COMP 2402 Winter 2022
Assignment 2

Due: Thursday February 10 2:00pm
Topic focus: Lec 5 - 8
lates accepted within 24 hours
submit early and often

Academic Integrity

You may:

- Discuss general approaches with course staff,
- Discuss general approaches with your classmates,
- Use code and/or ideas from the textbook,
- Use a search engine / the internet to look up basic Java syntax,
- Send your code / screen share your code with course staff (although it is unlikely the course staff has the bandwidth to look at individuals' code, unfortunately.)

You may **not**:

- Send or otherwise share code or code snippets with classmates,
- Use code not written by you, unless it is code from the textbook (and you should cite it in comments),
- Use a search engine / the internet to look up approaches to the assignment,
- Use code from previous iterations of the course, unless it was solely written by you,
- Use the internet to find source code or videos that give solutions to the assignment.

If you ever have any questions about what is or is not allowable regarding academic integrity, **please do not hesitate to reach out to course staff**. We will be happy to answer. Sometimes it is difficult to determine the exact line, but if you cross it the punishment is severe and out of our hands. **Any student caught violating academic integrity, whether intentionally or not, will be reported to the Associate Dean** and be penalized as follows:

- First offence: F in the course.
- Second offence: One-year suspension from program.
- Third offence: Expulsion from the University.

These are standard penalties. More-severe penalties will be applied in cases of egregious offences. For more information, please see Carleton University's Academic Integrity Policy.

Coding Environment Setup

If you want suggestions on how to set up your coding environment, please watch this video or read the "Setting Up Your Programming Environment" post on piazza.

If you want Alexa's 20-minute "Assignment 1 setup" video, please go here. The setup for assignment 1 is almost identical to that of assignment 2 so there will not be a new video.

How to Get Help

Problem Solving & Assignment Tips Document

- We made this document just for you, full of tips to help you complete your programming assignments faster and better! We will add to it as the course progresses, and you are welcome to add your suggestions as well.

Piazza

- Good for getting a quick response, since it goes to a wider audience than email.
- Good for viewing questions that other students have. Chances are, if you have the question, so does someone else. If tagged correctly (e.g. "a2" for assignment 2), you can easily see all questions relating to a particular subject.
- Good for general questions that don't divulge your approach (e.g. "Am I allowed to use standard JCF data structures in my solution?" "Does anyone else get this strange "file not found" error when submitting?" etc.)
- Good for asking specific-to-you questions of the course staff, which you can ask "privately" on piazza. This gets you the benefits of a wider audience yet keeps your personal details private to course staff. (You can and should also contact Alexa this way.)

Student Hours

- Good for quick questions that may have some back and forth.
- Good for clarifications.
- Good for "tips and tricks".
- Not good for debugging sessions. The person holding the student hour may have many people waiting, and as such cannot spend 20 minutes helping you debug, unfortunately. They may give you a push in the right direction, then ask you to go back in line while you work with that push, while they help someone else.

Discord

- Good for light social interactions and commiseration.

- Please remember that this discord is an official class discord, and as such you should keep your behaviour “professional”. Please be respectful. Jokes are great, just not at the expense of an individual or a specific group. Disrespectful behaviour (intentional or not) will be met with our zero-tolerance policy (removal from server.)
- This is not a good place for asking questions, as the course staff will be spending most of their time monitoring piazza. Piazza has a better system for tracking open questions and answers, and so it is more time efficient for the course staff to spend their time there instead of discord.

Grading

This assignment will be tested and graded by a computer program (and **you can submit as many times as you like, your highest grade is recorded**). For this to work, there are some important rules you must follow:

- Keep the directory structure of the provided zip file. **If you find a file in the subdirectory comp2402w22a2 leave it there.**
- Keep the package structure of the provided zip file. **If you find a package comp2402w22a2; directive at the top of a file, leave it there.**
- Do not rename or change the visibility of any methods already present. If a method or class is public, leave it that way.
- Do not change the `main(String[])` method of any `PartX` class. These are setup to read command line arguments and/or open input and output files. Don't change this behaviour. Instead, learn how to use command-line arguments to do your own testing.
- **Submit early and reasonably often.** The submission server compiles and runs your code, and gives you a mark. You can submit once every 5 minutes and we keep track of your highest overall score. There is no excuse for submitting code that does not compile or does not pass tests. The 5 minute wait limit is to prevent you from using the server to debug, potentially overloading it. (You should debug locally! More on that later.)
- Write efficient code. The submission server places a limit on how much time it will spend executing your code, even on inputs with a million lines. For some questions it also places a limit on how much memory your code can use. If you choose and use your data structures correctly, your code should easily execute within the time limit. Choose the wrong data structure, or use it the wrong way, and your code will be too slow or use too much space for the submission server to work (resulting in a failure of that test).

Submitting and Server Tests

The submission server is here. If this is your first time submitting, you will need to get a secret key which you will then use to upload your assignment to the submission server. If you have issues, please post to piazza to the Instructors (or the class) and we’ll see if we can help.

When you submit your code, the server runs tests on your code. **These are bigger and better tests than the small number of tests provided in the "Local Tests" section further down this document**. They are obfuscated from you, because **you should try to find exhaustive tests of your own code**. This can be frustrating because you are still learning to think critically about your own code, to figure out where its weaknesses are. But have patience with yourself and make sure you read the question carefully to understand its edge cases.

Warning: Do not wait until the last minute to submit your assignment. There is a hard 5 second limit on the time each test has to complete, and 5 minutes between submissions. If the server is heavily loaded, borderline tests may start to fail. **You can submit multiple times and your best score is recorded** so there is no downside to submitting early.

The Assignment

Purpose & Goals

Generally, the assignments in this course are meant to give you the opportunity to practice with the topics of this course in a way that is challenging yet also manageable. At times you may struggle and at others it may seem more straight-forward; just remember to keep trying and practicing, and over time you will improve.

Specifically, assignment 2 aims to improve your understanding of the ArrayStack, LinkedList, and Skiplist interfaces by:

- Using the ArrayList, LinkedList, and ConcurrentSkiplistSet interfaces to solve a variety of problems (Part A),
- Implementing some new methods in the IntDLList, DLList, and SkiplistList classes (Part B), and
- Reflecting on your choice of data structures and algorithms (Part C).

Setup

(Alexa made a 20-minute Assignment 1 Setup video that may help you do the following steps; it is very similar to the setup for Assignment 1.)

Start by downloading and decompressing the Assignment 2 Zip File (comp2402w22a2.zip on piazza's Resources tab), which contains a skeleton of the code you need to write. You should have the files: Part0.java, Part1.java, Part2.java, Part3.java, Part4.java, DLList.java, IntDLList.java, SkiplistList.java, and MyList.java.

The skeleton code in the zip file compiles fine. Here's what it looks like when you unzip, compile, and run `Part0` and `Part1` from the command line:

```
a2 % pwd
/Users/asharp/2402/a2
a2 % ls
comp2402w22a2.zip
a2 % unzip comp2402w22a2.zip
Archive:  comp2402w22a2.zip
   creating: comp2402w22a2/
  inflating: comp2402w22a2/DLList.java
  inflating: comp2402w22a2/IntDLList.java
  inflating: comp2402w22a2/Part1.java
  inflating: comp2402w22a2/Part0.java
  inflating: comp2402w22a2/Part3.java
  inflating: comp2402w22a2/MyList.java
  inflating: comp2402w22a2/Part2.java
  inflating: comp2402w22a2/Part4.java
  inflating: comp2402w22a2/SkiplistList.java
a2 % ls
comp2402w22a2        comp2402w22a2.zip
a2 % javac comp2402w22a2/*.java
a2 % ls
comp2402w22a2        comp2402w22a2.zip
a2 % ls comp2402w22a2
DLList.class         Part0.class     Part3.class
DLLIst.java          Part0.java      Part3.java
IntDLList.class      Part1.class     Part4.class
IntDLList.java       Part1.java      Part4.java
MyList.class         Part2.class     SkiplistList.class
MyList.java          Part2.java      SkiplistList.java
a2 % java comp2402w22a2.Part0
3
50
4
13 [I hit Ctrl-D to end the input here]
3D [The D here is a remnant of the Ctrl-D]
50
4
13
Execution time: 5.986822196
a2 % java comp2402w22a2.Part1
Execution time: 2.2645E-5
```

You can also run the `DLStack` and `IntDLList` programs:

```
a2 % java comp2402w22a2.IntDLList
<... output omitted >
a2 % java comp2402w22a2.DLList
<... output omitted >
```

If you are having trouble running these programs, **figure this out first before attempting to do the assignment**. Check the assignment 2 FAQ on piazza and/or the aforementioned setup video, or ask a question there if you are stuck and course staff or another student will likely help you fairly quickly.

Part A: The Interface

The file `Part0.java` in the zip file actually does something. You can use its `doIt()` method as a starting point for your solutions. Compile it. Run it. Test out the various ways of inputting and outputting data. **You should not need to modify Part0.java, it is a template. Do not modify the main(String[]) methods for any PartX.java file, as they are expected to remain the same for the server tests.**

You can download some sample input and output files for each question as a zip file (a2-io.zip on piazza's Resources tab). If you find that a particular question is unclear, you can probably clarify it by looking at the sample files for that question. Part of problem solving is figuring out the exact problem you are trying to solve.

You may assume that all lines are integers under 32 bits. If you need some help with modular arithmetic, try this khan academy page. Unlike assignment 1, let's just use java's floorMod(x,d) to compute x mod d. This always returns a non-negative integer, which will simplify our lives.

1. [8 marks] Given a file where each line is a single (32-bit) integer, read in all lines of the file one at a time; for each even-indexed line `L`, find the line prior to it that is closest to but `<= L` (if such a line exists) and add it to a sum (modulo 2402) that you will output. For example, if the input is

2400	← line 0, an even-indexed line, no lines previous, contributes 0.
99	
100	← line 2, even-indexed. line 1's 99 <= 100, contributes 99.
27	
100	← line 4, even-indexed. line 2's 100 <= 100, contributes 100.
30	
10	← line 6, even-indexed. no lines previous <= 10, contributes 0.

   Then the output is (99+100)%2402=199. If line 4 had been 99, the output would be (99+99)=198. If line 4 had been 25 then the output would be (99+0)=99.

Hint: You can and should look at the Java documentation of the data structure you choose to use to see if there are any useful methods in it that you can use. Relevant documentation is linked to from the Purpose & Goals section of this assignment.

Note: Please use `Math.floorMod` instead of `%` to make everyone's lives a bit easier. See the note just before Part 1's description.

2. [8 marks] (Assignment 1.4 Redux) Given a file where each line is a single (32-bit) integer, read in all lines of the file one at a time; output the sum **modulo 2402** of the 24022022 values obtained as follows: take every `x`th value from values from the file in sorted order (where `x>0` is a parameter to the `doIt` method) starting with the first, wrapping around to process these elements again if necessary, until you have the sum modulo 2402 of the 24022022 elements. For example, if `x`=2 and the input is

   4
   7
   5
   7
   3
   6
   8

   Then the output is (3+5+7+3+5+7+...) modulo 2402, where we are summing 3, 5, 7 because the *values* in sorted order are 3, 4, 5, 6, 7, 8; every other element is 3, 5, 7, then we wrap around and take the same elements again, and so on. If `x`=3 with this same example, we would sum (3+6) 24022022/2 times, modulo 2402. If there are not enough elements to sum you should return 0.

   Note: Please use `Math.floorMod` instead of `%` to make everyone's lives a bit easier. See the note just before Part 1's description.

3. [10 marks] Given a file where each line is a single (32-bit) integer, read in all lines of the file one at a time; for each **distinct** value `V`, select the smallest value `W` in the file that is > `2*V` and output the sum modulo 2402 of all selected lines. For example, if the input is

   4 ← contributes 9 to sum since 9 > 2*4
   2 ← contributes 8 to sum since 8 > 2*2 and there is no other value > 2*2 but < 8
   8 ← contributes nothing to sum since there is no value > 2*8 = 16
   9 ← contributes nothing to sum since there is no value > 2*9 = 18
   3 ← contributes 8 to sum since 8 > 2*3 and there is no other value >2*3 but < 8
   2 ← contributes nothing to sum since we've already dealt with a 2
   10 ← contributes nothing to sum since there is no value > 2*10=20

   Then the output is 9+8+8=26 (mod 2402), which is 25.

Note: Please use `Math.floorMod` instead of `%` to make everyone's lives a bit easier. See the note just before Part 1's description.

4. [10 marks] Given a non-empty file where each line is a single (32-bit) integer, read in all lines of the file one at a time; consider the list constructed so that for every `i=0,1,2,...,x-1`, we insert the `x` elements `1,2,3,…,x` as a contiguous block at index `curr=list.get(i)%l.size()` of the list so far. (As usual, `x>0` is a parameter to the `doIt` method.) For example, if x=2 and the input is

   4
   20
   5
   9
   30
   8 .

   When i=0 we insert 1, 2 as a contiguous block at index 4%6=4 to get

   4
   20
   5
   9
   1
   2
   30
   8

   Then when i=1 we insert 1, 2 as a contiguous block at index 20%8=4 to get

   4
   20
   5
   9
   1
   2
   1
   2
   30
   8

   Once we have this final list we return the sum modulo 2402 of `(line i)·(i+1)` for each index `i=0,1,...,l.size()`. That is, on this example we return

   $(4{\cdot}1 + 20{\cdot}2 + 5{\cdot}3+9{\cdot}4+1{\cdot}5+2{\cdot}6+1{\cdot}7+2{\cdot}8+30{\cdot}9+8{\cdot}10)$ % 2402 = 485

   Tip: While this seems convoluted, just take it one step at a time. Focus on the separate "stages" of this problem; solve and debug one stage first before proceeding to the next. Get something that works even if it's "dumb" and then work on improving each part.

Hint: You might want to investigate ListIterators, which allow you to walk through a list one element at a time, and even add/remove from that list as you walk through it. All of the interfaces listed in the Purpose & Goals section have ListIterators you can read about and get from their documentation.

Note: If you downloaded the a2-io.zip file prior to Monday Jan 31 2:50pm you'll want to download it again to get the updated part4 testing files.

Note: Please use `Math.floorMod` instead of `%` to make everyone's lives a bit easier. See the note just before Part 1's description.

Part B: The Implementation

The (provided) `DLList`, `IntDLList`,, and `SkiplistList` classes are (incomplete) implementations of the `MyList` interface. Your task is to complete the implementation of the `DLList`, `IntDLList`, and `SkiplistList` classes, as specified below. Default implementations are provided for the 4 methods so that you can compile and run the main methods of the relevant classes right out of the gate; these defaults will not pass any of the tests, however.

5. [8 marks] (Assignment 1.6 Redux) In the `IntDLList` class, implement

   ```
   public void sum(IntDLList other)
   ```

   Replace the `i`th element of `this IntDLList` by `this[i] + other[i]`. That is, the elements from this are now each value-shifted by the corresponding entry in `other`. If `other` is shorter than `this`, just repeat `other` as many times as is necessary until you have shifted every element of `this` once.

   For example, if the list is originally `mal=[4,5,6,7]` and `other=[1,2,3]` then `mal.sum(other)` changes `mal` to `[5, 7, 9, 8]`.

6. [8 marks] (Assignment 1.7 Redux) In the `IntDLList` class, implement

   ```
   public void intervalInsert(IntDLList other)
   ```

   Modify this `IntDLList` by inserting `-1` into `this` at intervals specified by `other`. That is, there are `other[0]` elements of `this` before `-1` then the next `other[1]` elements of `this` before the next `-1`, etc. If `other's` intervals do not cover all of `this`, just repeat `other` as many times as is necessary until you have "processed" all of `this` once.

   For example, if the list is originally `ml=[4,5,6,7,8,9]` and `other=[1,2]` then `mal.intervalInsert(other)` changes `ml` to `[4,-1,5,6,-1,7,-1,8,9]`.

You may assume that all entries of `other` are positive.

7. [12 marks] In the `DLList` class, implement
   ```
   public void insertSingleBlock(int i, DLList<T> other)
   ```
   which inserts the `DLList other` at index `i` of `this DLList`.

   For example, if the list is originally `ml=[1,2,3,4,5,6,7]` and `other=[-1,-2]` then `ml.insertSingleBlock(1, other)` changes `ml` to `[1, -1, -2, 2, 3, 4, 5, 6, 7]`. You may assume that input `i` is a valid index (i.e., `0 <= i <= size()`.)

8. [10 marks] In the `SkiplistList` class, implement
   ```
   public String toString()
   ```
   to return a more useful String representation of the Skiplist than the current implementation. You will want to overwrite the existing code such that if you were to print this you would get information about the heights of the nodes and the lengths of the edges within them. This will really help you in debugging Part 9.

   Your string should represent each level of the skiplist on an individual line, where the sentinel is represented by the pipe |, each level 0 element is divided by the following by two dashes, then the length of that edge in parentheses, then two more dashes. More generally, a length `L` line is `2L-1` pair of dashes, then the length in parentheses, and then `2L-1` more pair of dashes.

   For example, a skiplist on 9 elements might be represented by the String
   ```
   |----------------------------------(9)----------------------------------i
   |------------------(5)------------------e--------------(4)--------------i
   |--(1)--a--(1)--b--(1)--c--(1)--d--(1)--e--(1)--f--(1)--g--(1)--h--(1)--i
   ```

   whereas a skiplist on 5 elements might be represented by the String
   ```
   |------------------(5)------------------e
   |--(1)--a--(1)--b--(1)--c--(1)--d--(1)--e
   ```

   and a skiplist on 7 elements might be represented by the String
   ```
   |------------------(5)------------------e
   |--(1)--a--(1)--b--(1)--c--(1)--d--(1)--e--(1)--f--(1)--g
   ```

   A different skiplist on the same 7 elements might look like this String
   ```
   |-------------------------(7)-------------------------6
   |-------------------------(7)-------------------------6--(1)-
   -7
   |-------------------------(7)-------------------------6--(1)-
   -7
   ```

```
|--(1)--0--(1)--1--(1)--2--(1)--3--(1)--4--(1)--5--(1)--6--(1)-
-7
```

The empty skiplist is just

```
|
```

Look at the existing `toString()` method to help you use the StringBuilder, then overwrite that code to have the new desired functionality.

9. [12 marks] In the `SkiplistList` class, implement

    ```
        public void insertSingleBlock(int i, SkiplistList<T>
    other)
    ```

    which inserts the `SkiplistList` `other` at index `i` of `this` `SkiplistList`.

    For example, if the list is originally `l=[1,2,3,4,5,6,7]` and `other=[-1,-2]` then `l.insertSingleBlock(1, other)` changes `l` to `[1, -1, -2, 2, 3, 4, 5, 6, 7]`. You may assume that input `i` is a valid index (i.e., `0 <= i <= size()`.)

Part C: The Debrief

10. [5 marks] Do the assignment 2 debrief (multiple-choice questions) on brightspace after the programming deadline has passed (and the solutions are available). The debrief questions are due 8 days after the assignment deadline (in this case, by Friday February 18th, 2:00pm.) No lates accepted.
    The debrief questions are meant to encourage retrospection on what you were meant to learn from the assignment. If you were stuck on a problem, this is an opportunity to look at the solutions and at the assignment 2 debrief, to figure out what went wrong for you, and to still learn what you were meant to learn.
    It is also an opportunity to provide feedback to Prof Alexa on certain aspects of the course so far.

Local Tests

For Parts 1, 2, 3, and 4, you can download some sample input and output files for each question as a zip file (a2-io.zip, on the Resources tab on piazza). Once you get a sense for how to test your code with these input files, write your own tests that test your program more thoroughly (**the provided tests are not exhaustive!**)

For Parts 5, 6, 7, 8 and 9 (`DLList`, `IntDLList` and `SkiplistList`), the main method of each file provides some tests you can and should add / modify as you go along.

Server Tests

See the “Submitting and Server Tests” section further up this document.

Tips, Tricks, and FAQs

For the most up-to-date Assignment-specific FAQ, please see the Assignment 2 FAQ post on piazza (and/or the a2 filter.)

Regarding tips and tricks, please see the Problem Solving & Assignment Tips document (which may be enhanced as the semester progresses, so check back frequently!)