CSC 115
Midterm Exam #2:
Sections: A01 and A02
Monday, June 29th, 2020

Name:____**KEY**_____(please print clearly!)

UVic ID number:_____

Signature:_____

Exam duration: 60 minutes

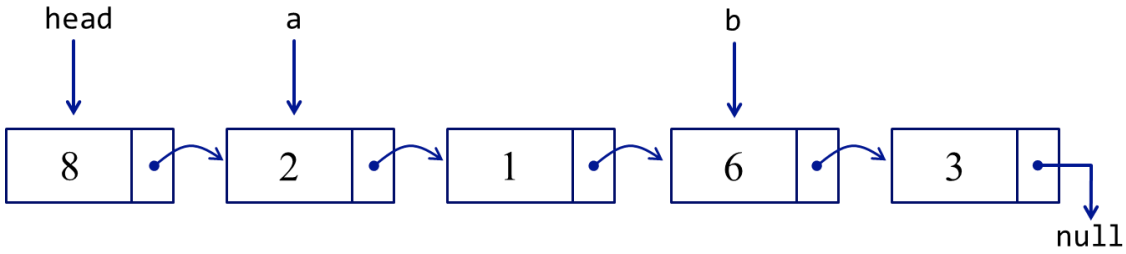
Instructor: Anthony Estey

Students must check the number of pages in this examination paper before beginning to write, and report any discrepancy immediately.

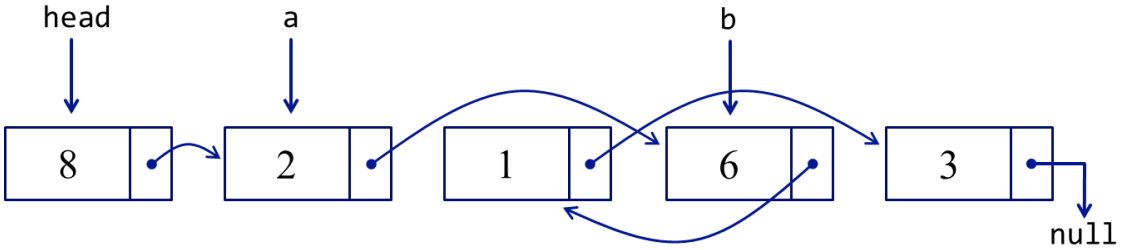
- We will not answer questions during the exam. If you feel there is an error or ambiguity, write your assumption and answer the question based on that assumption.
- Answer all questions on this exam paper.
- The exam is closed book. No books or notes are permitted.
  No electronic devices of any type are permitted.
- The marks assigned to each question and to each part of a question are printed within brackets. Partial marks are available.
- There are fifteen (15) pages in this document, including this cover page.
- Page 15 is left blank for scratch work. If you write an answer on that page, clearly indicate this for the grader under the corresponding question.
- Clearly indicate only one answer to be graded. Questions with more than one answer will be given a **zero grade**.
- It is strongly recommended that you read the entire exam through from beginning to end before beginning to answer the questions.
- Please have your ID card available on the desk.

Part 1: Linked Lists (14 marks)

1. Examine the following Nodes linked together, with Node pointers **head**, **a**, and **b**:



Write code to update next reference arrows to the following:



a) Write your code to in the box below:

```
a.next.next = b.next;
b.next = a.next;
a.next = b;
```

b) In the second diagram, what is the order the nodes are visited, beginning at **head** and traversing through the other nodes until the end of the sequence?

```
8 2 6 1 3
```

2. Implement the **`addFront`** method for a singly-linked list with the Node class defined below, in which Nodes ***only*** have a reference to the **PREVIOUS** Node in the list.

```
public class Node {
    public int data;
    public Node prev;

    public Node (int data) {
        this.data = data;
        this.prev = null;
    }
}
```

In the LinkedList class implementation, shown below, there is only a **`tail`** reference variable. **Note**: there is **NO HEAD** reference variable.

```
public class LinkedList {
    private Node tail;

    public LinkedList() {
        tail = null;
    }

    public void addFront(int val) {
        // TODO: implement this method
    }
}
```

a) Based on these restrictions, complete the implementation for the addFront method.

```
Node n = new Node(val);
if (tail == null) {
   tail = n;
} else {
   Node cur = tail;
   while (cur.prev != null) {
      cur = cur.prev;
   }
   cur.prev = n;
}
```

1 mark: new Node declared correctly
2 marks: sets tail to new node when empty
2 marks: otherwise loops from tail to front
- must loop until cur.prev != null NOT cur != null
- must create a temp Node, not use tail reference
2 marks: sets front node's prev to new node

b) What is the growth rate of the addFront method in Big-Oh terms? Assume n represents the number of elements in the list. Circle one answer

$O(1)$

$O(\log n)$

$O(n)$

$O(n^2)$

$O(n^3)$

Part 2: Recursion (7 marks)

3. Complete a **RECURSIVE** implementation of the **`getPosition`** method for a doubly-linked list with the Node class defined below:

```
public class Node {
    public String data;
    public Node next;
    public Node prev;

    public Node (String data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}
```

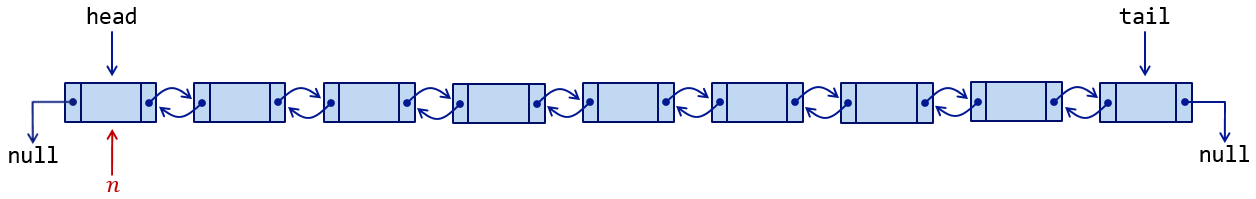
For this question the LinkedList class has head and tail references.

```
public class LinkedList {
    Node head;
    Node tail;

    public LinkedList() {
        head = null;
        tail = null;
    }

    /*
     * Purpose: return the number of places n is found from
     *          the beginning of the list
     * Parameters: Node n - the node to get the position for
     * Returns: int - the position
     * Preconditions: n is in the list, head and tail have
     *                been linked correctly, and all prev
     *                and next references are correct.
     */
    public int getPosition(Node n) {
        // TODO: Implement this method RECURSIVELY
    }
}
```

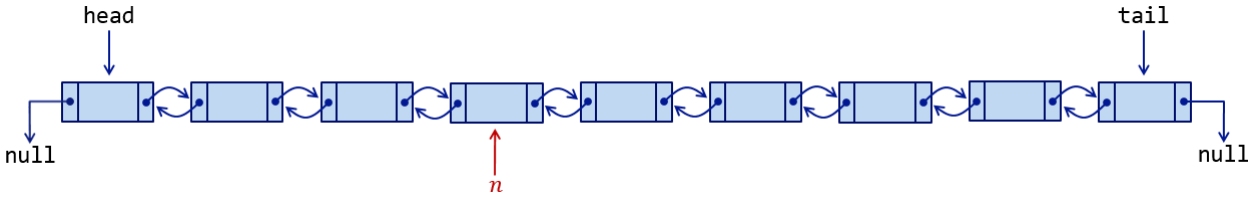
(The problem description is continued on the next page.)

The **`getPosition`** method is given a node from the linked list, and returns the number of places from the front of the list the node is positioned. For example:

Example 1: when Node *n* shown below is passed to `getPosition`, 0 is returned:



Example 2: when Node *n* shown below is passed to `getPosition`, 3 is returned:




```
public int getPosition (Node n) {

     if (n == head) {
        return 0;
     } else {
        return 1 + getPosition(n.prev);
     }

     2 marks: base-case checks if n==head
     1 mark: return 0 in base case
     2 marks: recursion called with n.prev
     2 marks: returns 1 + result of recursion









}
```

Part 3: Stacks (9 marks)

4. For this question you will work with an instance of the IntegerStack class, which is an implementation of the Stack interface shown below:

```
interface Stack {

    // Adds a new element with value v to the top of the stack
    public void push(int v);

    // Removes and returns value of the element at the top
    public int pop();

    // Returns the value of the element at the top of the stack
    public int top();

    // Returns the number of elements in the stack
    public int size();

    // Returns true if the stack is empty, false otherwise
    public boolean isEmpty();

}
```

Assume all of the methods specified in the Stack Interface have been implemented correctly in the IntegerStack class. There is no `isFull` method as the implementation allows for an unlimited number of elements to be inserted.

Complete the implementation of the countNegatives method specified below, which takes a reference to an IntegerStack as a parameter and returns a count of the number of negative numbers found in the stack.

```
 * Purpose: counts the number of negative values in a stack
 * Parameters: IntegerStack s - the stack to analyze
 * Returns: int - the number of negative values found
 * Post-conditions: The number and order of elements
 *                  in the stack are unchanged.
 */
public static int countNegatives(IntegerStack s) {
    // TODO: implement this method
}
```

Note: The countNegatives method is a static method defined in a **DIFFERENT** class than IntegerStack.java.

Similar to Assignment 4, you can use any of the Stack methods on the `IntegerStack` variable **`s`**, (ie. s.push(x) or s.pop()). You may create any other variables, including another IntegerStack, in your implementation of the `countNegatives` method.

You will receive marks for the following:

- returning the correct value representing the number negative integers found in the given stack (focus on this first)
- maintaining the order and number of elements in the stack when the value is returned (when the result is returned, the stack referenced by **`s`** should contain the same number of elements, in the same order, as it did originally).

```
public static int countNegatives (IntegerStack s) {


    public static int countNegatives(IntegerStack s) {
       int result = 0;
       IntegerStack temp = new IntegerStack();

       while (!s.isEmpty()) {
          int v = s.pop();
          if (v < 0) {
             result++;
          }
          temp.push(v);
       }

       while (!temp.isEmpty()) {
          s.push(temp.pop());
       }

       return result;
    }
```

pop elements until empty: 2 marks
check if popped value is negative: 2 marks
increment counter: 1 marks
order is maintained: 4 marks

```
}
```

Part 4: Exceptions (10 marks)

5. Carefully examine the following three methods, defined below:

```
public static void method1(int x, int y, int z) {
    try {
        method2(x, y);
        method3(y, z);
    } catch (ExceptionA e) {
        System.out.println("Caught A in method1");
    } catch (ExceptionB e) {
        System.out.println("Caught B in method1");
    } catch (ExceptionC e) {
        System.out.println("Caught C in method1");
    }
}
public static void method2(int h, int i) throws ExceptionA, ExceptionC {
    if (h > i) {
        throw new ExceptionA();
    }
    try {
        method3(h, i);
    } catch (ExceptionB e) {
        System.out.println("Caught B in method2");
    }
}
public static void method3(int j, int k) throws ExceptionB, ExceptionC {
    if (j == k) {
        throw new ExceptionB();
    }
    if (k == 0) {
        throw new ExceptionC();
    }
    // Finished method 3!
}
```

For this question you will be determining if different outputs are possible by calling `method1` with different input values for x, y, and z.

For example, your answer might be: **method1(1, 2, 3);**

a) Is it possible to call method1 and produce only output **"Caught A in method1"**? If so, provide an example method1 call, if not, simply write **no**.

Yes, any time x>y. Ex: method1(3, 2, 1);

b) Is it possible to call method1 and produce only output **"Caught B in method1"**? If so, provide an example method1 call, if not, simply write **no**.

Yes, when x<y and y==z. Ex: method1(1, 2, 2);

c) Is it possible to call method1 and produce only output **"Caught C in method1"**? If so, provide an example method1 call, if not, simply write **no**.

Yes, when x<y and z==0. Ex: method1(0, 1, 0);

d) Is it possible to call method1 and produce only output **"Caught B in method2"**? If so, provide an example method1 call, if not, simply write **no**.

Yes, when x==y. Ex: method1(1, 1, 2);

e) Is it possible to call method1 and have it complete execution without any exceptions being thrown? If so, provide an example method1 call, if not, simply write **no**.

Yes, when x<y, y!=z, and y,z > 0. Ex: method1(1, 2, 3);

f) Is it possible to call method1 and have ExceptionA, ExceptionB, or ExceptionC be thrown and never caught? If so, provide an example method1 call, if not, write **no**.

No

g) Is it possible to call method1 and produce output: **"Caught A in method1"** followed by **"Caught B in method1".** If so, provide an example method1 call, if not, write **no**.

No

h) Is it possible to call method1 and produce output: **"Caught B in method2"** followed by **"Caught B in method1".** If so, provide an example method1 call, if not, write **no**.

Yes, x==y==z. Ex: method1(1, 1, 1);

i) Is it possible to call method1 and produce output: **"Caught B in method2"** followed by **"Caught C in method1".** If so, provide an example method1 call, if not, write **no**.

Yes, x==y, y!=z, and z==0. Ex: method1(1, 1, 0);

j) Is it possible to call method1 and produce output: **"Caught C in method1"** followed by **"Caught B in method1".** If so, provide an example method1 call, if not, write **no**.

No

… Left blank for scratch work…

END OF EXAM

Question	Value	Mark
Part 1	14	
Part 2	7	
Part 3	9	
Part 4	10	
Total	**40**	