- **You must submit your work electronically (via moodle) by 11:59 PM of the due date (see above)**.
- **Write your group's own answer and show all your work**.
- **Every assignment must be typed/generated into a pdf file (convert your file into a pdf format before submission).**
- **Weight: 5%**
- **On your assignment, specify the following at the beginning**.
  Assignment number: 2
  Course number: CSC 220 001
  Term: Winter, 2018
  Instructor name: Dr. Manki Min
  Student names: (names of all the members in your group)

Do the following:

1. Create a generic `Stack` class that supports the following operations (and only these) as defined in our lectures:

   - Push: insert an item on top of the stack
   - Pop: delete an item from the top of the stack (and return it)
   - Peek: return (but do not delete) the item on top of the stack
   - Size: return the size of the stack
   - IsEmpty: return if the stack is empty
   - IsFull: return if the stack is full
   - Equals: compare two stacks
   - Add: concatenate two stacks

   Also include a constructor, a copy constructor, and the output capability – i.e.:

   - `Stack()`
   - `Stack(Stack s)`
   - `String toString()`

   Implement Stack using the generic linked lists either of your own or the moodle-posted one. Use the same StackTest.java to test your Stack.java. Submit your Stack.java code both in the pdf file and as a file.

2. Create a generic `Queue` class that supports the following operations (and only these) as defined in our lectures:

   - Enqueue: insert an item at the end of the queue
   - Dequeue: delete an item from the beginning of the queue (and return it)
   - Peek: return (but do not delete) the item at the beginning of the queue
   - Size: return the size of the queue
   - IsEmpty: return if the queue is empty
   - IsFull: return if the queue is full
   - Equals: compare two queues
   - Add: concatenate two queues

   Also include a constructor, a copy constructor, and the output capability – i.e.:

   - `Queue()`
   - `Queue(Queue q)`
   - `String toString()`

Implement Queue using the generic linked lists either of your own or the moodle-posted one. Use the same QueueTest.java to test your Queue.java. Submit your Queue.java code both in the pdf file and as a file.

3. Write a program that converts infix expressions to their postfix equivalents and subsequently evaluates them. Receive input from `stdin` and send output to `stdout`. An example run of your program might be: `java InfixToPostfix < input.txt > output.txt`. The input contains a single infix expression per line as follows:

```
5+7
3*8+6
2^2+9
7^7-6^6
9-5*8/4^2
2^3^2-4^2+4
3*3+9*8-2
6-4-3+3+4*6/2+8*8
4-3*(4*(3*6-(8+9)^2)*(9-3)/3)-7
(3+(9-3)*(5-3)*4)
((((((9-9))))))
((9/2*3/2+1)*1)*1*1*1*1
9*3-(5-7/5)
```

There are several requirements:

- You must use your generic `Stack` and `Queue` classes to implement the infix to postfix conversion and the subsequent evaluation. This will, in turn, mean using your generic `List` and `Node` classes. To make this all work, make sure that `Stack.class`, `Queue.class`, `List.class`, and `Node.class` are in the same directory as your infix to postfix source (e.g., `InfixToPostfix.java`).
- Structure your output so that it displays the infix expression on a line, the postfix expression on the next line, the result on the next line, followed by a blank line. For example:

```
5 + 7
5 7 +
12.0

9 * 3 - ( 5 - 7 / 5 )
9 3 * 5 7 5 / - -
23.4
```

- Evaluations must be precise (i.e., no integer division). Think about what this means in terms of your evaluation stack.
- Note that `2^3^2 = 512`!

Submit your InfixToPostfix.java code both in the pdf file and as a file.

4. [Bonus problem] Implement Queue using two stacks. Use the same QueueTest.java to test your Queue.java. Submit your Queue.java code both in the pdf file and as a file. **(bonus 20 points)**

5. Describe the logic of Push(data) and Pop() for every possible case using vertical notation of stacks which are implemented with a linked list of nodes and a node variable 'top'. Show the change of stack contents as well as the links such as top, the new node's link, the remaining nodes' links, etc (using arrows).

6. Describe the logic of Enqueue(data) and Dequeue() for every possible case using horizontal notation of queues which are implemented with a linked list of nodes and two node variables 'head' and 'tail'. Show the change of queue contents as well as the links such as head, tail, the new node's link, the remaining nodes' links, etc (using arrows).

7. Convert the following infix expression into its postfix equivalent:
   1 - 2 + 3 * 4 - 6 / 3 + 5

8. Show how to evaluate your postfix expression for the previous problem using a stack.
