# CMPSCI 187

# Midterm 1

# October 15, 2015

**100 Points**

**75 Minutes**

**Name:** ________________________________________

**Student ID:** ____________________________________

| Problem: | 1 | 2 | 3 | 4 | Total |
|---|---|---|---|---|---|
| Maximum: | 28 | 20 | 26 | 26 | 100 |
| Score: | | | | | |

**Question 1: Short Answer (28 points)**

a) All but one of the following statements are true about Java's Iterator<E> and Iterable<E> interfaces. Which one is false?

A) A class that implements Iterator<E> can take advantage of Java's enhanced for loop (eg. *for (String e : a))*.

B) The Iterable<E> interface provides an iterator() method which returns an Iterator<E>.

C) The Iterator<E> interface provides a hasNext() method which returns true if the iteration has more elements.

D) The Iterator<E> interface provides a next() method which returns the next element in the iteration.

b) Suppose we are using a linked list in which the list pointer L points to the first node. If L points to a list containing a sequence of integers [1,2,3,4,5], what list would be represented by L after execution of the statement L.getNext().getNext().setNext(null);

c) What is the value of helloIsWorld?

```
public static void main(String[] args){
     String hello = new String("hello");
     String world = new String("hello");
     boolean helloIsWorld = (hello == world);
}
```

d) What does the following piece of code output when we call main()?

```
class Base {
    public void Print() {
        System.out.println("Base");
    }
}

class Derived extends Base {
    public void Print() {
        System.out.println("Derived");
    }
}

class Main{
    public static void main(String[] args) {
        Base x = new Base();
        Base y = new Derived();
        Derived z = new Derived();
       x.Print();
       y.Print();
       z.Print();
    }
}
```

e) Find the mistake in this piece of code. Assume that the user of this iterator properly checks hasNext() before calling next. Also, we omitted the remove method to simplify the code.

```
class LinkedNodeIterator<E> implements Iterator<E> {
  private LinkedNode<E> current;

  public LinkedNodeIterator(LinkedNode<E> head) {
    this.current = head;
  }

  public boolean hasNext() {
    return this.current.getNext() != null;
  }

  public E next() {
    E result = current.getData();
    current = current.getNext();
    return result;
  }
}
```

f) Imagine you are using a stack to evaluate a long arithmetic postfix expression. So far you have evaluated this part of the expression: 5 2 1 - + 6 3 / 7 1. Draw the state of the stack at this point in the evaluation:

g) What is the output of running the following code?

```
class Code {
     public static void foo(int i) throws RuntimeException {
          System.out.print("1");
          if (i < 0)
               throw new RuntimeException("2");
          System.out.print("3");
     }

     public static void main(String[] args) {
          System.out.print("1");
          try {
               System.out.print("4");
               foo(-1);
               System.out.print("5");
          }
          catch(RuntimeException e) {
               System.out.print( e.getMessage() );
          }

          foo(1);

          System.out.print("6");
     }
}
```

**Question 2 Complexity (20 points)**

a. What is the Big-O execution time of the **contains** operation when using an (unsorted) linked list, assuming a list size of N?

b. What is the Big-O execution time of the **size** operation of linked-list based stack when keeping a size instance variable (assume push/pop update size, but do not consider the complexity of push/pop)?

c. What is the Big-O execution time of the **size** operation of a linked-list based stack if you iterate through linked list and count up all elements in the list (i.e. without keeping a size instance variable)?

d. What is the Big-O execution time of the push operation of a linked-list based stack?

e. What is the Big-O execution time of the following function, for input n?

```
public void printfoo(int n) {
      for (i = n; i > 0; i--){
            for (j = n; j > 0; j-- ){
                  System.out.println("foobar");
            }
      }
}
```

**Question 3 Data Structures with Arrays (26 points)**

Fill in the following methods for a stack implementation using an array with a slight twist: we want to keep the top element in position 0 of the array (a[0]) at all times.

Note that here we just implement pop and it returns the top element. There is no top implementation necessary. You only need to implement push and pop. There is no need for any new instance variables.

```
public class ArrayStack<T> implements BoundedStackInterface<T> {

     protected final int DEFCAP = 100; //default capacity
     protected T[ ] stack; // holds stack elements
     protected int size;

     public ArrayStack( ) {
          stack = (T[ ]) new Object[DEFCAP];
     }

     public ArrayStack(int maxSize) {
          stack = (T[ ]) new Object[maxSize];
     }

     public void push (T element) throws StackOverflowException {









     }
```

```
    public T pop( ) throws StackUnderflowException{











    }

    public boolean isEmpty( ) {
        return (size == 0);
    }

    public boolean isFull( ) {
        return (size == stack.length);
    }

    public int size( ) {
        return size;
    }
}
```

**Question 4 Data Structures with LinkedLists (26 points)**

Build a stack implementation using a linked list. **Your size() method should be O(1).**

Use the following definition of an LLNode<T>. Do not modify the LLNode<T>.

```
public class LLNode<T> {
     private T data;
     private LLNode<T> next;

     public LLNode(T data) {
          this.data = data;
     }
     public T getData() {
          return data;
     }
     public void setData(T data) {
          this.data = data;
     }
     public LLNode<T> getNext() {
          return next;
     }
     public void setNext(LLNode<T> next) {
          this.next = next;
     }
}

public class LinkedStack<T> implements StackInterface<T> {

     LLNode<T> head;




     public LinkedStack(){




     }
```

```
public int size() {  // Your size() method should be O(1)

}

public boolean isEmpty() {

}

public T pop() throws StackUnderflowException {

}
```

```
        public void push(T elem) {

























        }
}
```