1. public boolean delete(String target) {

Node prev = rear;

Node current = rear.next;

while (rear != current) {

if (current.data.equals(target)) {

prev.next = current.next; //deletes a node

return true; //deletion successful

}

prev = current;

current = current.next; //moves on to next nodes

}

if (rear == current) { //last node is being checked

if (current.data.equals(target)) {

rear = prev;

prev.next = current.next; //deletes last node

return true; //deletion successful

}

}

return false; //no target was found

}

2. public boolean addAfter(String newItem, String afterItem) {

Node current = rear.next;

while (rear != current) { //not at last node

if (current.data.equals( afterItem)) {

Node newNode = new Node(newItem, current.next); //creates new node with newItem

current.next = newNode; //inserts new Node after current node

if (current == rear) { //at last node

this.rear = newNode; //new node becomes last node

}

return true; //insertion successful

}

current = current.next; //moves on to next node

}

return false; //no targeted item to add after from list (insertion unsuccessful)

}

3. public static DLLNode moveToFront(DLLNode front, DLLNode target) {

if (target == null)

return front; //no nodes in list or when target isn’t found

DLLNode current = target;

DLLNode prev = null;

while (current != null) { //loops until last node is reached

prev = current;

current = current.next;

} //prev will become last node after loop

prev.next = front; //last (prev) node will point to first node (last node become first node)

front.prev = prev; //first (prev) node will point back to last node (due to DLL)

return target; //new node is front of the list

}

4. public static DLLNode reverse(DLLNode front) {

DLLNode temp = null;

DLLNode current = front;

DLLNode check= null;

while (current != null) {

check = current;

temp = current.next;

current.next = current.prev; //current.setNext(previous)

current.prev = temp; //current.setPrevious(next)

current = current.prev; //current.getPrevious()

}

return check;

}

5. public static Node deleteAll(Node front, String target) {

if (front == null) return null; //if it’s empty list

if (target == null) return front; //nothing to delete

if (front.data == target) {

return (deleteAll(front.next,target)); //skips current node (not saved)

}

else {

return new Node(front.data, deleteAll(front.next,target));

} //else: creates new nodes that saves data not same value as target

}

6. public static Node merge(Node frontL1, Node frontL2) {

if(frontL1== null) return frontL2; //if it’s empty list, moves on to merge 2nd list

if(frontL2 == null) return frontL1; //if it’s empty list, moves on to merge 1st list

if(frontL1.data<frontL2.data) return new Node(frontL1.data,merge(frontL1.next,frontL2));

else if(frontL1.data > frontL2.data) return new Node(frontL2.data,merge(frontL1,frontL2.next));

else if(frontL1.data == frontL2.data) return new Node(frontL2.data,merge(frontL1.next,frontL2.next));

return null;

}