1)

public static IntNode addBefore(IntNode front, int target, int newItem) {

//creates current & previous nodes

IntNode current = front;

IntNode prev = null; //target node is first item in linked list

//when target isn’t reached, moves on to next node in linked list

while(current != null && current.data !=target) {

prev = current;

current = current.next;

}

//target hasn’t been found (so it returns original front)

if (current == null) {

return front;

}

//creates new node inserted in linked list before target

IntNode temp = new IntNode(newItem, current);

//initialized “newItem” data that points current

if (prev == null) {

return temp; //target node is first item in linked list & temp becomes new front

}

prev.next = temp; //inserts before target

return front; //front unchanged

}

2)

public static IntNode addBeforeLast(IntNode front, int item) {

//this class adds an integer before the last integer (aka second to last integer)

//List is empty, so there is no last node to add before

if (front == null)

return front;

IntNode current = front;

IntNode prev = null;

//when target isn’t reached, moves on to next node in linked list

while(current.next != null) {

prev = current;

current = current.next;

}

//list has only one node, creating a new node as the new front (head)

if (prev == null)

return new IntNode (item,current);

//creates new node (2nd to last) before previous (which is last integer)

prev.next = new IntNode(item,current);

return front; //returns original front

}

3)

public static int numberOfOccurrences(StringNode front, String target) {

//method counts how many times current’s Strings equals to target’s Strings

int count = 0;

StringNode current = front;

while(current != null) {

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

count++;

}

current = current.next;

}

return count; //returns & shows how many times current’s String equals to target

}

4)

public static void deleteEveryOther(IntNode front) {

if (front == null)

return;

int dice = 0;

IntNode current = front;

IntNode prev = null;

while(current != null) {

if (dice == 0) {

prev = current;

current = current.next;

dice = 1;

}

else if (dice == 1) {

prev.next = current.next;

current = prev.next;

dice = 0;

}

}

}

5)

public static StringNode deleteAllOccurrences(StringNode front, String target) {

//method removes multiple occurrences of strings

if (front == null)

return null;

while (front != null && front.data.equals(target)) {

front = front.next;

}

StringNode current = front;

StringNode prev = null;

while (current != null) {

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

prev.next = current.next;

current = prev.next; //skips references in linked list when current = target

}

else {

prev = current;

current = current.next; //moves onto next node in linked list

}

}

return front;

}

6)

public static IntNode commonElements(IntNode frontL1, IntNode frontL2) {

IntNode currentL1 = frontL1;

IntNode currentL2 = frontL2;

IntNode currentL3 = null;

IntNode bookmark = null;

while(currentL1 != null) {

while(currentL2 != null) {

if (currentL1.data == currentL2.data) { //find common elements from 2 linked lists

if (currentL3 == null) {

currentL3 = new IntNode(currentL1.data, null); //list with common element data

bookmark = currentL3;

} else {

bookmark.next = new IntNode(currentL1.data, null);

bookmark = bookmark.next;

}

}

currentL2 = currentL2.next;

}

currentL2 = frontL2;

currentL1 = currentL1.next; //compares one element from 1st list with other list elements

}

return currentL3; //returns new list with the common elements

}