Problem 1

```
public boolean delete(String target) {

  //first case we need to check is if list is empty
  if (rear == null) {
    return false;
  }

   //second case we need to check is if list only has one node
  if (rear == rear.next) {
    if (rear.data.equals(target)) {
      rear = null;
      return true;
    }
    else {
      return false;
    }
  }

   //third case is if there is a regular list
  Node previous = rear;
  Node current = rear.next;
  do {
    if (target.equals(current.data)) {
      previous.next = current.next;
      if (current == rear) {
        rear = previous;
      }
      return true;
    }

    previous = current;
    current = current.next;
  }
  while (previous != rear);

  //target does not exist in the linked list
  return false;
}
```

Problem 2

```
public boolean addAfter(String newItem, String afterItem){

  //Checking if list is empty
  if (rear == null) {
    return false;
  }

  Node pointer = rear;
  do {
    if (pointer.data.equals(afterItem)) {
      Node temp = new Node(newItem, pointer.next);
      pointer.next = temp;
      if (pointer == rear) {
        rear = temp;
      }
      return true;
    }
    pointer = pointer.next;
  }
  while (pointer != rear);


  //If afterItem does not exist in the list
  return false;
}
```

Problem 3

```
public static DLLNode moveToFront(DLLNode front, DLLNode target) {

  // If front doesn't exist, target doesn't exist, or if target is at the front we aren't supposed to do anything
  if (front == null || target == null || target == front) {
    return target;
  }

  // Set the node before target next value to the node after target
  target.prev.next = target.next;

  // Need to make sure node after target isn't null to do operations
  if (target.next != null) {
    target.next.prev = target.prev;
  }

  target.prev = null;
  target.next = front;
  front.prev = target;
  return target;
}
```

Problem 4

```
public static DLLNode reverse(DLLNode front) {

  //Check if list is empty
  if (front == null) {
    return null;
  }

  //Create a temp node to hold the value
  DLLNode rear = front;
  DLLNode prev = null;
  while (rear != null) {
    DLLNode temp = rear.next;
    rear.next = rear.prev;
    rear.prev = temp;
    prev = rear;
    rear = temp;
  }
  return prev;
}
```

Problem 5

```
public static Node deleteAll(Node front, String target) {

  //Check if list is empty
  if (front == null) {
    return null;
  }

  //If the value to delete is at the front
  if (front.data.equals(target)) {
    return deleteAll(front.next, target);
  }

  //If the value to delete is anywhere else in the list
  front.next = deleteAll(front.next, target);

  return front;
}
```

Problem 6

```
public static Node merge(Node frontL1, Node frontL2) {

  //If List1 is null, return the entirety of List2
  if (frontL1 == null) {
    return frontL2;
  }

  //If List2 is null, return the entirety of List1
  if (frontL2 == null) {
    return frontL1;
  }

  //If the two list’s nodes are equal, return that value (once) and merge the rest of the lists
  if (frontL1.data == frontL2.data) {
    frontL1.next = merge(frontL1.next, frontL2.next);
    return frontL1;
  }

  //If List1 node is less than List2 node, return List1 node and merge the rest of the sets
  if (frontL1.data < frontL2.data) {
    frontL1.next = merge(frontL1.next, frontL2);
    return frontL1;
  }

  //If List2 node is less than List1 node, return List2 node and merge the rest of the sets
  if (frontL1.data > frontL2.data) {
    frontL2.next = merge(frontL2.next, frontL1);
    return frontL2;
  }
}
```