CSC 2430 – Data Structures 1
Homework #7 – "By Position" and "Sorted" Linear Single-Linked Lists
Due: Friday, March 11, 2016, in class

The purpose of this lab assignment is to study and modify "by position" and "sorted" single-linked linear lists of integer values.

Download from Blackboard three C++ source files containing an initial starting program for the lab:

- **LabListP.h** and **LabListP.cpp**
  Defines and implements **ListClass**, a class containing a "by position" linked list of nodes.
  The LabListP.cpp file is completely implemented except for the assignment operator=( ) method.

- **LabListMain.cpp**
  An initial version of the main program for this assignment which:
  a. creates a **ListClass** variable named "**`listbyposition`**"
  b. creates and inserts a collection of random data values into consecutive positions in this list
  c. prints out the list.

Create a Visual Studio solution project for this lab using these three source files. After creating an empty Console-application project, copy the three source files into the project subdirectory (that's the inner subdirectory with the solution) and use the menu "Project/Add Existing files" to select them into the project. The project solution should correctly build and execute, with output similar to:

```
Linked List Lab: Implemented by Mike Tindall
Randomly generate list of 10 values and convert into a sorted list

listbyposition: 10 items
1: 69
2: 65
3: 68
4: 96
5: 22
6: 49
7: 67
8: 51
9: 61
10: 63
```

Change the initial main program to print out your own name (instead of Mike Tindall).

Your assignment:
Create a new class "**SortedListClass**" that is functionally similar to **ListClass**, except the insert( ) method is changed so that it inserts new values into its list "by value" in ascending sorted order rather than specifying the position with a parameter. The SortedListClass must also include an additional method "find( )", that searches the sorted list of data values and returns it's position within the list of values, and SortedListClass must have a fully implemented assignment operator=() method. Then modify the LabListMain.cpp main program as described in the following instructions (step 7) below to test out the SortedListClass.

1. Create copies of the initial **LabListP.h** and **LabListP.cpp** files named as **SortedLabList.h** and **SortedLabList.cpp**. Add these two new files to your project.

2. Modify the new SortedLabList header and implementation files so that they define and implement the "**SortedListClass**" class (instead of the original "**ListClass**" class).

3. Change the declaration and implementation of the insert( ) method in SortedListClass to insert a new item data value into the linked list "by value", e.g.,

   ```
   bool  insert(ListItemType& newItem);
   ```

   The actual position of the new node in the linked list is not specified with a parameter, but instead the method must search through the existing list of items and insert the new node into its proper position so that the sequence of data values in the nodes remain in ascending order.

4. Add a new method declaration and implementation to SortedListClass:

   ```
   int  find(ListItemType& dataItem) const;
   ```

   find(x) searches through the linked list to find dataItem x, and returns its position within the list of items. Positions in the list are numbered beginning with position 1 (e.g., positions are from 1 .. k). If the dataItem is not found anywhere in the list, then find(x) returns -1.

5. **Complete the implementation of the assignment operator= ( ) method in SortedListClass**. (If you want, you may also complete this method implementation in the original ListClass, although this is not required for this lab).

6. Note that your lab project will now have class implementations for both the original ListClass data type and also the new SortedListClass data type.

7. Make the following changes to the ListLabMain.cpp main( ) program:
   a. #include "SortedLabList.h" along with the original #include "LabListP.h".
   b. After creating the initial unsorted list of random data values in the listbyposition variable, create a new variable of type SortedListClass.
   c. Insert the same set of data values into your sorted list variable.
      i. To do this, the main() program must iterate through the listbyposition data to retrieve each of the data values one at a time, and then insert each data value into the sorted list variable.
      ii. Once the full set of data values have been inserted into the sorted list, then output the contents of the sorted list variable in a similar format as the listbyposition contents in the original output listing (e.g., as shown in the example on page 1).
   d. Produce another output table listing that shows for each item in the original listbyposition, what position that item ended up being located in within the sorted list variable's linked list.
   e. Create another SortedListClass variable (using the default constructor for an empty list).
   f. Use the assignment operator=( ) to make this second sorted list variable a copy of your first sorted list variable.
   g. Then remove the first two items from your first sorted list variable (e.g., list.remove(1) twice).
   h. Then insert the data value (-10) into the second sorted list variable.
   i. Now, print out the first sorted list and then the second sorted list.

Hand-In:

1) Printed source code listings of your **SortedListLab.h** header file, your **SortedListLab.cpp** implementation file, and your modified **LabListMain.cpp** main program file.

2) The printed output from running your modified **LabListMain.cpp** main( ) program (which should look similar to the example below).

```
Linked List Lab: Implemented by Mike Tindall
Randomly generate list of 10 values and convert into a sorted list

listbyposition: 10 items
1: 69
2: 65
3: 68
4: 96
5: 22
6: 49
7: 67
8: 51
9: 61
10: 63

sortedlist: 10 items
1: 22
2: 49
3: 51
4: 61
5: 63
6: 65
7: 67
8: 68
9: 69
10: 96

Correlation between locations of items in listbyposition and sortedlist
listbyposition[  1] =    69 --> sortedlist[  9]
listbyposition[  2] =    65 --> sortedlist[  6]
listbyposition[  3] =    68 --> sortedlist[  8]
listbyposition[  4] =    96 --> sortedlist[ 10]
listbyposition[  5] =    22 --> sortedlist[  1]
listbyposition[  6] =    49 --> sortedlist[  2]
listbyposition[  7] =    67 --> sortedlist[  7]
listbyposition[  8] =    51 --> sortedlist[  3]
listbyposition[  9] =    61 --> sortedlist[  4]
listbyposition[ 10] =    63 --> sortedlist[  5]

Modified original sortedlist after removing the first two items:
sortedlist: 8 items
1: 51
2: 61
3: 63
4: 65
5: 67
6: 68
7: 69
8: 96

Modified sortedlistcopy after inserting the value (-10):
sortedlistcopy: 11 items
1: -10
2: 22
3: 49
4: 51
5: 61
6: 63
7: 65
8: 67
9: 68
10: 69
11: 96
```