CSCI 2720 – Fall 2025

Programming Assignment 1: Sorted Linked List

Due by 11:59 PM on September 19th, 2025 on Odin on Odin

You can complete this assignment individually or as a group of two students.

1 Project Setup

Your source code should be kept under a package `cs2720.p1`. You are free to setup your project using Maven (recommended) or by creating the directory structure manually. Regardless, you must briefly document how to compile and run your project in your README file.

2 Project Description

In this assignment, you will create a **Sorted Singly-Linked List** using the Java programming language.

1. The linked list should **not** allow duplicate elements.
2. Elements of the list should be of type `ItemType`. The `ItemType` class should have a `private int` variable with the name `value`.
3. Elements in the linked list should be sorted in ***ascending*** order according to this `value` instance variable.
    - E.g., the list `NodeType(1)` → `NodeType(2)` → `NodeType(3)` → ∅.

Your driver file (containing the `main` method) should be a command-line interface that allows you to perform list operations. Your driver should take a single command-line argument, representing a **path to a plain text file** that contains a space-separated list of unordered, positive integers. The application should first read this text file and insert the values into the sorted linked list.

You must implement all the operations that are shown in the example output. Further, 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.java
- SortedLinkedList.java
- NodeType.java
- LinkedListDriver.java

Note: You are free to add additional packages/Java classes as long as you follow all functional and non-functional requirements.

2.1 File I/O

There are many different ways to parse files in Java. Because input files used in this project will exclusively contain integers, it is recommended that you use a Scanner.

3 Functional Requirements

A functional requirement is *added* to your point total if satisfied.

You must create the following **mandatory variables and methods** in the aforementioned files. You may add your own methods and variables as long as they follow the other project requirements.

ItemType.java

Private Data Members:

- `int value`

Public Functions:

- `int compareTo(ItemType item)` - Compares the value of item with the current object's value and returns -1 if the value of the current object is less than value in item, 0 if equal, and 1 if greater.
- `int getValue()` - Returns the value instance variable.
- `void setValue(int num)` - Initializes the data member by variable num.

Note: If you want to use a constructor instead of the function `setValue`, you can do that, too.

NodeType.java

You should create a class called `NodeType` to create nodes in the linked list.

Private Data Members:

- `ItemType info`
- `NodeType next`

Feel free to implement public constructors/methods however you like.

SortedLinkedList.java

Private Data Members:

- `NodeType head`

Constructors:

- `SortedLinkedList()` - Initializes a sorted linked list object.

Public functions:

- `int getLength()` - Returns the length of the linked list. You need to do a loop over the list and get the number of nodes in the list. You **may not** use an instance variable counter to keep track of the length (this is to help you practice traversing linked lists).

- `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
    * Throw an appropriate unchecked exception if the user tries to insert a duplicate item.
- `void deleteItem(ItemType item)` – The node in the list 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.
    * Throw an appropriate unchecked exception if the user tries to delete an item that does not exist.
    * Throw an appropriate unchecked exception if the user tries to delete from an empty list.
- `int searchItem(ItemType item)` - Search the linked list that contains an item equal to the parameter item and return its index.
  - Throw an appropriate unchecked exception if the user tries to search for an item that does not exist.

The three methods below can be implemented however you like. However, you should make sure that their input and output formatting match the sample output below. Implement the following functions in your singly linked list class.

- `mergeList` - This function should merge two lists and not include any duplicate items in the list. If there are duplicates in the two lists, the merge function should keep only one of the duplicate instances in the resulting list.
  - Example:
    List 1: 9 13 45 36 47 89
    List 2: 3 45 89 96
    Merged list: 3 9 13 36 45 47 89 96
  - See sample output below for more examples.
  - 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 the most efficient solution for the merge problem. We just want a solution that is correct, but you should comment on its complexity.
- `Delete Alternate Nodes` - This function will delete alternate nodes from the list. It should skip the first node, delete the second, skip the third, delete the fourth and so on. You are not allowed to create a new list in this function. Just modify the original list by deleting the nodes in it.
  - Example:
    List before alternate delete: 3 7 14 26 74 78
    List after alternate delete: 3 14 74
  - See sample output below for more examples.
  - The complexity of this function should not be more than $O(n)$.

- `Intersection` - This function uses another list, finds the common elements between input list and original list, and then prints the result.
  - 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
  - See sample output below for more examples.
  - Like the merge function in the README file, give the pseudo-code (steps) for the "Intersection" function. Using this pseudocode, explain the complexity (Big-O) of your "Intersection" function.

  Note: We are not looking for the best or most efficient solution for the intersection problem. We just want a solution that is correct, but you should comment on its complexity.

In methods `merge` and `intersection`, you need to read a second list from the user. You don't need to read the inputs of this second list from a text file, read the inputs directly from the user as shown in the sample output below.

3.1 Exception Handling

Some node/list operations may throw exceptions, which is easy to check for in a unit test (using something like `assertThrows`). Whenever an unchecked exception is expected, we do not have a preference for which exception you choose to throw. However, you should use the most appropriate exception type possible. For example, trying to delete a non-existent item would be more appropriate for an `IllegalArgumentException` than an `ArithmeticException`.

Your `main` method should ensure that the program does not crash after list-related exceptions are thrown. Instead, you should print the exception message as seen in the example output.

You can assume that an `ItemType` object will never be `null`.

LinkedListDriver.java

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.

```
$ java -cp bin cs2720.LinkedListDriver input.txt
```

Using Maven:

```
$ mvn exec:java -Dexec.mainClass="cs2720.LinkedListDriver" -Dexec.args="input.txt"
```

Note: Your driver program can be under a different package as long as it is clear how to run it in your README file.

Make sure your code does error checking for file I/O operations (i.e., it should throw an error message if the requested file is not present or if it cannot open the file correctly). You should not hardcode the input file name in the `main` method.

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

Operation	Character
INSERT	'i'
DELETE	'd'
SEARCH	's'
DEL_ALT	'a'
MERGE	'm'
INTERSECTION	't'
PRINT_ALL	'p' prints all the numbers in the list
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!"***

4 Sample Output

Please download the txt file from the eLC folder.

- input.txt

If you are not able to read from the given txt file, please create a text file with the exact same inputs on Odin to test your program.

The command-line interface of your program must exactly match 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 hardcode any information like file name to get the sample output.

For the following examples, the green "// comments" would not be printed by your program.

Blue text is entered by the user.

Sample Output 1 (input.txt)

```
Commands:
(i) - Insert value
(d) - Delete value
(s) - Search value
(a) - Delete alternate nodes
(m) - Merge lists
(t) - Find intersection
(p) - Print list
(l) - Print length
(q) - Quit program


//1. Test PRINT(p)
Enter a command: p
The list is: 1 3 5 8 10 12 20
```

```
//2. Test LENGTH(l)
Enter a command: l
The length of the list is 7


//3. Test INSERT(i)
//3a. Insert at the beginning of the list
Enter a command: i
Enter a number to insert: 0
Original list: 1 3 5 8 10 12 20 // List before insert
New list: 0 1 3 5 8 10 12 20 // List after insert


//3b. Insert at the middle/end
Enter a command: i
Enter a number to insert: 14
Original list: 0 1 3 5 8 10 12 20
New list: 0 1 3 5 8 10 12 14 20
Enter a command: i
Enter a number to insert: 34
Original list: 0 1 3 5 8 10 12 14 20
New list: 0 1 3 5 8 10 12 14 20 34


//3c. Insert duplicate item
Enter a command: i
Enter a number to insert: 34
Original list: 0 1 3 5 8 10 12 14 20 34 // List before insert
Item already exists
New list: 0 1 3 5 8 10 12 14 20 34 // List after insert
//3d. Insert an item in an empty list
// (Check sample output after delete function)


//4.Test DELETE(d)
//4a. Delete first element
Enter a command: d
Enter a number to delete: 0
Original list: 0 1 3 5 8 10 12 14 20 34 // List before delete
New list: 1 3 5 8 10 12 14 20 34 // List after delete
```

```
//4b. Delete from the middle/end
Enter a command: d
Enter a number to delete: 14
Original list: 1 3 5 8 10 12 14 20 34
New list: 1 3 5 8 10 12 20 34
Enter a command: d
Enter a number to delete: 34
Original list: 1 3 5 8 10 12 20 34
New list: 1 3 5 8 10 12 20
```

```
//4c. Delete an item that does not exists
Enter a command: d
Enter a number to delete: 34
Original list: 1 3 5 8 10 12 20
The item is not present in the list
New list: 1 3 5 8 10 12 20
```

```
//4d. Delete the only item
Enter a command: d
Enter a number to delete: 20
Original list: 20 // List before delete
New list: // List after delete
```

```
//4e. Delete from an empty list
Enter a command: d
Enter a number to delete: 20
You cannot delete from an empty list
```

```
//3d. Insert an item in an empty list
Enter a command: i
Enter a number to insert: 22
Original list: // List before insert
New list: 22 // List after insert
```

```
//5. Test SEARCH(s)
//5a. Item is present in the list
Enter a command: s
Enter a number to search: 3
Original list: 1 3 5 8 10 12 20
The item is present at index 1
Enter a command: s
Enter a number to search: 20
Original list: 1 3 5 8 10 12 20
The item is present at index 6


//5b. Item is not present in the list
Enter a command: s
Enter a number to search: 14
Original list: 1 3 5 8 10 12 20
Item is not present in the list


//5c. Empty list
Enter a command: s
Enter a number to search: 22
Original list:
The list is empty


//6. Test DEL_ALT(a)
//6a. Odd length
Enter a command: a
Original list: 1 3 5 8 10 12 20
New list: 1 5 10 20


//6b. Even length
Enter a command: a
Original list: 1 5 10 20
New list: 1 10


//6c. Length two
Enter a command: a
Original list: 1 10
New list: 1
```

```
//6d. Length one
Enter a command: a
Original list: 1
New list: 1


//6e. Empty list
Enter a command: a
Original list:
The list is empty
Modified list:


//7. Test MERGE(m)
//7a. Merge a list to the original list
Enter a command: m
Enter the length of the new list: 3
Enter the numbers: 17 4 33
list 1: 1 3 5 8 10 12 20
list 2: 4 17 33
Merged list: 1 3 4 5 8 10 12 17 20 33


//7b. Merge a list with duplicate items
Enter a command: m
Enter the length of the new list: 4
Enter the numbers: 2 5 15 24
list 1: 1 3 5 8 10 12 20
list 2: 2 5 15 24
Merged list: 1 2 3 5 8 10 12 15 20 24
Enter a command: m
Enter the length of the new list: 4
Enter the numbers: 10 7 10 20
list 1: 1 3 5 8 10 12 20
list 2: 7 10 20
Merged list: 1 3 5 7 8 10 12 20
```

```
//8. Test INTERSECTION(t)
//8a. Intersection of two lists
Enter a command: t
Enter the length of the new list: 4
Enter the numbers 10 5 2 7
list 1: 1 3 5 8 10 12 20
list 2: 2 5 7 10
Intersection of lists: 5 10
Enter a command: t
Enter the length of the new list: 3
Enter the numbers 20 8 5
list 1: 1 3 5 8 10 12 20
list 2: 5 8 20
Intersection of lists: 5 8 20


//8b. Intersection of two lists with no common elements
Enter a command: t
Enter the length of the new list: 3
Enter the numbers 14 7 34
list 1: 1 3 5 8 10 12 20
list 2: 7 14 34
Intersection of lists:


//9. Test invalid command
Enter a command: v
Invalid command try again: 4
Invalid command try again: p
The list is: 1 3 5 8 10 12 20
Enter a command: w
Invalid command try again: l
The length of the list is 7


//10. Test QUIT(q)
Enter a command: q
Exiting the program...
```

Grading Rubric

Insert item	12
Delete item	12
Search an item	10
Merge (along with complexity discussion)	15
Delete alternate	15
Intersection (along with complexity discussion)	15
Length	5
Printing the list	5
Exception handling	6
The `main` method	5
Total	**100**

5 Nonfunctional Requirements

5.1 Project Structure

Your implementation should match the provided package structure and visibility of each class and method within reason. Use your best judgment when writing your own classes/methods not explicitly listed in this document. (10–100 points)

5.2 Project Dependencies

Your project must exclusively use your `NodeType` class to build the linked list. You may not use arrays or any Java collections. (100 points)

5.3 Compiling and Running

Code that fails to compile or code that compiles but fails to run will receive a grade of zero. Your program must accept a single command-line argument as a path to the input file.

5.4 README

Commands to run and compile the program should be documented clearly in the README file.

You must include your full name and university email address in your README 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. (5 points)

5.5 Style Requirements

All submitted source files must conform to the CSCI 1302 style guide. Please see the guide on eLC for setting up style checks. (5 points for each style violation, up to 20)

A lack of appropriate inline comments may also constitute improper documentation.

5.6 Unit Tests

You are *encouraged* to write unit tests for your project. In CSCI 1302, you may have done this using custom classes, variables, and methods. Here, we recommend using a framework like JUnit to test your linked list features. Like integrating Checkstyle with your build process, testing frameworks provide another convenient way to speed-up your development process.

A sample test file is included on eLC. Although the package for `SortedLinkListTest` is the same as `SortedLinkList`, note that the test file is under `src/test/java/cs2720/p1`, while the main file is under `src/main/java/cs2720/p1`.

Feel free to check out the JUnit 5 documentation for more details.

You should also verify your JUnit and Surefire dependencies are compatible.

```
<dependencies>
  ...
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.13.4</version>
    <scope>test</scope>
  </dependency>
  ...
</dependencies>
...
<build>
  <pluginManagement>
    <plugins>
      ...
      <plugin>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>3.5.3</version>
      </plugin>
      ...
    </plugins>
  ...
  </pluginManagement>
</build>
```

Here, the "..." represent the rest of the POM that is not included here; these dots should not literally be included.

5.6.1 Note on Checkstyle

Unit tests are meant to be descriptive by design, particularly since they are often only a few lines of code. Because of this, you are not explicitly required to Javadoc methods that are given the `@Test` annotation (aka the actual unit tests that are run). Bear in mind that you are still expected to document any longer tests whose purpose is not immediately clear.

5.7 Late Submission Policy

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.
- Assignment extensions will not be granted, regardless of the situation. If you need extra time, submit up to two days late and consider the late penalty waiver option discussed in the syllabus.

5.8 Submission Notes

Submit the following files on Odin:

- ItemType.java
- SortedLinkedList.java
- NodeType.java
- LinkedListDriver.java
- README.txt

Odin Login Instructions

Projects should be submitted through the School of Computing's Odin server. You can ssh into Odin using your myID credentials. For example,

```
$ ssh abc12345@odin.cs.uga.edu
```

Group Submissions:

****Note: This 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 follows:

```
LastName1_LastName2_assignment1
```

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

```
$ 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 follows:

```
LastName_assignment1
```

Submission command should be used as follows:

```
$ 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 submissions, the folder name and README file are 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
```

This should include something 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 42634	TR 11:10 AM	`csci-2720f`