# PA2: Linking Nodes

Due: Friday, 10/11/2019 before 5:00PM

Total points = 50

## 1 Overview

In this assignment, we will make a custom object that can dynamically group records based on different fields of the data. You will also come to appreciate the considerations for space and time efficiency, keeping that goal in mind while developing your logic and by analyzing your code at the end. This assignment will also develop your understanding of memory organization by providing the hands-on experience of managing a structure composed of linked nodes (that are not embedded within an array).

## 2 Objectives

In this assignment you will:

- Write a Scala custom class.
- Write the code for a basic iterator to access the data stored in your custom class.
- (Externally) analyze your code that you write.

Each `TaxEntry` object contains a number of values associated with each header. We will leverage this to implement a `"GROUPBY"` operation based on the values under these headers.

## 3 Useful Resources

Review the lecture notes and provided example code for some insight into the Scala syntax. You will also want to read the Scala references provided below:

- ArrayBuffer: `https://www.scala-lang.org/api/2.13.0/scala/collection/mutable/ArrayBuffer.html`
- Map: `https://www.scala-lang.org/api/2.13.0/scala/collection/mutable/Map.html`
- MapOps: `https://www.scala-lang.org/api/2.13.0/scala/collection/mutable/MapOps.html`

Relevant textbook sections:

- Nature of Arrays/Nature of Lists – §12.2,12.3 p406-408
- Mutable Singly Linked List – §12.4 p408-417
- Mutable Doubly Linked List – §12.5 p417-422

## 4 Instructions

Complete the following programming assignment and submit your answers to Autolab via the "PA2 Programming" assignment. You will only be submitting your `GroupByStore.scala` file.

**Expect this section to take 10-12 hours of setting up your environment, reading through documentation and base code, and planning, coding, and testing your solution.**

**Problem 1.** (50 points)
Your task is to implement a new container that is a cross between a linked list and an array. This new container will be used to store sequences of TaxEntry objects in a way that makes certain queries easy to answer. In order to provide access to the sequence of data stored within, you will also create two iterator methods to provide container access. The data organization will be done as follows:

- `TaxEntry` objects will be stored among many doubly-linked lists, where all entries that share the same value under the grouping attribute (column in the data set) are grouped in the same doubly-linked list.
  - The default attribute to group by is the street (i.e., `"STREET"`).
  - Any `String` in `TaxEntry.HEADERS` is valid for grouping.
    - E.g., `"PRINT KEY"`, `"FRONT"`, `"DEPTH"`, etc.
  - All `TaxEntry` objects matching on the grouping attribute should be kept within the same list.
    - E.g., when grouping by `STREET`, all properties on `MAIN` should be stored under one list.
    - New entries should be added to the head of your grouping lists.
    - The lists should be kept in sorted order based on the value corresponding to the grouping attribute.
    - E.g., when grouping by `STREET`, you would have lists corresponding to the different streets ordered alphabetically. For instance, if you only had `TaxEntry` objects on the streets `HERTEL`, `MAIN`, and `UNIVERSITY`, then the list of all entries on `HERTEL` should be linked in the list under index 0, all entries on `MAIN` should be linked in the list under index 1, and all entries on `UNIVERSITY` should be linked in the list under index 2 within `groupings`.
  - Grouping lists should be maintained within `groupings` (an `ArrayBuffer`).
    - The head of each doubly-linked list should be stored in `groupings`.
  - No list should be empty (i.e., empty `GroupByStore` object should contain 0 lists).
- The `TaxEntry` objects should be stored within the `DNode[TaxNode]` nodes.
  - You are required to build/maintain the `prev` and `next` references for the nodes.
- Upon regrouping, all nodes within a list should match on the new grouping attribute (column).
  - The ordering of the nodes within a grouping list is not important.
  - There should only be one list per unique value under the grouping attribute.

### 4.1 Insertion Data Organization

Insertion of elements should maintain storage so that the `TaxEntry` is added to the head of the list corresponding to the elements that match on the current matching attribute. If no such list exists that matches the new entry under the grouping attribute, add a new list to `groupings` and link in the new element.

### 4.2 How to Regroup

When `regroup` is called, you need to consider the following:

- If the regrouping attribute given is the same as the current grouping attribute, do nothing.
- Otherwise, you must go through each list and reorganize the nodes into new lists so that every `TaxEntry` under a given list matches on the new matching attribute.
  - You may want to create a temporary `ArrayBuffer` to hold the new list heads until you are finished redistributing the nodes.
  - The order that the `TaxEntry` objects they appear within a list after regrouping is unimportant.

### 4.3 GroupByStore Methods Overview

The following method definitions are left for you to complete:

- `insert(taxEntry: TaxEntry): Unit`
  - insert the `TaxEntry` into your storage under the appropriate group.
- `regroup(attribute: String): Unit`
  - regroup the `TaxEntry` objects based on the provided `attribute`.
- `iterator(): Iterator[TaxEntry]`
  - construct an iterator that traverses through all nodes within the sequence.
- `iterator(groupValue: String): Iterator[TaxEntry]`
  - construct an iterator that traverses through all nodes of the group that matches the value provided.
  - if no group matches the `value` under the given grouping, the iterator should simply be constructed so that `hasNext` is false from the start.
  - e.g., if you are grouped by `STREET` and call `.iterator("MAIN")` you should iterate over all properties on `MAIN`. However, if you call `.iterator("AAAA")` you would receive an iterator where `hasNext` is false from the start because there is no street AAAA.

### 4.4 Assumptions/Notes

- After calling `iterator()` or `iterator(String)`, no container modification will occur.
  - If the container is modified, new iterators will be constructed and old iterators will be discarded.
  - You don't have to worry about whether or not the iterator is still valid.
- We will not perform any copying of your container, but we will create multiple separate containers.
- You should make your operations as efficient as possible.
  - There is no specific runtime required to complete the assignment.
  - There is not much flexibility in the implementation, but you should be mindful of what can be done efficiently.
- If your program is too slow, it may fail to pass tests due to time constraints.

## 4.5 Suggested Approach

1. Read the entire PA2-instructions.pdf document. **Do not try to write any code until you have done this**.
2. Plan out how the container works.
   - Try drawing out the contents of the container and its attributes in different situations and then work on your approach.
   - E.g., how will your container look when freshly initialized? after a few insertions?
3. Download the PA2 skeleton files (`PA2-handout.zip`), extract the files, and open the folder `PA2` into IntelliJ.
   - Be sure to configure the JDK/Scala settings if prompted.
   - Reimport the Maven project to ensure dependencies are correctly setup.
4. Update the copyright statements with your UBIT and person number in your submission files.
   - If you fail to do so, your submission will not be graded.
5. Copy the .csv data files over from PA1.
6. Read over all code among the provided files.
7. After some planning, work on the setup of the initial storage and begin work on insert.
   - Make sure you test that you correctly insert (open up your internals in the debugger).
8. Now work on the iterator functionality. Check that you correctly traverse with your iterator object in different scenarios (tests are provided to utilize this).
9. Lastly, work on regroup. Make sure that you get the correct groupings afterward.
   - Feel free to extract rows from the large dataset and curate a scenario with only a few groups.

It is particularly important to follow some semblance of this approach when working on this assignment, as it will be confusing to work out of this order. For instance, it doesn't make sense to try to work on the iterator when you don't have insert working, etc.

## 4.6 Allowed Header/Library Usage

- You may not use any additional headers beyond what is already included in the handout code.
- You may use `scala.util.controls.Breaks._`
- You may not use any other containers aside from the `ArrayBuffer` holding the heads of the lists and the `DNode`.

## 5 Submission

For each section (PA2 Programming), you will be allowed 5 submissions to each, without penalty. Starting from the 6th submission, you will receive

- a 5 points per submission deduction from your score on the respective assignment.

Note: your score is what you receive on your latest submission. If you receive a score and then resubmit, even if you receive a lower score/0 points, that will be your score for the assignment.

Also note: if you submit early/late, the bonus/penalty would be awarded against the entirety of the assignment, not just the single part you submitted early/late. The timestamp for your submission is that of the latest part submitted.

### Creating Your Submission

Your submission should be a single file: `GroupByStore.scala`. No other files should be provided for grading.

## 6 Late Policy

The policy for late submissions on assignments is as follows:

- More than 5 days before the deadline: +5 points + 100% of what you earn on your best submission.
- Up to 5 days before the deadline: +1 point per day + 100% of what you earn on your best submission.
- On the deadline: 100% of what you earn on your best submission.
- One day late: 25 point deduction (lower total points by 25).
- Two days late: 50 point deduction (lower total points by 50).
- $>$ 2 days late: 0 points.

A day is considered as a 24 hour period counting from the deadline time. From this policy, you will see that all assignments must be submitted within two days past the assigned deadline. The date of submission is that of your latest submission made, even if this is not your highest scoring submission. For assignments with multiple component submissions, only 1 penalty will be assessed based on the file submitted last. If a staggered deadline is given (e.g., two components due one week apart), the earlier deadline will be a hard deadline and no late submissions will be accepted for the first component.

You will have the ability to use three grace days throughout the semester, and at most two per assignment (since assignments are not be accepted beyond two days late). Using a grace day will negate the 25 point penalty for one day of late submission, but will not allow you to submit more than two days late. Please plan accordingly. You will not be able to recover a grace day if you decide to work late and your score was not sufficiently higher. **Grace days are automatically applied** to the first instances of late submissions, **and are non-refundable**. For example, if an assignment is due on a Friday and you make a submission on Saturday, you will automatically use a grace day, regardless of whether you perform better or not. **Be sure to test your code before submitting**, especially with late submissions, **in order to avoid wasting grace days**.

**Keep track of the time if you are working up until the deadline**. Submissions become late after the set deadline. Keep in mind that **submissions will close 48 hours after the original deadline** and you will no longer be able to submit your code after that time.

## 7 AI Policy Overview

As a gentle reminder, please re-read the academic integrity policy of the course. I will continue to remind you throughout the semester and hope to avoid any incidents.

### 7.1 What constitutes a violation of academic integrity?

These bullets should be obvious things not to do (but commonly occur):

- Turning in your friend's code/write-up (obvious).
- Turning in solutions you found on Google with all the variable names changed (should be obvious). This is a copyright violation, in addition to an AI violation.
- Turning in solutions you found on Google with all the variable names changed and 2 lines added (should be obvious). This is also a copyright violation.
- Paying someone to do your work. You may as well not submit the work since you will fail the exams and the course.
- Posting to forums asking someone to solve the problem.

  **Note:** Aggregating every [stack overflow answer|result from google|other source] because you "understand it" will likely result in full credit on assignments (if you aren't caught) and then failure on every exam. Exams don't test if you know how to use Google, but rather test your understanding (i.e., can you understand the problems to arrive at a solution on your own). Also, other students are likely doing the same thing and then you will be wondering why another person that you don't know has your solution.

You should know that seeking solutions to the assignment does not fall under solving the problem yourself. Things that may not be as obvious:

- Working with a tutor who solves the assignment with you. If you have a tutor, please contact me so that I may discuss with them what help is allowed.
- Sending your code to a friend to help them. **If another student uses/submits your code, you are also liable and will receive the same punishment**.
- Joining a chatroom for the course where someone posts their code once they finish, with the honor code that everyone needs to change it in order to use it.
- Reading your friend's code the night before it is due because you just need one more line to get everything working. It will most likely influence you directly or subconsciously to solve the problem identically, and your friend will also end up in trouble.

**The assignments should be solved individually with only assistance from course staff and allowed resources**. You may discuss and help one another with technical issues, such as how to get your compiler running, etc. It is not acceptable that you both worked together and have nearly identical code. If that is going to be a problem for you, don't solve the problems in that close of proximity.

## 7.2 What collaboration is allowed?

There is a gray area when it comes to discussing the problems with your peers and **I do encourage you to work with one another to solve problems**. That is the best way to learn and overcome obstacles. At the same time you need to be sure you do not overstep and not plagiarize. Talking out how you eventually reached the solution from a high level is okay:

> I used a stack to store the data and then looked for the value to return.

but explaining every step in detail/pseudocode is not okay:

> I copied the file tutorial into my code at the start of the function, then created a stack and pushed all of the data onto the stack, and finished by popping the elements until the value is found and use a return statement.

The first example is OK but the second is basically a summary of your code and is not acceptable, and remember that you shouldn't be showing any code at all for how to do any of it. Regardless of where you are working, you must always follow this rule: **Never come away from discussions with your peers with any written work, either typed or photographed, and especially do not share or allow viewing of your written code.**

If you have concerns that you may have overstepped or worked closely with someone, please address me prior to deadlines for the assignment. We will address options at that point.

## 7.3 What resources are allowed?

With all of this said, please feel free to use any [files|examples|tutorials] that we provide directly in your code (with proper attribution). Feel free to directly use anything from lecture or recitations. You will never be penalized for doing so, but must always provide attribution/citation for where you retrieved code from. Just remember, if you are citing an algorithm that is not provided by us, then you are probably overstepping.

More explicitly, you may use any of the following resources (with proper citation/attribution in your code):

- Any example files posted on the course webpage (from lecture or recitation).
- Any code that the instructor provides.
- Any code that the TAs provide.
- Any code from the tour of Scala (https://docs.scala-lang.org/tour/tour-of-scala.html)
- Any code from Scala Collections (https://docs.scala-lang.org/overviews/collections-2.13/introduction.html)
- Any code from Scala API (https://www.scala-lang.org/api/2.13.0/)
- Additional references may be provided as the semester progresses, but only those provided publicly by course staff are allowed to be utilized. These will be listed on Piazza under Resources.

**Omitting citation/attribution will result in an AI violation** (and lawsuits later in life at your job). This is true even if you are using resources provided.

Lastly, **if you think you are going to violate/have violated this policy, please come talk to me ASAP so we can figure out how to get you on track to succeed in the course**. This policy on assignments is here so that you learn the material and how to think yourself. There is no benefit to submitting solutions (which likely exist in some form).