Com S 228

Spring 2015

Exam 2 Sample Solution

1.

Code snippet	**Output**	**List and iterator state, or exception thrown**
`iter = aList.listIterator();`	(none)	`|A B C D E`

`// 3 pts`
`iter = aList.listIterator();`
`iter.next();`
`System.out.println(iter.previousIndex());`
	0	`A | B C D E`

`// 5 pts`
`iter = aList.listIterator();`
`while (iter.hasNext())`
`    System.out.println(iter.next());`
`System.out.println(iter.next());`
	A
	B
	C
	D
	E
		`NoSuchElementException`

`// 4 pts`
`aList.add("O");`
`iter = aList.listIterator(aList.size()/2);`
`while (iter.hasPrevious())`
`{`
`    iter.set("X");`
`    iter.remove();`
`    iter.previous();`
`}`
	(none)	`IllegalStateException`

`// 8 pts`
`iter = aList.listIterator(aList.size());`
`List<String> bList = new ArrayList<String>();`
 	
`while (iter.hasPrevious())`
`{`
`    bList.add(iter.previous());`
`    iter.remove();`
`}`
 	
`iter = bList.listIterator();`
 	
`while (iter.hasNext())`
`    aList.add(iter.next());`
		(need only show aList)
		
		aList: A B C D E |
		bList: (empty)
		
		After the 1st while loop:
		aList: |
		bList: E D C B A
		
		bList: | E D C B A
		
		After the 2nd while loop:
		aList: **E D C B A**
		bList: E D C B A |

`// 8 pts`
`iter = aList.listIterator();`
`iter2 = aList.listIterator(aList.size());`
 	
`while (iter.nextIndex() < iter2.previousIndex())`
`{`
`    String s = iter.next();`
`    String t = iter2.previous();`
`    iter.set(t);`
`    iter2.set(s);`
`}`
 	
`iter.next();`
`iter.remove();`
		`| A B C D E |`
		
		`iteration 1:`
		`A | B C D E |     (s = A)`
		`A | B C D | E     (t = E)`
		`E | B C D | A`
		
		`iteration 2:`
		`E B | C D | A     (s = B)`
		`E B | C | D A     (t = D)`
		`E D | C | B A`
		
		`E D C || B A`
		`E D || B A`

2a)

i) $O(n)$
ii) $O(1)$

b) $O(n)$ — 1/2 partial credit for $O(n^2)$

3a)

```
2 a b ^ ^ c * d e + 2 * f g h - ^ + - 5 i j + / -
```

b)

```
a * b / 2 ^ c ^ d - 3 * ((e - f) * (g - h) + 5)
```

4.

```
import java.util.Stack;

// (2 pts) fill in the condition on E below so that compareTo()
// is available to E objects.
public class CDLinkedList <E extends Comparable<? super E>>
{
      private class Node
      {
            public E data;
            public Node next;
            public Node previous;
      }

      private Node head;  // dummy node
      private int size;

      public CDLinkedList()
      {
        head = new Node();
        head.next = head;
        head.previous = head;
        size = 0;
}

/**
 * (2 pts) Unlink a node from the list.
 * Precondition: node != head
 */
private void unlink(Node node)
{
        // insert code below (2 pts)
        node.previous.next = node.next;
        node.next.previous = node.previous;
}

/**
 * (4 pts) Insert newNode into the list after cur without
 * updating size.
 *
 * Precondition: cur != null && newNode != null
 */
private void link(Node cur, Node newNode)
{
        // insert code below (4 pts)
        newNode.next = cur.next;
        cur.next.previous = newNode;
        newNode.previous = cur;
        cur.next = newNode;
}


/**
 * (8 pts) Add to the beginning of the linked list.  Do not add
 * if an equal value is already in the list, as determined by
 * compareTo().
 *
 * @param  val
 * @return true  if an insertion takes place
 *         false otherwise
 */
public boolean add(E val)
{
        // search the linked list for val.
        Node node = head.next;
        while (node != head)
        {
                // insert the condition in the if statement below (2 pts)
                if (node.data.compareTo(val) == 0)
                        return false;

                // insert one line of code below (2 pts)
                node = node.next;
        }

        // if val is not found, then create a node and add it to the
        // linked list.

        // insert code below (4 pts)
        Node newNode = new Node();
        newNode.data = val;
        link(head, newNode);
        size++;

        return true;
  }

  /**
   * (6 pts) Search for the node storing a given value, and remove it
   * if found.
   *
   * @param  val
   * @return true  if val is found
   *         false otherwise
   */
  public boolean remove(E val)
  {
        // search the linked list for val.
        Node cur = head.next;

        // insert condition in the while statement below (2 pts)
        while (cur != head && cur.data.compareTo(val) != 0)
              cur = cur.next;

        // insert condition in the if statement below (2 pts)
        if (cur == head)
              return false;  // val not found
        else
        {
              // insert code below (2 pts)
              unlink(cur);
              size--;

              return true;
        }
  }


  /**
   * (14 pts) Rearrange the list.  The parameter val splits the nodes
   * into two groups.  The first group includes those nodes that store
   * data less than or equal to val, and the second group includes
   * those nodes that store data greater than val.  After the
   * rearrangement, the following two conditions must be satisfied.
   *
   *     a) The nodes from the first group must precede those from the
   *        second group.
```

```
         *     b) Within each group, the original order of the nodes is
         *        reversed.
         *
         * You are asked to use two stacks for the task.
         *
         */
        public void rearrange(E val)
        {
               // declare two stacks
               Stack<E> stk1 = new Stack<E>();
               Stack<E> stk2 = new Stack<E>();

               // split the values stored in the linked list using the stacks.
               // remove these values from the list in the same time.
               Node cur = head.next;
               while(cur != head)
               {
                      // insert condition in the if statement below (2 pts)
                      if (cur.data.compareTo(val) <= 0)
                             // insert a line of code below (2 pts)
                             stk1.push(cur.data);
                      else
                             // insert a line of code below (2 pts)
                             stk2.push(cur.data);

                      // insert code below (2 pts)
                      unlink(cur);
                      cur = cur.next;
               }

               // pop the values out of the two stacks to reconstruct the
               // linked list.
               while (!stk1.empty())
               {
                      // insert code below (3 pts)
                      Node node = new Node();
                      node.data = stk1.pop();
                      link(head.previous, node);
               }

               while (!stk2.empty())
               {
                      // insert code below (3 pts)
                      Node node = new Node();
                      node.data = stk2.pop();
                      link(head.previous, node);
               }
        }
}
```