COMS 3137 Data Structures and Algorithms. Homework 2

Submit your electronic files via `http://courseworks.columbia.edu`. Do the written assignments electronically if possible, since it is easier for all parties involved. Use the provided makefile for the programming problems. Place your files into a directory named

`<your_uni>_homework2`

Archive your submission folder using the command:

```
tar -czvf uni1234_homework2.tgz uni1234_homework2
```

Upload your archive to the Class Files section of courseworks in the Homework 2 subdirectory. It is also recommended that you keep a pristine copy of your submission folder in case there is any submission error.

1 Written Problems

Either pseudocode or Java are acceptable when asked to provide algorithms.

1. (4 points) **Weiss 3.2**. Swap two adjacent elements by adjusting only the links (and not the data) using:
   (a) Singly linked lists.
   (b) Doubly linked lists.
2. (4 points) **Weiss 3.7**. What is the running time of the following code?

```
public static List<Integer> makeList( int N )
{
    ArrayList<Integer> lst = new ArrayList<Integer

    for( int i = 0; i < N; i++ )
    {
        lst.add( i );
        lst.trimToSize( );
    }
}
```

(Hint: you'll want to research what the trimToSize() call does and how it is implemented.)

3. (4 points) **Weiss 3.8**. The following routine removes the first half of the list passed as a parameter:

```
public static void removeFirstHalf( List<?> lst )
{
    int theSize = lst.size() / 2;

    for( int i = 0; i < theSize; i++ )
        lst. remove( 0 );
}
```

   (a) Why is **theSize** saved prior to entering the **for** loop?

   (b) What is the running time of **removeFirstHalf** if **lst** is an **ArrayList**?

   (c) What is the running time of **removeFirstHalf** if **lst** is a **LinkedList**?

   (d) Does using an iterator make **removeFirstHalf** faster for either type of **List**?

4. (4 points) **Weiss 3.24**. Write routines to implement two stacks using only one array. Your stack routines should not declare an overflow unless every slot in the array is used. (Pseudocode is fine.)

5. (4 points) **Weiss 3.29**. Write an algorithm for printing a **singly** linked list in reverse, using only constant extra space. This instruction implies that you cannot use recursion but you may assume that your algorithm is a list member function.

6. (4 points) **Weiss 3.34**. A linked list contains a cycle if, starting from some node p, following a sufficiant number of **next** links brings us back to node p. [Node] p does not have to be the first node in the list. Assume that you are given a linked list that contains N nodes. However, the value of N is unknown.

   (a) Design an $O(N)$ algorithm to determine if the list contains a cycle. You may use $O(N)$ extra space.

   (b) Repeat part (a), but use only $O(1)$ extra space. (Hint: Use two iterators that are initially at the start of the list, but advance at different speeds.)

7. (4 points) **Weiss 3.36**. Suppose we have a reference to a node in a singly linked list that is guaranteed *not to be the last node* in the list. We do not have references to any other nodes (except by following links). Describe an $O(1)$ algorithm that logically removes the value stored in such a node from the linked list, maintaining the integrity of the list. (Hint: Involve the next node).

2 Programming Problems

1. (12 points) **Palindrome Detector**. A palindrome is a phrase that reads the same forwards as it does backwards. For example, "a man, a plan, a canal, Panama," is a palindrome. Write a program that uses a stack to check for palindromes in each line of a text file. Try your program on the example text file,

   `http://www.cs.columbia.edu/~bert/courses/3137/palindromes.txt`

   Your program should output the palindromes that it finds in the document. For example:

   ```
   java FindPalindromes palindromes.txt
   "a man, a plan, a canal, Panama" is a palindrome.
   "Don't nod" is a palindrome.
   "Taco Cat!" is a palindrome.
   ...
   ```

   You must write your own MyStack class for this problem. Don't use the built in Stack. Feel free to use either a LinkedList, ArrayList or an array to implement the Stack.

2. (20 points) **Housing Queue Simulator**. Columbia University provides limited off-campus apartments (UAH) to graduate students. The housing system uses a waiting list for apartment assignments. According to UAH policy, when a possible match between applicant and apartment is found, the applicant is contacted via email and must decide whether to accept the apartment. If the applicant does not accept, he or she is placed back at the *beginning* of the waiting list, and must wait again.[1]

   (a) We will simulate this process using a queue, and simplify the rules. We will ignore the different types of apartments, and simply give each apartment a quality rating. Moreover, we will ignore the realistic constraints of the applicants, and simply have each applicant store a quality threshold. We will also assume that each apartment, once accepted, will be occupied for a random number of school years between 1 and 3, and that vacant apartments are filled between school years. This means our simulation will use the school year as its unit of time.

   - Assume k students apply for housing each school year. Each student node should contain a randomly generated quality threshold (a random double between 0 and 1), and an ID number (or String) to keep track. Get N and k from the user via command line arguments.
   - Create a list of N apartments. Each apartment should store its ID number or String and two pieces of information: its quality score (a random double between 0 and 1), and how many school years left in its occupancy (0 means it is vacant and ready to be assigned).
   - At the end of each school year, decrement the number of school years remaining for each occupied apartment, and run the assignment process for each vacancy.

[1] Note the process we are simulating is rather different than the actual UAH policies. This assignment is merely *inspired by* the UAH policy that you are put back into the waiting list if you reject an apartment.

- The assignment process should dequeue a student, check if the student "accepts" the apartment (is the apartment's quality value above the student's threshold?). If the student accepts, set the apartment to occupied for a random period of between 1 and 5 school years and delete the student node. If the student rejects the apartment, enqueue the student, dequeue the next student and repeat.
- If all students reject an apartment, move on to the next apartment. This means there may be vacancies in terrible apartments even if there are still students on the waiting list.
- The assignment process repeats until either there are no vacancies, all students are housed, or all unhoused students have seen all apartments. Once either of these occur, move to the next school year.
- To make the simulation more realistic (and to avoid an infinite loop during the assignment step), each time a student is put back on the waiting list, multiply his or her threshold by 0.9. This will simulate the student growing desperation.

Output statistics after each school year. Display the number of apartments vacated that year, the number of apartments newly filled. When k is small, output the steps of the process as they occur.

(b) Add a feature to the code that outputs the average years spent on the waiting list. Experiment with N and k to see how the two parameters affect the waiting time.

You must write your own MyQueue class for this problem. Do not use the built-in Queue interface.

Here is the makefile. Copy it into a text file named Makefile in your code directory.

```
#Begin 3137 Homework 2 Makefile
JFLAGS = -g
JC = javac
.SUFFIXES: .java .class
.java.class:
        $(JC) $(JFLAGS) $*.java

CLASSES = \
        MyStack.java
        MyQueue.java
        FindPalindromes.java
        HousingSimulator.java

default: classes

classes: $(CLASSES:.java=.class)

clean:
        $(RM) *.class

#end of file
```