Alexandria University

Faculty of Engineering

Communication and Electronics Department

DS Sheet 2

Linked Lists

Name/ Mostafa Ahmed Mohamed Rashed
ID/ 19017528
Section/ 3

1. Write pseudo code to implement these two classes Singly Linked List and Doubly Linked List. Each of the classes has to include the following methods:

1. Insertion at the tail.
2. Deletion from the tail.
3. Insertion at the head.
4. Deletion from the head.

	Singly linked List	**Doubly linked List**

1	Algorithm addLast(Node n):
	n.next ←null
	if tail ≠ null
	   tail.next ← n
	   tail ←n
	else // tail = null
	   head ←n
	size ←size+1
		Algorithm addLast(Node n):
		n.prev ←tailer
		tailer.next ← n
		tailer ←n
		size ← size+1

2	Algorithm removeLast():
	temp ← head
	for i=0 → linkedlist.size-1
	   temp ← temp.next
	temp.next ← (temp.next).next
	size ← size - 1
		Algorithm removeLast():
		temp1 ← tailer.prev
		temp2 ← temp1.prev
		tailer.prev ← temp2
		temp2.next ← tailer
		temp1.prev ← null
		temp1.next ← null
		size ← size - 1

3	Algorithm addFirst(Node n):
	n.next ← head
	head ← n
	size ← size+1
		Algorithm addFirst(Node n):
		temp ← header.next
		n.next ← temp
		n.prev ← header
		header.next ← n
		size ← size+1

4	Algorithm removeFirst():
	If head = null
	   throw error
	else // head≠null
	   head ← head.next
	   size ← size - 1
		Algorithm removeFirst():
		header ← header.next
		header.prev ← null
		size ← size-1

2. Write the following algorithms to search a list for the occurrence of a node having certain data and return a reference to that node if found and null otherwise.

1. Recursive algorithm
2. Iterative algorithm

1. Recursive algorithm:

```
Algorithm findNode(Node n, Data d):
If n.data = d
    return n
else if n = null
    return null
return findNode(n.next, d)
```

2. Iterative algorithm

```
Algorithm findNode(Node n, Data d):
Node temp ← head
while(temp ≠ null)
    If temp.data = d
        Return temp
    else
        temp ← temp.next
return null
```

3. Write the following algorithms for a grounded linked list F1 having head pointing to the front node (Use these pseudocodes in your assignment implementation)

1. Insert a new node y at the front of the list
2. Insert a new node with data value val in a sorted list
3. Insert a new node as the kth node in the list
4. Append an element to the end of the list
5. Delete a node with value val from the list (first occurrence only)
6. Delete all occurrences of a node with value val from the list (write recursive and iterative algorithms)
7. Delete the node at the kth position in the list
8. Make a copy of F1; let F2 be a pointer to the first node of the new list (write the iterative and recursive algorithms)
9. Reverse the order of the nodes in F1 without creating any new node.
10. Test whether the elements in a list are ordered.
11. Interchange the first and last elements in a list.
12. Remove duplicates from the list (Assume F1 is sorted).

1	Algorithm addFirst(Node y):
	y.next ← head
	head ← y
	size ← size+1

2	Algorithm insertNode(Data val):
	Node tempNode
	tempNode.data ← val
	currentNode ← head
	while(currentNode.next ≠ null)
	    if tempNode.data < (currentNode.next).data
	        tempNode.next ← currentNode.next
	        currentNode.next ← tempNode
	        size ←size+1
	        return
	else //empty linked list
	    tempNode.next ←head
	    head ←tempNode
	    size ←size+1

3	Algorithm addToIndex(Node y, int k):
	currentNode ← head
	if k = 0 //insert first
	   y.next ← head
	   head ← y
	   size ← size + 1
	   return
	if currentNode ≠ null
	   for I = 0 → k-1
	      currentNode = currentNode.next
	y.next ← currentNode.next
	currentNode.next ← y
	size ← size + 1

4	Algorithm addLast(Node y):
	y.next ←null
	if head ≠ null
	   currentNode ← head
	   while (currentNode.next ≠ null)
	      currentNode ← currentNode.next
	   currentNode.next ← y
	else
	   head ←y
	size ←size+1

5	Algorithm delFirst(Data val):
	currentNode ← head
	if currentNode = null
	   return error list is empty
	while (currentNode.next.data≠ val)
	   currentNode ←currentNode.next
	   if currentNode = null
	      return error list is empty
	currentNode.next ← (cureentNode.next).next
	size ←size-1

6	Recursive	Algorithm delAll(Node currentNode, Data val):
		currentNode ← head
		if currentNode = null
		    return error list is empty
		if currentNode.next.data = val
		    currentNode.next ← (currentNode.next).next
		    size ←size-1
		    delAll(currentNode ,val)
		else delAll(currentNode.next ,val)

	Iterative	if head = null
		    return error list is empty
		while (head ≠ null and head.data = val)
		    head ← head.next
		    size ← size - 1
		currentNode ← head
		while (currentNode.next ≠ null)
		    if (currentNode.next).data = val
		        currentNode.next ← (cureentNode.next).next

7	Algorithm delAtIndex(int k):
	If k = 0
	    head ← head.next
	    size ← size-1
	    return
	Node currentNode ← head
	For I = 0 → k-1
	    currentNode ← currentNode.next
	currentNode.next = (currentNode.next).next;
	size ← size - 1;	

8	Recursive	Algorithm copyNode(Node x, Node y):
		If x = null
		    Return
		y.data ← x.data
		CopyNode(x.next, y.next)
		Algorithm copyList():
		If head = null
		    Return
		Node head2 ← F2.head
		Head2.data = head.data
		CopyNode(head.next, head2.next)

	Iterative	Algorithm copyList():
		If head = null
		    Return null
		Node head2 ← F2.head
		Head2.data = head.data
		Node tempNode1 ← head.next
		Node tempNode2 ← head2.next
		While(head.next ≠ null)
		    tempNode2 data ← tempNode1.data
		    tempNode1 ← tempNode1.next
		    tempNode2 ← tempNode2.next

9		Algorithm reverseList():
		If head = null
		    Return
		Let curr ← head
		Let prev ← null
		Let next ← null
		While (curr ≠ null)
		    next ← curr.next
		    curr.next ← prev
		    prev ← curr
		    curr ← next
		head ← prev

10	Algorithm isordered():
	If head = null
	   Return
	Node currentNode ← head
	While (currentNode.next ≠ null)
	   If (currentNode.next).val < currentNode.val
	      Return false
	   currentNode ← currentNode.next
	return true

11	Algorithm headToTail():
	If head = null
	   Return
	Node currentNode ← head
	Node tempNode
	While (currentNode.next ≠ null)
	   currentNode ← currentNode.next
	tempNode.data ← currentNode.data
	currentNode.data ← head.data
	head.data ← tempNode.data

12	Algorithm delDup():
	If head = null
	   Return
	Node currentNode ← head
	While (currentNode.next ≠ null)
	   If (currentNode.next).val = currentNode.val
	      currentNode.next ← (currentNode.next).next
	   else
	      currentNode ← currentNode.next

4. Consider the two grounded linked lists F1 and F2. Write algorithms for the following:

1. Testing F1 and F2 for equality; two lists are equal if they have the same length and they have the same data values in similar nodes.
2. Concatenating F2 to the end of F1.
3. Copying F1 to F2.

1	Node current1 = f1.head
	Node current2 = f2.head
	While(current1 $\neq$ null and current2$\neq$ null)
	    If current1.val $\neq$ current2.val // if not equal size value is compared with none
	        Return false
	    Current1 $\leftarrow$ current1.next
	    Current2 $\leftarrow$ current2.next
	Return true
	Algorithm length(linkedlist F):
	If f = null
	    Return 0
	Else
	    Return 1+length(F.next)

2	Algorithm concat():
	Node current1 $\leftarrow$ f1.head
	Node current2 $\leftarrow$ f2.head
	If current1 = null
	    Return f2
	If current2 = null
	    Return f1
	While(current1.next $\neq$ null)
	    Current1 $\leftarrow$ current1.next
	current1.next $\leftarrow$ current2

3	Algorithm copyList(): Node current1 $\leftarrow$ f1.head Node current2 $\leftarrow$ f2.head Current2 $\leftarrow$ current1

5. Assume F and R are references to the first and last node of a doubly linked list. Write algorithms to:

1. Delete the last element in the list.
2. Insert an element after the last element in the list.

1

```
Algorithm delLast():
If F = null
   Return
If F=R
   F=R=null
R  ← R.prev
R.next ← null
```

2

```
Algorithm addLast(Node n):
If F = null
   F ← n
   retuen
R.next ← n
n.prev ← R
R ← n
```