**Hands-On Activity 3**

**Multiple Choice**

1. One difference between a queue and a stack is:
   1. Queues require linked lists, but stacks do not.
   2. Stacks require linked lists, but queues do not.
   3. Queues use two ends of the structure; stacks use only one.
   4. Stacks use two ends of the structure, queues use only one.
2. If the characters 'D', 'C', 'B', 'A' are placed in a queue (in that order), and then removed one at a time, in what order will they be removed?
   1. ABCD
   2. ABDC
   3. DCAB
   4. DCBA
3. I have implemented the queue with a linked list, keeping track of a front node and a rear node with two reference variables. Which of these reference variables will change during an insertion into a NONEMPTY queue?
   1. Neither changes
   2. Only front changes.
   3. Only rear changes.
   4. An exception is caused
4. To simulate people waiting in a line, which data structure would you use?
   1. Vector
   2. Queue C Stack
5. Set
6. List

**Algorithm Design (Linked Lists)**

1. Write a method that checks whether given two linked lists are identical (i.e. have the same items) or not. The method should have two parameters of type SlinkedList, and return true if the items are the same, false otherwise.

public class Node

{

int item;

Node next;

}

public class SLinkedList

{

private Node head = null;

// Linked list methods

…

}

**public** **static** **boolean** identical(SLinkedList s1, SLinkedList s2) {

Node curr1 = s1.head;

Node curr2 = s2.head;

**while** (curr1 != **null** && curr2 != **null**) {

**if** ((curr1.data != curr2.data) || (s1.size != s2.size)) {

**return** **false**;

}

curr1 = curr1.next;

curr2 = curr2.next;

}

**return** **true**;

}

1. Write a method that removes every second node – nodes with index 1, 3, 5, … starting from index 0 – from the given linked list. The method should have a single parameter of type SlinkedList, and return void.

**public** **static** **void** removeEverySecondNode(SLinkedList s) {

Node curr = s.head;

Node second = curr.next;

**while** (curr.next != **null**) {

curr.next = second.next;

second.next = **null**;

s.size--;

curr = curr.next;

second = curr.next;

}

}

**Algorithm Design (Linked Lists / Queues / Stacks)**

1. Write a **Java method** that moves all of the contents of a linked list into a queue. Use the standard linked list and queue operations. After the move is complete, the first node in the linked list should be at front of the queue.

public class Node

{

String item;

Node next;

}

public class SLinkedList

{

private Node head = null;

// Linked list methods

…

}

// Move all of the contents of the linked list onto a queue public void moveContents(SLinkedList list, Queue q)

{

**…**

}

**public** **static** **void** moveContents(SLinkedList list, Queue q) {

q = **new** Queue(list.size);

Node curr = list.head;

**for** (**int** i = 0; i < list.size; i++) {

q.num[i] = curr.data;

curr = curr.next;

}

System.***out***.println(Arrays.*toString*(q.num));

}