Homework 3

1. Question 1.3.6

What does the following code fragment do to the queue q?

```
Stack<String> stack = new Stack<String >();
while (!q.isEmpty())
    stack.push(q.dequeue());
while (!stack.isEmpty())
    q.enqueue(stack.pop());
```

The code fragment moves the queue of q into stack and then pops the contents of stack in the reverse order back into q, reversing the order of the elements of q.

2. Question 1.3.3

Suppose tahat a client performs an intermixed sequence of (stack) *push* and *pop* operations. The push operations put the integers 0 through 9 in order onto the stack; the pop operations print out the return values. Which of the following sequences could *not* occur?

a	4	3	2	1	0	9	8	7	6	5
b	4	6	8	7	5	3	2	9	0	1
c	2	5	6	7	4	8	9	3	1	0
d	4	3	2	1	0	5	6	7	8	9
e	1	2	3	4	5	6	9	8	7	0
f	0	4	6	5	3	8	1	7	2	9
g	1	4	7	9	8	6	5	3	0	2
h	2	1	4	3	6	5	8	7	9	0

b and g are impossible

3. Question 1.3.22 and 1.3.23

Suppose that x is a linked list Node. What does the follosing code fragment do?

```
t.next = x.next;
x.next = t;
```

The code fragment places the node t after the node x by setting the node after t to the node that was previously after x and setting the node after x to t.

Why does the following code fragment not do the same thing as in the previous question?

```
x.next = t;
t.next = x.next;
```

The code fragment makes t.next equal to t, splitting x and t from whatever came after x initially. x now points to t which points to itself.

4. Study Guide Question

Understand why the we prevent loitering by adding the line A[N] = null in Algorithm 1.1 on page 141 of the book. Why do we not have a special purpose line like this for our linked list implementation (page 149, algorithm 1.2)?

When a node has no more references to it in the program the garbage collector is allowed to free the memory regardless of whether or not the node references anything. This is different from an array implementation where the memory is still in use by the array structure until the cell is manually set to null.