Unit 8 Practice Test

____ 1. Which of the following data structures would be most appropriate for a program that simulates the operation of an airport in order to determine the maximum delays encountered by arriving and departing planes?

A. queue
B. binary tree
C. bitset
D. string
E. array

____ 2. A stack `S` of integers initially contains the following data:

6 (top)
2
7
3

The following code is then executed:

```
int x = S.pop();
int y = S.pop();
int z = S.pop();
S.push(x+y);
int w = S.pop();
S.push(w+z);
```

After this code has been executed, what are the contents of the stack?

A.
8
10

B.
10
8

C.
15
3

D.
10
8
6
3

E. The stack is empty.

Questions 3 and 4 involve singly linked lists constructed out of nodes defined as follows:

```
public class Node {
    public int datum;
    public Node next;
}
```

In all of methods shown in both of these questions, the parameter `first` refers to the first node in the list, if there is one, and has the value `null` otherwise.

____ 3. Which of the following methods correctly inserts a value `x` at the front of the list and returns a reference to the new front of the list?

I.
```
public Node insertFront(Node first, int x)
{
     first = new Node();
     first.datum = x;
     first.next = first;
     return first;
}
```

II.
```
public Node insertFront(Node first, int x)
{
     Node n = new Node();
     n.datum = x;
     n.next = first;
     return n;
}
```

III.
```
public Node insertFront(Node first, int x)
{
     Node n = new Node();
     first = n;
     n.datum = x;
     n.next = first;
     return first;
}
```

A. I only
B. II only
C. III only
D. I and II
E. II and III

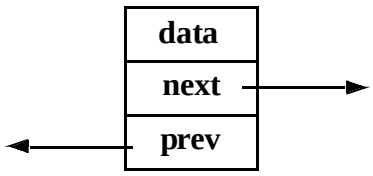
____ 4. The intent of the method below is to delete the last node of the list.

```
public void removeLast(Node first) {
     Node p, q;

     p = first;
     q = p.next;

     while (q.next != null) {
          p = q;
          q = q.next;
     }

     p.next = null;
}
```

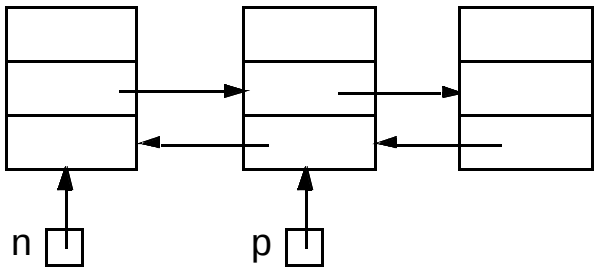
Which of the following describes the class of all linked lists for which this method works correctly?

A. No linked lists
B. All nonempty linked lists
C. All linked lists with more than one node
D. The empty list and all linked lists with more than one node
E. All linked lists

____ 5. Nodes for a doubly linked list are defined to have the following structure:



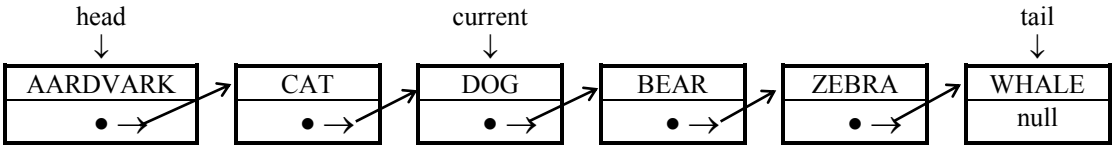
The `next` instance variable stores a reference to the next node in the list, and the `prev` instance variable refers to the previous node in the list. Below is a list of three of these nodes, along with two reference variables, `p` and `q`, that refer to specific nodes in the list.



Which of the following expressions does *not* refer to the third node in the list?

A. `p.next`
B. `n.next.next`
C. `p.prev.next`
D. `p.prev.next.next`
E. `n.next.next.prev.next`

6. The diagram below depicts a linked list of strings that has been partially sorted by a version of the *insertion sort* algorithm that has been adapted to work with linked lists. As with the array-based version of the algorithm, we consider the elements of the list from left to right, and we insert each element in the correct position with respect to the elements that come before it in the list. However, because we're dealing with a linked list, the algorithm can't perform a backwards pass to determine the appropriate location for a given element. Rather, it performs a *forwards* pass to find the point of insertion.



Assume the following declarations have been made:

```
public class Node {
    public String id;
    public Node next;
}

Node head;
Node tail;
Node temp;
Node current;
```

`head` refers to the first element of the list, `tail` to the last element, and `current` to the last element in the portion of the list that is known to be correctly ordered.

(a) Show on the diagram above how references will be modified after the node with the id "BEAR" has been moved by the algorithm to its appropriate position in the list.

(b) When insertion sort is used for sorting a large array of **n** elements, both the number of data moves and the number of comparisons are proportional to **n*n**. For sorting a linked list, one aspect of this behavior is better. State which, and explain why.

(c) Write some lines of Java code (*not* a method) that test whether there is a node beyond the one that `current` refers to and, if there is, inserts that node in its appropriate position in the linked list, changing `current`, `head`, and `tail` if necessary. You may use as many temporary reference variables as you need. (Some techniques require more than others.)

Note: To determine whether one id string "comes before" another id string (i.e., whether it would appear before the other string in a sorted list), you should make use of a `String` method called `compareTo()`:

`str1.compareTo(str2)` will return a value that is < 0 if `str1` comes before `str2`
`str1.compareTo(str2)` will return a value that is > 0 if `str1` comes after `str2`
`str1.compareTo(str2)` will return 0 if `str1` and `str2` represent the same string