CSCI 2720
Assignment 2 - Sorted Linked List
Due date: October 1, 2021 11:59 PM

You can complete 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 'ItemType'. 'ItemType' 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 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 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 like the example output** provided at the end of this document.

You must create the following files to implement the program.

- ItemType.h
- ItemType.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.

ItemType.h

Enumerations:

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

Private Data Members:

- **int value**

Functions:

- **ItemType( )** - Default Constructor:
- **Comparison compareTo(ItemType 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.
- **void initialize(int num)** - Initializes the data member by variable num

ItemType.cpp

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

ListNode.h

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

Public Data Members:

- **ItemType item**
- **ListNode *next**

SortedLinkedList.h

Private Data Members:

- **ListNode *head**
- **ListNode *currentPos**

Functions:

- **SortedLinkedList( )** - Initialise a sorted linked list object.
- **~SortedLinkedList( )** - Free up all the user allocated memory and destruct the SortedLinkedList instance.
- **int length( ) const** - Return the length of the linked list.
- **void insertItem(ItemType item)** - item should be inserted to the linked list maintaining the ascending sorted order.
    - General Case: Insert at the middle or end.
    - Special Cases:
        - Insert the first element
        - Insert in an empty list
        - Print ***"Sorry. You cannot insert the duplicate item"*** when the user tries to insert duplicate item

- **void deleteItem(ItemType item)** - ListNode that contains an item equal to the item parameter should be removed. You should handle all cases of deleting an element.

    - General Case: Deleting the last element or an element in the middle.
    - Special Cases:
        - Deleting the first element.
        - Deleting the only element.
        - Attempt to delete a non-existing item should print ***"Item not found"***.
        - Attempt to delete from an empty list should print ***"You cannot delete from an empty list"***.
- **int searchItem(ItemType 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.
- **ItemType GetNextItem( )** - This function returns the next item in the list pointed by the currentPos pointer.

  Notice: The code in the slides is not correct in this case because it didn't check the end of the list or the empty list.
    - Print ***"List is empty"*** when the list is empty
    - Print ***"The end of the list has reached"*** when reach the end of the list or you can start printing the list again when end of list is reached (i.e go back to the first element from last last element)
- **void ResetList( ) -** This will initialize the currentPos pointer to null. See the description below to check how you need to call these functions.

The following functions can be implemented however you like. However, you should make sure their input and output formatting matches the sample output below. Implement these functions as class functions using prototypes of your choice.

- **Merge Function -** This function will take another list as input, merge the input list to the original list, and then print the output
    - Note: Print ***"Sorry. You cannot insert the duplicate item"*** when the user tries to merge lists with duplicate items. The merge should then fail and not modify the original list.
    - Note: This should modify the original list
    - Example:

      List 1: 9 13 36 47

      List 2: 3 45 89 96

      3 9 13 36 45 47 89 96
    - In the readme file give the pseudo code (steps) for your merge operation. Using this pseudocode, explain the complexity (big O) of your merge operation.
    - **Note: We are not looking for the best or most efficient solution for the merge problem. We just want a solution from you but you should comment on its complexity.**
- **Delete Alternate Nodes** - This function will delete alternate nodes from the list.
    - Note: This should skip the first node, delete the second, skip the third, delete the fourth and so on.
    - Example:

List before alternate delete: 3 7 14 26 74 78

List after alternate delete: 3 14 74

- **Find common elements function -** This function will take another list as input, find the common elements between input list and original list, and then print the output
    - Note: This should modify the original list
    - Example:

      List 1: 2 4 14 16 35 47 54 83

      List 2: 1 3 4 15 35 54 74 91

      Intersection: 4 35 54
    - Like the merge function in the readme file, give the pseudo-code (steps) for 'find common elements' operation. Using this pseudocode, explain the complexity (big O) of your 'find common elements function' operation.
    - **Note: We are not looking for the best or most efficient solution for the intersection problem. We just want a solution from you but you should comment on its complexity.**

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 input.txt
```

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

- INSERT = 'i' Inserts a node in the linked list
- DELETE = 'd' Deletes a node in the linked list
- SEARCH = 's' Searches a node in the linked list
- ITR_NEXT* = 'n'
- RESET_ITR* = 'r'
- DEL_ALT = ‘a’
- MERGE = ‘m’
- INTER = ‘t’
- PRINT_ALL = 'p' Should print all the integer values stored in the linked list nodes
- LENGTH = 'l'
- QUIT = 'q' Quit the program

** If the user enters anything other than the option listed above, you should print ***“Invalid command, try again!”***

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

1. 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”*** or or you can start printing the list again when end of list is reached (i.e go back to the first element from last last element)
2. Also if the ‘n’ is invoked when the list is empty, you should print ***“List is empty”***. Assume that the user is not going to add or delete an item from the list while calling the iterator.
3. Invoking the ‘r’ command should cause the ‘n’ command to start printing the elements starting from the beginning in the consequent invocations. You need to call GetNextItem() function for command ‘n’ and ResetList functions for command ‘r’.

Sample Output:

Please download the txt file from the ELC folder.

- input.txt
- empty.txt

If you are not able to read correctly from the given txt file, please create a text file with the exact same format in Odin unix environment to test your program.

The command-line interface of your program must be exactly equal to the following examples. As shown in the Sample 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.
Notice: You should NOT hard-code any information to get the sample output.

Sample Output 1 (input.txt)

```
./main input.txt Commands:
(i) - Insert value
(d) - Delete value
(s) - Search value
(n) - Print next iterator value
(r) - Reset iterator
(a) - Delete alternate nodes
(m) - Merge two lists
(t) - Intersection
(p) - Print list
(l) - Print length
(q) - Quit program
```

```
//1. Test INSERT(i)
//1a. Insert the first element
Enter a command: i
3 9 10 19 37 45 63 84 100
Enter number: 1
1 3 9 10 19 37 45 63 84 100
//1b. Insert at the middle/end
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

//1c. Insert duplicate item
```

```
Enter a command: i
1 3 9 10 12 19 37 45 63 84 100
Enter number: 3
Sorry. You cannot insert the duplicate item.
1 3 9 10 12 19 37 45 63 84 100
```

```
//1d. Insert Item in an empty list
//     (Check Sample Output 3)
```

//2.Test DELETE(d)

```
//2a. Delete last element or an element in the middle
```

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

```
//2b. Delete first element
```

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

```
//2c. Delete the a non-existing element
```

```
Enter a command: d
3 9 10 12 19 37 45 63 84
Enter value to delete: 90
Item not found.
3 9 10 12 19 37 45 63 84
```

```
//2d. Delete the only element
//     (Check Sample Output 3)
//2e. Delete from an empty list
//     (Check Sample Output 3)
```

//3. Test DEL_ALT(a)

```
Enter a command: a
List before alternate delete: 3 9 10 12 19 37 45 63 84
List after alternate delete: 3 10 19 45 84
```

//4. Test MERGE(m)

```
//4a. Merge a list to the original list
```

```
Enter a command: m
Length of list to merge: 3
```

```
List elements separated by spaces in order: 11 20 40
List 1: 3 10 19 45 84
List 2: 11 20 40
3 10 11 19 20 40 45 84
```

//4b. Merge a list with duplicate items

```
Enter a command: m
Length of list to merge: 3
List elements separated by spaces in order: 3 10 25
List 1: 3 10 11 19 20 40 45 84
List 2: 3 10 25
Sorry. You cannot insert the duplicate item.
3 10 11 19 20 40 45 84
```

//5. Test INTERSECTION(t)

//5a. Intersection of two lists

```
Enter a command: t
Length of list to find intersection: 4
List elements separated by spaces in order: 2 10 19 25
List 1: 3 10 11 19 20 40 45 84
List 2: 2 10 19 25
Intersection: 10 19
```

//5b. Intersection of two lists with no common elements

```
Enter a command: t
Length of list to find intersection: 4
List elements separated by spaces in order: 5 17 32 41
List 1: 10 19
List 2: 5 17 32 41
Intersection:
```

Sample Output 2 (input.txt)

```
./main input.txt Commands:
(i) - Insert value
(d) - Delete value
(s) - Search value
(n) - Print next iterator value
(r) - Reset iterator
(a) - Delete alternate nodes
(m) - Merge two lists
(t) - Intersection
(p) - Print list
(l) - Print length
(q) - Quit program
Enter a command: p
3 9 10 19 37 45 63 84 100
```

//1. Test SEARCH(s)

//1a. Search an element that's in the list

```
Enter a command: s
Enter a value to search: 10
Index 2
```

//1b. Search an element that's not in the list

```
Enter a command: s
Enter a value to search: 1
Item not found.
```

//2. Test IRR_NEXT(n)

//2a. General Case:

```
Enter a command: n
3
Enter a command: n
9
Enter a command: n
10
```

//… Once you reach the end of the list…

//2b. Invoke 'n' at the last element

```

Enter a command: n
```

```
100
Enter a command: n
The end of the list has been reached
```

```
//2c. Invoke 'n' when list is empty
//     (Check Sample Output 3)
```

//3. Test RESET_ITR(r)

```
Enter a command: r
Iterator reset.
Enter a command: n
3
```

//4. Test PRINT_ALL(p)

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

//5. Test LENGTH(l)

```
Enter a command: l
List Length is 9
```

//6. Test invalid command

```
Enter a command: 9
Invalid command, try again!
```

//7. Test QUIT(q)

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

Sample Output 3 (empty.txt)

```
./main empty.txt Commands:
(i) - Insert value
(d) - Delete value
(s) - Search value
(n) - Print next iterator value
(r) - Reset iterator
(a) - Delete alternate nodes
(m) - Merge two lists
(t) - Intersection
(p) - Print list
(l) - Print length
(q) - Quit program
Enter a command: p
```

//1. Test INSERT(i)

//1a. Insert Item in an empty list

```
Enter a command: i

Enter number: 2
2
```

//2.Test DELETE(d)

//2a. Delete the only element

```
Enter a command: d
2
Enter value to delete: 2
```

//2b. Delete from an empty list

```
Enter a command: d

Enter value to delete: 9
You cannot delete from an empty list.
```

//3. Test IRR_NEXT(n)

//3a. Invoke 'n' when list is empty

```
Enter a command: n
List is empty
```

Grading Rubric

Insert item	10
Delete item	10
Search an item	10
Merge (along with complexity discussion)	10
Delete alternate	10
Find Common Elements (along with complexity discussion)	10
Length	5
Next and reset commands	10
Clearing the list (destructor)	5
Printing the list	5
Comments, Code formatting and Readme.txt	5
Following the specification	5
Handling Memory leaks	5
Total	**100**

- Make sure to run valgrind and check for any memory leaks before submitting your assignment.

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.

The code that fails to compile or the code that compiles but fails to run will receive a grade of 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 emergencies should contact the instructor as soon as possible before a project's deadline. Based on the circumstances, the instructor will decide on an appropriate course of action.

Submission Notes:

You must include your full name and university email address in your Readme.txt file. If you are doing a project in a group of two, list the full names and 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 Odin:

- ItemType.h
- ItemType.cpp
- SortedLinkedList.h
- SortedLinkedList.cpp
- ListNode.h
- main.cpp
- Makefile
- Readme.txt

Odin Login Instructions

This semester we are switching from Nike to a new server called Odin. Unlike with Nike, to use Odin, you must use the UGA VPN first. Please visit remote.uga.edu if you haven't already and install the VPN client for your OS. Be sure to have an SSH client like PuTTY or MobaXterm and use that to connect. The server's name is odin.cs.uga.edu and you will log in with your MyID credentials. For those that have used Nike before, the login process is almost the same, except that nike.cs.uga.edu is replaced by odin.cs.uga.edu, and, as stated before, you'll log in with your MyID credentials instead of your regular Nike credentials.

Group Submissions:

****Note: Assignment can be done individually or in a group of two students. The members of the group should be from the same section. Submit a readme file with your names, UGA emails and any other information that you want to give the TA to run your code. If you are doing it in a group of two make sure to include the contribution of each student in the readme file.**

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

```
LastName1_LastName2_assignment1
```

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

```
$ submit LastName1_LastName2_assignment1 <odin-submission-destination>
```

Make sure to replace `<odin-submission-destination>` with the correct destination from the table on the bottom of this document.

If you are working in a group, you must **submit the assignment from the Odin account of only 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 <odin-submission-destination>
```

Make sure to replace `<odin-submission-destination>` with the correct destination from the table on the bottom of this document.

****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.**

****Note: Make sure you submit your assignment to the right folder to avoid grading delays.**

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

You can check your current working directory by typing:

```
$ pwd
```

For example if your assignment is in the following directory,

```
/documents/csci2720/assignments/LastName_assignment1
```

Before executing the submit command, change to the assignment's immediate parent directory:

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

Then execute the submit command:

```
$ submit LastName_assignment1 <odin-submission-destination>
```

Make sure to replace `<odin-submission-destination>` with the correct destination from the table on the bottom of this document.

To check if your submission command was valid, there should be a receipt file.

```
$ ls LastName_assignment1
// should return rec##### like 'rec54321'
```

Selecting the Odin Submission Destination

You must select your Odin submission destination using the following mapping based on the section of the course you are registered to.

Section	Class Time	Odin Submission Destination
CRN 17504	M 10:20 AM	csci-2720a
CRN 42634	M 04:10 PM	csci-2720c