Solution to CSE143X Section #9 Problems

1. Basic list manipulation.

before	after	code
p->[1]->[2]->[3] q	p->[1]->[2] q->[3]	q = p.next.next; p.next.next = null;

p->[1]->[2]
 	
q->[3]
	p->[2]
	
	q->[3]->[1]
		q.next = p;
		p = p.next;
		q.next.next = null;

p->[1]->[2]
 	
q->[3]
	p
	
	q->[1]->[2]->[3]
		p.next.next = q;
		q = p;
		p = null;

p->[1]->[2]
 	
q->[3]
	p->[1]->[3]->[2]
	
	q
		q.next = p.next;
		p.next = q;
		q = null;

p->[1]->[2]
 	
q->[3]->[4]
	p->[1]->[3]->[4]->[2]
	
	q
		q.next.next = p.next;
		p.next = q;
		q = null;

p->[5]->[4]->[3]	p->[4]->[3]->[5]	p.next.next.next = p;
		p = p.next;
		p.next.next.next = null;

p->[1]->[2]
 	
q->[3]
	p->[3]->[1]->[2]
	
	q
		q.next = p;
		p = q;
		q = null;

2. Intermediate list manipulation.

before	after	code

p->[1]->[2]
 	
q->[3]->[4]
	p->[1]->[3]->[2]->[4]
	
	q
		p.next.next = q.next;
		q.next = p.next;
		p.next = q;
		q = null;

p->[1]
 	
q->[2]->[3]
	p->[3]
	
	q->[1]->[2]
		p.next = q;
		q = p;
		p = p.next.next;
		q.next.next = null;

p->[1]->[2]
 	
q->[3]->[4]
	p->[2]->[4]
	
	q->[3]->[1]
		p.next.next = q.next;
		q.next = p;
		p = p.next;
		q.next.next = null;

(2, cont.) Intermediate list manipulation.

before	after	code
p->[1]->[2]->[3]	p->[2]	q = p; p = p.next;
q	q->[1]->[3]	q.next = q.next.next; p.next = null;

p->[5]->[4]->[3]	p->[4]->[5]->[3]	ListNode temp = p;
		p = p.next;
		temp.next = p.next;
		p.next = temp;

3. List manipulation involving construction.

before	after	code
p->[1]->[2]	p->[1]->[2]->[3]	p.next.next = new ListNode(3, null);
p->[1]->[2]	p->[3]->[1]->[2]	p = new ListNode(3, p);
p->[1]->[2]	p->[1]->[3]->[2]	p.next = new ListNode(3, p.next);

4. Advanced list manipulation.

before	after	code

p->[5]->[4]->[3]	p->[3]->[4]->[5]	ListNode temp =
		  p.next.next;
		temp.next = p.next;
		p.next.next = p;
		p.next.next.next = null;
		p = temp;

p->[1]->[2]
q->[3]->[4]
	p->[3]->[2]->[1]
	q->[4]
		p.next.next = p;
		ListNode temp = q.next;
		q.next = p.next;
		p.next = null;
		p = q;
		q = temp;

p->[1]->[2]
q->[3]->[4]->[5]
	p->[1]->[3]->[5]
	q->[2]->[4]
		ListNode temp = p.next;
		p.next = q;
		temp.next = q.next;
		q.next = q.next.next;
		q = temp;
		q.next.next = null;

(4, cont.) Advanced list manipulation.

before	after	code

p->[1]->[2]->[3]
 	
q->[4]->[5]
	p->[2]->[1]->[4]
	
	q->[3]->[5]
		p.next.next.next =
		  q.next;
		ListNode temp = q;
		q = p.next.next;
		p.next.next = p;
		p = p.next;
		p.next.next = temp;
		temp.next = null;

p->[1]->[2]->[3]
 	
q->[4]->[5]
	p->[5]->[4]->[3]->[2]
	
	q->[1]
		q.next.next = q;
		q = q.next;
		q.next.next =
		  p.next.next;
		p.next.next.next =
		  p.next;
		p.next.next = null;
		p.next = null;
		ListNode temp = q;
		q = p;
		p = temp;

5. LinkedIntList Development.

```
// Class LinkedIntList can be used to store a list of integers.

public class LinkedIntList {
    private ListNode front;  // first value in the list

    // post: constructs an empty list
    public LinkedIntList() {
        front = null;
    }

    // post: returns the current number of elements in the list
    public int size() {
        int count = 0;
        ListNode current = front;
        while (current != null) {
            current = current.next;
            count++;
        }
        return count;
    }

    // pre : 0 <= index < size()
    // post: returns the integer at the given index in the list
    public int get(int index) {
        ListNode current = front;
        for (int i = 0; i < index; i++) {
            current = current.next;
        }
        return current.data;
    }
```

```
// post: creates a comma-separated, bracketed version of the list
public String toString() {
    if (front == null) {
        return "[]";
    } else {
        String result = "[" + front.data;
        ListNode current = front.next;
        while (current != null) {
            result += ", " + current.data;
            current = current.next;
        }
        result += "]";
        return result;
    }
}

// post : returns the position of the first occurrence of the given
//        value (-1 if not found)
public int indexOf(int value) {
    int index = 0;
    ListNode current = front;
    while (current !=  null) {
        if (current.data == value) {
            return index;
        }
        index++;
        current = current.next;
    }
    return -1;
}

// post: appends the given value to the end of the list
public void add(int value) {
    if (front == null) {
        front = new ListNode(value);
    } else {
        ListNode current = front;
        while (current.next != null) {
            current = current.next;
        }
        current.next = new ListNode(value);
    }
}

// pre: 0 <= index <= size()
// post: inserts the given value at the given index
public void add(int index, int value) {
    if (index == 0) {
        front = new ListNode(value, front);
    } else {
        ListNode current = front;
        for (int i = 0; i < index - 1; i++) {
            current = current.next;
        }
        current.next = new ListNode(value, current.next);
    }
}
```

```
    // pre : 0 <= index < size()
    // post: removes value at the given index
    public void remove(int index) {
        if (index == 0) {
            front = front.next;
        } else {
            ListNode current = front;
            for (int i = 0; i < index - 1; i++) {
                current = current.next;
            }
            current.next = current.next.next;
        }
    }
}
```