and a list of grades for that student. Here is a sample file:

```
whitney 79 82 99
olson 55 100 92
shah 89 91 94
```

The name and grades are separated by a space. Create a Grades class that contains at least the function members average(String name) and grade(String name). The function member average(String name) returns the average score for the given student. If there is no such student throw a NoSuchStudent exception. The function member grade(String name) returns the letter grade for the given student. If there is no such student throw a NoSuchStudent exception. Letter grades are given by the chart below.

Letter Grade	
A	90-100%
B	80-89%
C	70-79%
D	60-69%
F	0-59%