CSCI 2720 Assignment 1 - **Sorted Linked List**

Due: **July 1, 2018 11:59PM**

You can do this assignment individually or as a group of two people.

In this assignment you will create a **Sorted Singly-Linked List** that performs basic list operations using C++. This linked list should not allow duplicate elements. Elements of the list should be of type 'DataType'. 'DataType' class should have a private integer variable with the name 'value'. Elements in the linked list should be sorted in the ascending order according to this 'value' variable.

You should create a command line application (Main.cpp) to utilize the linked list. This application should take a single command line parameter. This parameter should be a **path to a plain text file** that contains a space separated list of positive or negative integer numbers in an unsorted order. The main application should read this text file and insert the values to the sorted linked list.

The command line application should provide an interactive command line interface to allow the users to perform operations on the list. **You must implement all the operations that are shown in the example output**. **And the command line interface must be exactly equal to the example output** given at the end of this document.

This assignment includes one bonus question.

You must create the following files to implement the program.

- DataType.h
- DataType.cpp
- SortedLinkedList.h
- SortedLinkedList.cpp
- ListNode.h
- Main.cpp

You must create the following **mandatory variables and functions** in the above files. You may add your own functions and variables in addition to the following functions.

DataType.h

Enumerations:

- There should be an enumeration called '**Comparison**' with values GREATER, LESS, and EQUAL. This enumeration should be used when comparing the 'DataType' elements when sorting.

Private Data Members:

- int value

Functions:

- **DataType(int value)**
  Initialise the DataType object with the constructor parameter as the value.
- **Comparison compareTo(DataType &item)**
  Compare the value of item with the current object's value and return GREATER, LESS or EQUAL.
- **int getValue() const**
  Return the value instance variable.

DataType.cpp

This file should provide implementations for the members in Datatype.h.

ListNode.h

You should create a struct called ListNode to be used as the Nodes in the linked list.

Public Data Members:

- DataType item
- ListNode *next

Functions:

- **ListNode(DataType &item)**
  Initialise a ListNode instance with the item parameter.

SortedLinkedList.h

Private Data Members:

- int count
- ListNode *head

Functions:

- **SortedLinkedList()**
  Initialise a sorted linked list object.
- **~SortedLinkedList()**
  Free up the used memory and destruct the SortedLinkedList instance. This method should handle memory leaks.
- **int length() const**
  Return the length of the linked list.
- **void insertItem(DataType &item)**
  item should be inserted to the linked list maintaining the ascending sorted order.
    - You have to handle inserting the first element as a special case.

- **void deleteItem(DataType &item)**
  ListNode that contains an item equal to the item parameter should be removed.
  You should handle all cases of deleting an element.
    - Deleting first element.
    - Deleting the last element or an element in the middle.
    - Deleting the only element.
    - Attemt to delete a non-existing item should print “Item not found”.
- **int searchItem(DataType &item)**
  Search the ListNode that contains an item equal to the parameter item and return its index. Print “Item not found” if the item was not found in the list.
- **void clear()**
  Remove all elements in the linked list and free up the memory. This method should handle memory leaks.
- **void reverse()**
  You should implement this method only if you wish to complete the bonus question.
  In this method you should reverse the linked list without using a separate list or an array.

SortedLinkedList.cpp

You should implement the members in the SortedLinkedList.h in this file.

Main.cpp

You should implement the command line application in this file. It should be able to take the input file in the following command line format.

```
$ ./main inpt.txt
```

You can use the following code snippet to read numbers from the input file.

```
#include <fstream>
#include <iostream>
CircularLinkedList list;
ItemType item;
int input;
std::fstream fs;
fs.open(argv[1], std::fstream::in);
if (fs.is_open()) {
  fs >> input;
  while (!fs.eof()) {
    item.initialize(input);
    list.insertItem(item);
    fs >> input;
  }
```

```
    } else {
      std::cout << "Failed to open the input file" << std::endl;
      return 0;
    }
```

Your application must use the following character constants as the commands in the command line interface.

- INSERT = 'i',
- DELETE = 'd'
- SEARCH = 's'
- ITR_NEXT = 'n'
- RESET_ITR = 'r'
- PRINT_ALL = 'p'
- LENGTH = 'l'
- CLEAR_ALL = 'c'
- REVERSE = 'b'
- QUIT = 'q'

‘n’ (ITR_NEXT) and ‘r’ (RESET_ITR) Command Explanation

Repeated invocations of ‘n’ command should print elements in the list one by one, starting from the first element to the last element. If ‘n’ is invoked at the last element, you should print “The end of the list has reached”. Also if the ‘n’ is invoked when the list is empty, you should print “List is empty”.

Invoking the ‘r’ command should cause the ‘n’ command to start printing the elements starting from the beginning in the consequent invocations.

Example Output:

Command line interface of your program must be exactly equal to the following examples. As shown in the example output, if any of the commands are going to modify the elements in the list, you must print the elements in the list before and after the modification is performed.

```
./main input.txt Commands:
      (i) - Insert value
      (d) - Delete value
      (s) - Search value
      (n) - Print next iterator value (r) - Reset iterator
      (p) - Print list
      (l) - Print length
      (b) - Reverse
      (c) - Clear list
      (q) - Quit program

Enter a command: i
```

```
1 3 9 10 19 37 45 63 84 100
Enter number: 12
1 3 9 10 12 19 37 45 63 84 100

Enter a command: d
1 2 3 9 10 12 19 37 45 63 84 100
Enter value to delete: 12
1 2 3 9 10 19 37 45 63 84 100

Enter a command: s
Enter a value to search: 10
Index 4

Enter a command: n
1

Enter a command: n
2

Enter a command: r
Iterator reset.

Enter a command: n
1

Enter a command: p
1 2 3 9 10 19 37 45 63 84 100

Enter a command: l
List Length is 11

Enter a command: b
Before
1 2 3 9 10 19 37 45 63 84 100
After
100 84 63 45 37 19 10 9 3 2 1

Enter a command: c
List cleared

Enter a command: q
Quitting program..
```

Grading Rubric

Insert item	20
Delete item	20
Search an item	20
Length	5
Next and reset commands	10
Clearing the list	5
Printing the list	5
Comments, Code formatting and Readme.txt	5
Following the specification	5
Handling Memory leaks	5
Bonus question	20
Total	**120**

Compiling the Program:

You must create a Makefile to compile your program. The Makefile must compile your code into an executable program called "**main**".

Your program should run with the following command syntax:

```
$ ./main <input file name>
```

Commands to run and compile the code should be documented clearly in the Readme file.

Code that fails to compile or the code that compiles but fails to run will receive a grade of a zero.

Late Submission Policy:

Except in the cases of serious illness or emergencies, projects must be submitted before the specified deadline in order to receive full credit. Projects submitted late will be subject to the following penalties:

- If submitted 0–24 hours after the deadline 20% will be deducted from the project score
- If submitted 24–48 hours after the deadline 40% points will be deducted from the project score.
- If submitted more than 48 hours after the deadline a score of 0 will be given for the project.

Students unable to submit a project due to a serious illness or other emergency should contact the instructor as soon as possible before a project's deadline. Based upon the circumstances, the instructor will decide an appropriate course of action.

Submission Notes:

You must include your full name and university email address in your Readme.txt file. If your are doing the project in a group of two, list the full names and the email addresses of all two group members. If you are in a group you must also describe the contributions of each group member in the assignment.

Submit the following files on Nike:

- DataType.h
- DataType.cpp
- SortedLinkedList.h
- SortedLinkedList.cpp
- ListNode.h
- Main.cpp
- Makefile
- Readme.txt

Group Submissions

If you are working in a group of two, your submission folder must be names as below,

LastName1_LastName2_**assignment1**

When you are submitting the folder you should use the submit command as below,

`submit LastName1_LastName2_`**`assignment1`** `cs2720a`

If you are working in a group, **you must submit the assignment only once** from the Nike account of one of the members in the group.

Individual Submissions

If you are working individually, your submission folder must be named as below,

LastName_**assignment1**

Submission command should be used as below

submit LastName_**assignment1** cs2720a

****For all the submissions, the folder naming and the readme is very important in order to be able to locate your assignment and grade it.**

When using the submit command, first set your current directory of the terminal to the immediate parent of the assignment folder.

For example if your assignment is in the following directory,

`/documents/csci2720/assignments/LastName_`**`assignment1`**

Before executing the submit command, execute:

```
cd /documents/csci2720/assignments/
```