**Lab 05 Specification** – A practical exercise to practice more on Linked List.
50 points

# Lab Goals

- Do multiple exercises on Linked List.

# Suggestions for Success

- Take a look at the suggestions for successfully completing the lab assignment, which is available at: `https://www.cs.allegheny.edu/sites/amohan/resources/suggestions.pdf`

# Learning Assignment

If you have not done so already, please read all of the relevant "GitHub Guides", available at the following website:
`https://guides.github.com/`
that explains how to use many of the features that GitHub provides. This reading will help you to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, you should also read

- **GT ch3: 3.1 & 3.2**

# Assignment Details

Now that we have completed discussing important concepts related to Linked List in the last few lectures, it is your turn. In this lab, you will practice a variety of programs to retain the knowledge of the linked list data structure and how to integrate such a Linked List data structure into a Java program. This includes modifying one or more code files to implement a series of functionalities.

This lab is a two-week lab and should be done remotely. All students are expected to work on this lab asynchronously for the most part. It will be highly recommended that every student remain online during the entire lab session in order to maximize learning. Students who are unable to join the lab due to a different time zone should promptly reach out to the Professor and an appropriate arrangement will be done individually.

Please make sure to push your code file(s) and commit by the deadline, April 8th, 2020, morning 8:00 AM.

It is required for all students to follow the honor code. Some important points from the class honor code are outlined below for your reference:

1. Students are not allowed to share code files and/or other implementation details. It is acceptable to have a healthy discussion with your peers. However, this discussion should be limited to sharing ideas only.
2. Submitting a copy of the other's program(s) is strictly not allowed. Please note that all work done during lab sessions will be an opportunity for students to learn, practice, and master the materials taught in this course. By doing the work individually, students maximize the learning and increase the chances to do well in other assessments such as skill test, exams, etc $\cdots$

At any duration during and/or after the remote lab session, students are recommended to team up with the Professor and the TL's to clarify if there is any confusion related to the lab and/or class materials.

# Linked List Expense Tracker (50 points)

- **ADD** the following statement to all your code files including the file named `Node.java`, `Expenses.java`, and `ExpenseStub.java`.

  **This work is mine unless otherwise cited - Student Name**

- The starter code is provided inside the lab repository in a file named, `Node.java`, `Expenses.java`, and `ExpenseStub.java`. The starter code has detailed implementation already in place for `Node.java` file. It is only required to fill in the code inside the `Expenses.java` and `ExpenseStub.java` class files.

- The detailed outline of the requirements are outlined below:

  1. **addExpenses** - Implement this method in `Expenses.java` class file to add new expenses. The new expenses should be added towards the end of the list. Recall, there are many options to add an item to the linked list. We looked at multiple examples to add an item to the front of the list, end of the list, and middle of the list. In this part, we are required to add the item to the end of the list.
  2. **deleteExpense** - Implement this method in `Expenses.java` class file to delete an expense. The input for this method is a dollar amount. Iterate through the list to see if an expense item matches the dollar amount, if so then delete that item. We already looked at how to perform delete in-class examples. Refer back to your earlier code and your notes on how to implement this part.
  3. **countExpenses** - Implement this method in `Expenses.java` class file to count the total number of expenses made that matches a given dollar amount. The input for this method is a dollar amount. Iterate through the list to see if an expense item matches the dollar amount, if so then increment and maintain a counter to track the count of the expenses.
  4. **displayLatestExpenses** - Implement this method in `Expenses.java` class file to display all the expenses latest to oldest format (reverse order). Iterate through the list in the reverse direction to display the expenses from latest to oldest. Please keep in mind that the latest expense is always added to the end of the list.
  5. **displayOldestExpenses** - Implement this method in `Expenses.java` class file to display all the expenses oldest to latest format (forward order). Iterate through the list in the forward direction to display the expenses from oldest to latest. Please keep in mind that the oldest expense is situated in the front of the list.
  6. **findLeastExpenseDay** - Implement this method in `Expenses.java` class file to display the day in which the lowest expenses was made. Iterate through the list and find the minimum expense and the day in which that expense is made should be returned as the output of this method.
  7. **findLargestExpenseDay** - Implement this method in `Expenses.java` class file to display the day in which the largest expenses was made. Iterate through the list and find the maximum expense and the day in which that expense is made should be returned as the output of this method.
  8. **findLeastExpense** - Implement this method in `Expenses.java` class file to display the lowest expense from all the expenses done. Iterate through the list and find the minimum expense and return the minimum expense amount as the output of this method.
  9. **findLargestExpense** - Implement this method in `Expenses.java` class file to display the largest expense from all the expenses done. Iterate through the list and find the maximum expense and return the maximum expense amount as the output of this method.
  10. **findTotalExpenses** - Implement this method in `Expenses.java` class file to display the total expenses. Iterate through the list and count all the expenses and return the total expense amount as the output of this method.

11. **main method** - Add the required logic in the main method of `ExpenseStub.java` class file. Based on the option provided, the individual methods from the`Expenses.java` class file should be invoked. For example, if the option provided is 1, then `addExpense` in `Expenses.java` class file should be invoked. One way to do this is by using a series of if-else statements. Another option is to construct and use a switch case statement. There is a lot of freedom for students to design and develop this part using their own creativity.

## Submission Details

For this assignment, please submit the following to your GitHub repository before the **deadline** by using the link shared to you by the Professor:

1. Commented source code from the “Node”, “Expenses”, and “ExpenseStub” program.
2. It is highly important, for you to meet the honor code standards provided by the college. The honor code policy can be accessed through the course syllabus. Make sure to add the statement ”This work is mine, unless otherwise cited.” in all your deliverables such as source code and PDF files.

## Grading Rubric

1. There will be full points awarded for the lab if all the requirements in the lab specification are correctly implemented. Partial credits will be awarded if deemed appropriate.
2. Failure to upload the lab assignment code to your git repo will lead you to receive no points given for the lab submission. In this case, there is no solid base to grade the work.
3. There will be no partial credit awarded if your code doesn’t compile correctly. It is highly recommended to validate if the correct version of the code is being submitted before the due date and make sure to follow the honor code policy described in the syllabus. If it is a late submission, then it is the student’s responsibility to let the professor know about it after the final submission in GitHub. In this way, an updated version of the student’s submission will be used for grading. If the student did not communicate about the late submission, then automatically, the most updated version before the submission deadline will be used for grading purposes. If the student had not submitted any code, then, in this case, there are no points awarded to the student.
4. If you need any clarification on your lab grade, talk to the Professor. The lab grade may be changed if deemed appropriate.