WIA1002/WIB1002 Data Structure

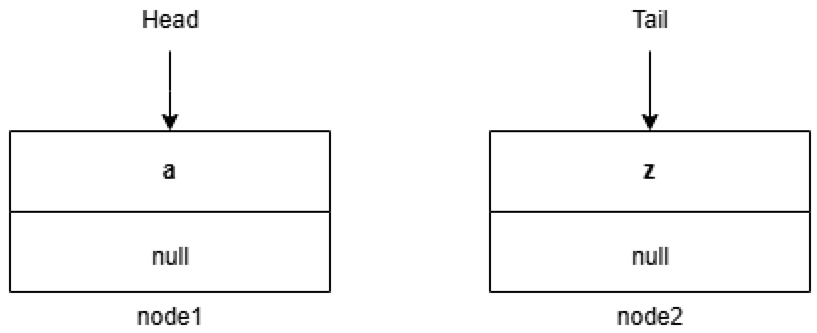
Tutorial: LinkedList

Question 1

a) Assume that a node class called Node<E> exist. Create two nodes called node1 and node2. Node1 contains alphabet ‘a’ and node2 contains alphabet ‘z’. Also, create 2 references, head and tail. Let head points to node 1 and tail points to node 2.

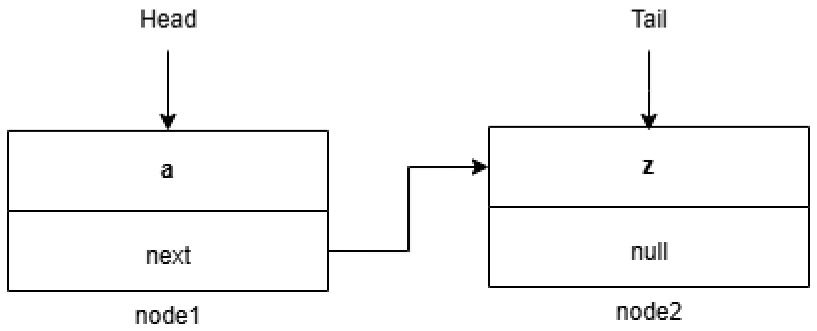
```
Node<Character> node1 = new Node<>('a');
Node<Character> node2 = new Node<>('z');

Node<Character> head = node1;
Node<Character> tail = node2;
```

b) Draw the nodes from (a).

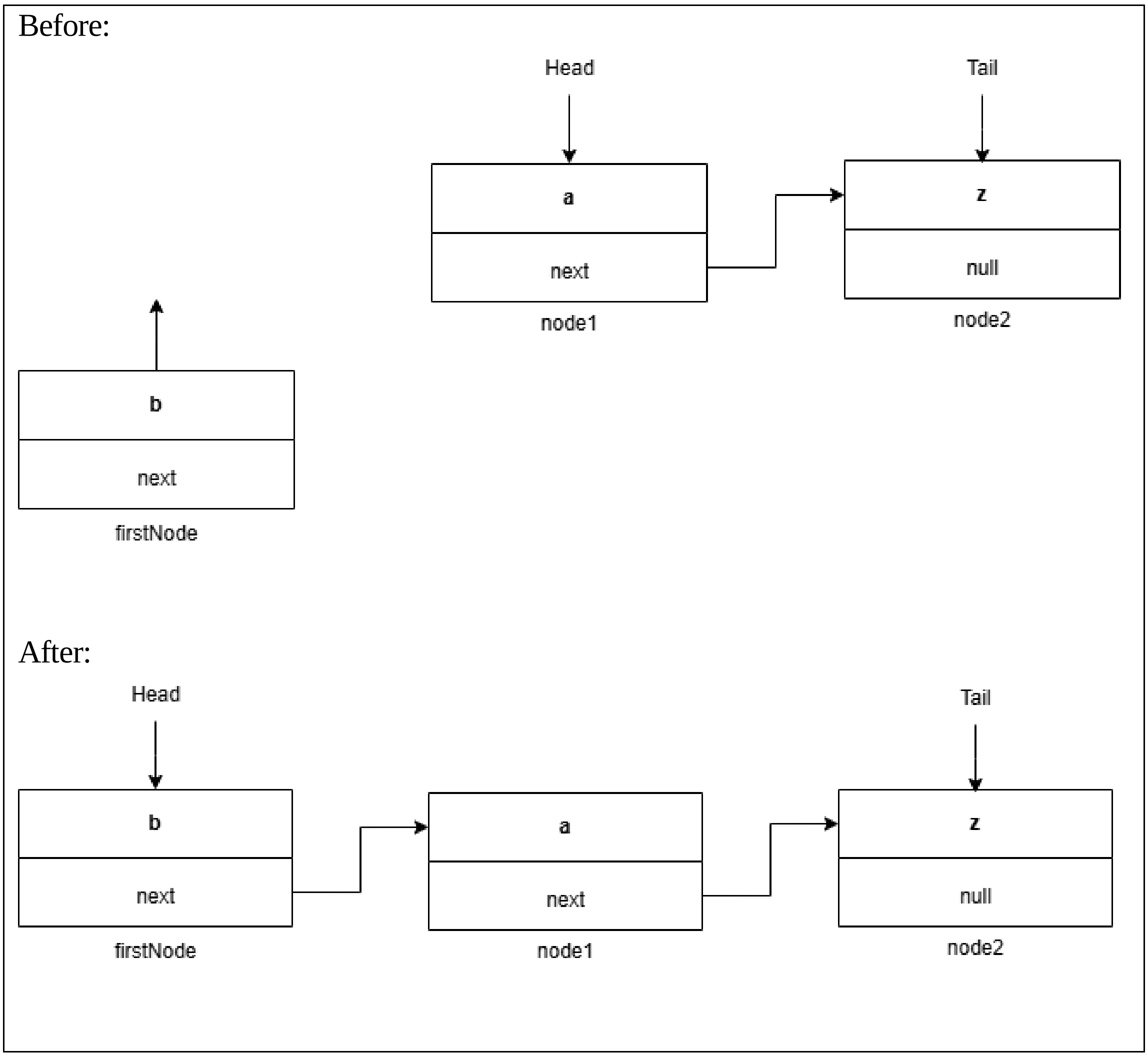


c) Write a statement/code for node1 accessing the node2. Modify 1(b) to show this.

```
node1.next = node2;
```



d) Create a new node, firstNode. Add this new node at the first location of all existing nodes. Draw these nodes.



e) If we have no information about the status of a linked-list, what are the conditions we need to consider to perform the operation in (d)?

- If the list is empty, inserting a new node means the head and tail will now point to the new node where node = head = tail. The new node's next is null.
- If the list has multiple nodes, the new node becomes the head and the new node's next reference points to the old head. The tail remains unchanged.
- Edge case: If the list has only 1 node, inserting a new node will set the new node's next to the old node and update head to the new node. The tail remains pointing at the old node.

f) Write a list of operations/steps/pseudocode needed to add the firstNode to the first location.

```
1. CREATE a new node named firstNode
2. SET the firstNode's next to current head
3. UPDATE head to point to the firstNode
4. INCREMENT size by 1
5. IF tail is equal to null
   5.1 UPDATE tail to the firstNode
6. END IF
```

g) Write codes to assign the firstNode to the first location.

```
public void addFirst(E e){
   Node<E> firstNode = new Node<>(e);
   firstNode.next = head; // create pointer to current head
   head = firstNode;
   size++; // increase size
   if (tail == null) { // no node exists
      tail = head;
   }
}
```

h) Repeat (d) – (f), for the following operations :

i. addLast() – value of element, c

ii. add(int index, E e) – value of element, d

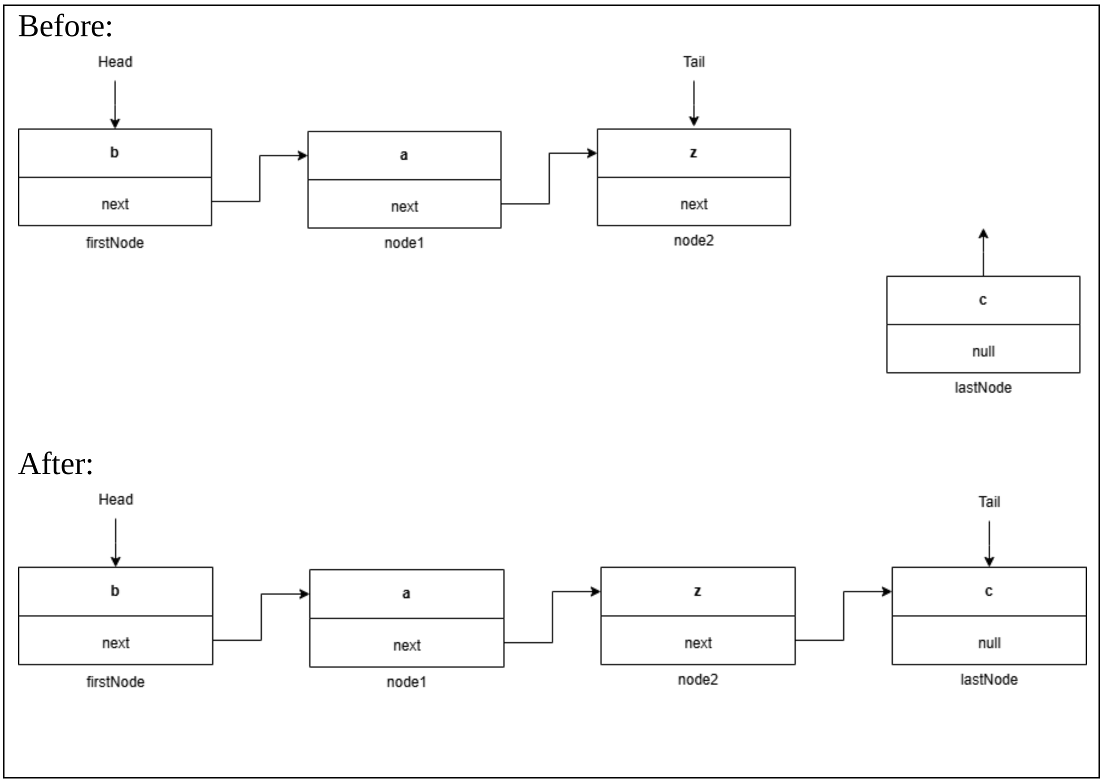
iii. removeFirst()

iv. removeLast()

v. remove(int index) – remove at index 1

i. addLast() – value of element, c

d)



e)

- If the list is empty, inserting a new node means the head and tail will now point to the new node where node = head = tail. The new node's next is null.
- If the list has multiple nodes, the new node becomes the tail and the old tail's next reference points to the new node. The head remains unchanged.
- Edge case: If the list has only 1 node, inserting a new node will set the old node's next to the new node and update tail to the new node. The new node's next remains null.

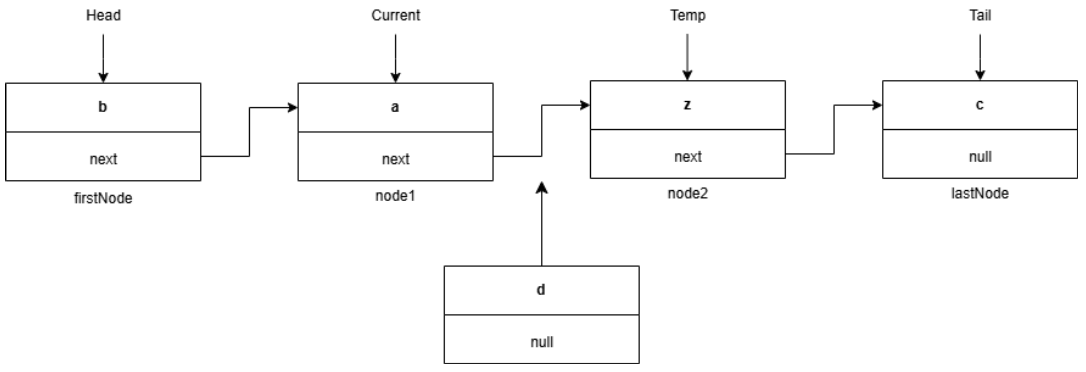
f)

```
1. CREATE a new node named lastNode with the value ‘c’
2. IF tail is equal to null
   2.1 SET head to lastNode
   2.2 SET tail to lastNode
3. ELSE
   3.1 SET tail’s next to lastNode
   3.2 SET tail to lastNode
4. INCREMENT size by 1
```

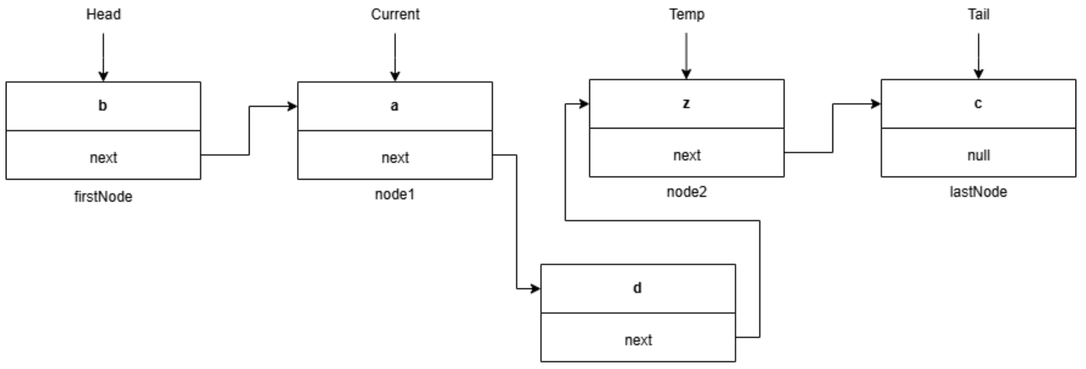
ii. add(int index, E e) – value of element, d

d)

Before:



After:



e)

- If the index is 0, call addFirst() method to insert the element at the beginning of the list.
- If the index is greater than or equal to size, call addLast() method to insert the element at the end of the list.
- Otherwise, create a new node to store the new element and locate where to insert it. The new node will be inserted between the nodes current and temp. The method assigns the new node to current.next and assigns the temp to the new node's next. Then, the size is increased by 1.

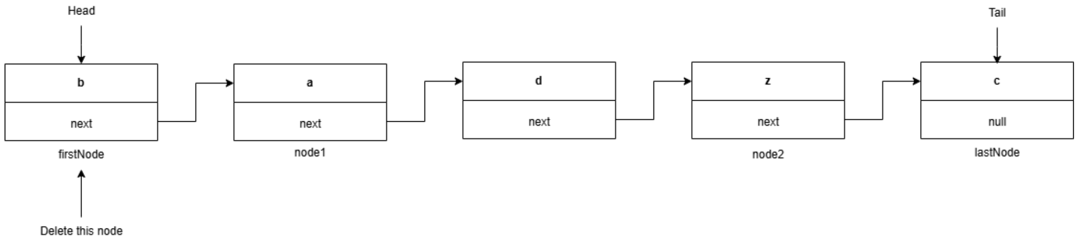
f)

```
1.  IF index is 0
    1.1 CALL addFirst(e) function
2.  ELSE IF index is greater than or equal to size
    2.1 CALL addLast(e) function
3.  ELSE
    3.1 CREATE a new node named current that points to head
    3.2 LET i = 1
    3.3 WHILE i is less than index
        3.3.1    SET current to the current's next
    3.4 CREATE a new node named temp with the value of 'd'
    3.5 SET current's next to the new node with the value of 'd'
    3.6 SET current.next's next to the temp
4.  END IF
5.  INCREMENT size by 1
```

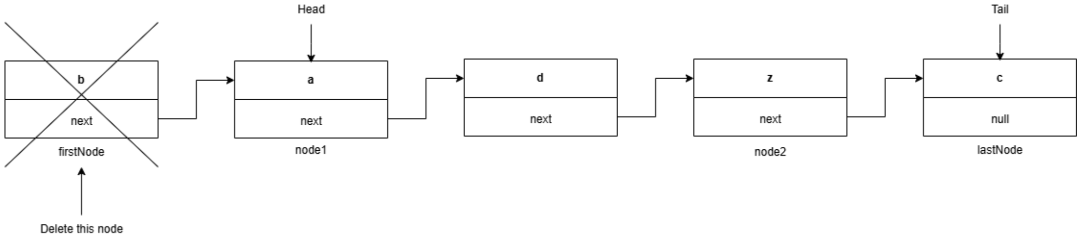
iii. removeFirst()

d)

Before:



After:



e)

- If the list is empty, there is nothing to delete, thus return null.
- Otherwise, remove the first node from the list by pointing head to second node. Then, the size is reduced by 1 after deletion. If the list becomes empty after removing the element, tail should be set to null as same as head.

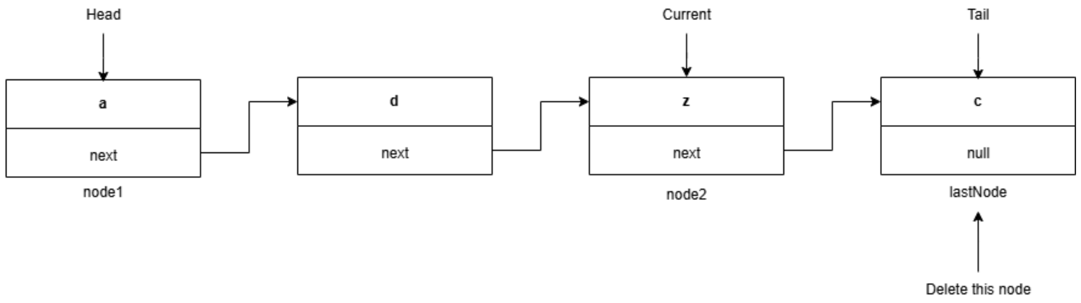
f)

```
1. IF size is equal to 0
   1.1 RETURN null
2. ELSE
   2.1 CREATE a new node named temp that points to head
   2.2 SET head to head's next
   2.3 DECREMENT size by 1
   2.4 IF head is equal to null
       2.4.1 SET tail to null
   2.5 RETURN temp.element
3. END IF
```

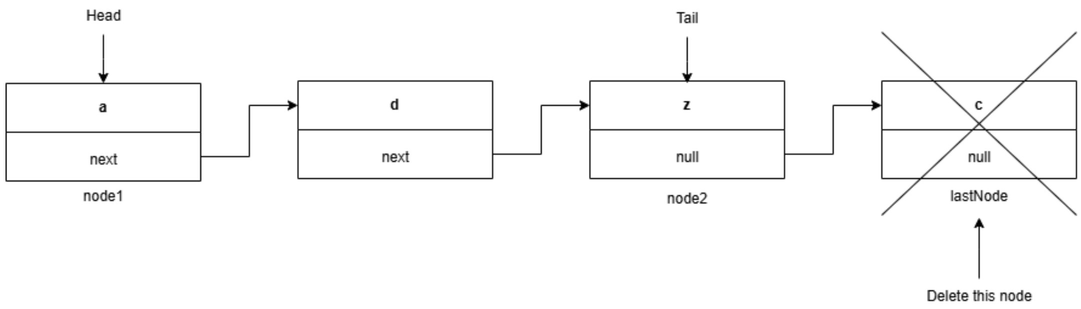
iv. removeLast()

d)

Before



After



e)

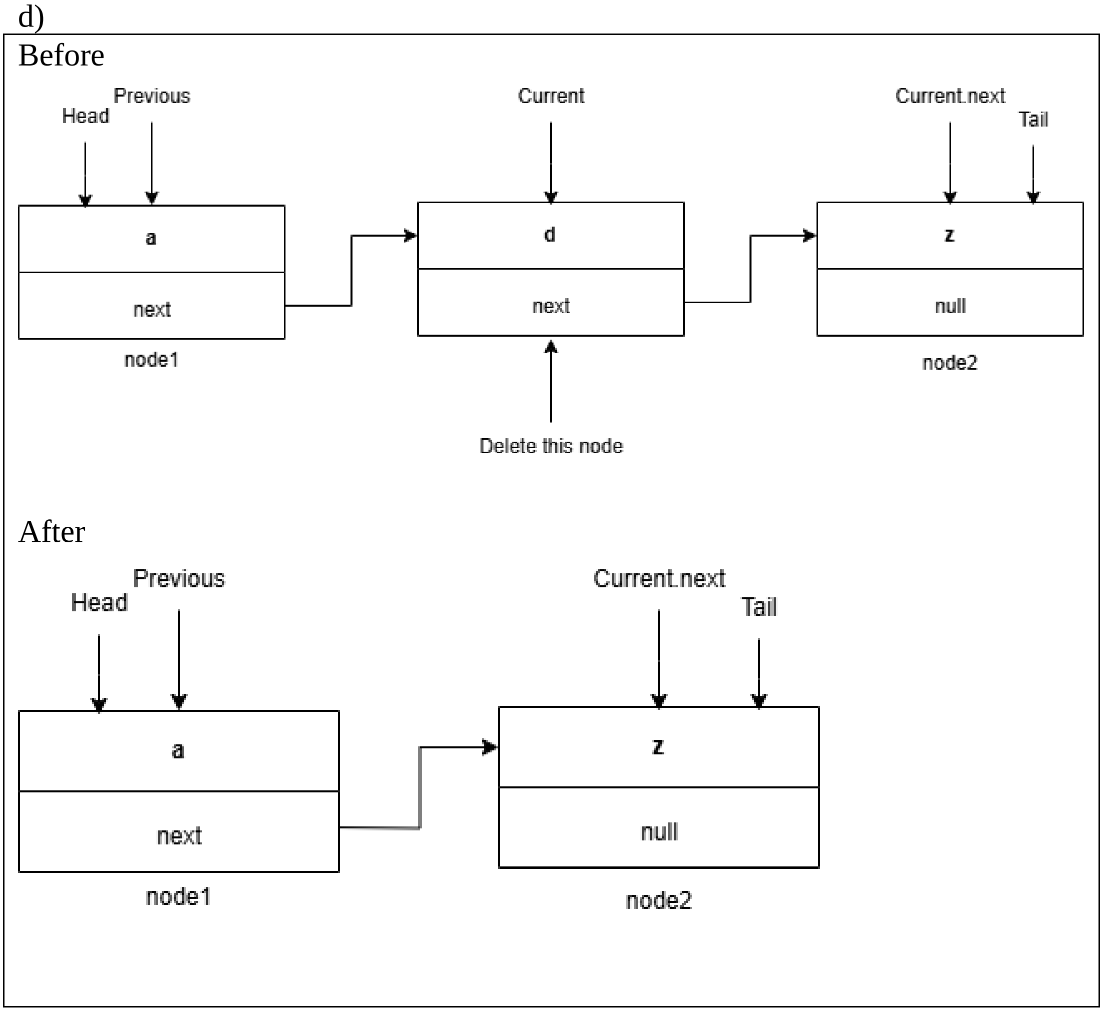
- If the list is empty or has a single element, call removeFirst() method to remove the element from the list.
- Otherwise, locate current to point to the second-to-last node. Save the value of tail to temp. Set tail to current. Tail is now repositioned to point to the second-to-last node and destroy the last node. Then, the size is reduced by 1 after the deletion. The element value of the deleted node is returned.

f)

```
1. IF size is equal to 0 OR size is equal to 1
   1.1 return removeFirst()
2. ELSE
   2.1 CREATE a new node named current that points to head
   2.2 LET i = 0
   2.3 WHILE i is less than size – 2
       2.3.1 SET current to current's next
   2.4 CREATE a new element named temp that assigns with tail's element
   2.5 SET tail to current
   2.6 SET tail's next to null
   2.7 DECREMENT SIZE BY 1
   2.8 RETURN temp
3. END IF
```

v. remove(int index) – remove at index 1

d)



e)

- If index is beyond the range of the list where index is less than 0 or index is greater than or equal to size, then return null.
- If index is 0, call removeFirst() to remove the first node
- If index is size-1, call removeLast() to remove the last node
- Otherwise, locate the node at the specified index. Assign current.next to previous.next to eliminate the current node where current is defined as this node and previous is defined as the node before this node.

f)

```
1.  IF index is less than 0 OR index is more than or equal to size
    1.1 RETURN null
2.  ELSE IF index is equal to 0
    2.1 RETURN removeFirst()
3.  ELSE
    3.1 CREATE a new node named previous that points to head
    3.2 LET i = 1
    3.3 WHILE i is less than index
        3.3.1    SET previous to previous's next
    3.4 CREATE a new node named current that points to previous's next
    3.5 SET previous's next points to current's next
    3.6 DECREMENT size by 1
    3.7 RETURN current's element
4.  END IF
```

Question 2

Given is a method containing incorrect statements that checks if an element is in a list.

```
public void operationX(E e) {
   pointerB.next = pointerB;

   for(int i++; i>size; int i) {
     System.out.println(current.element);
       if(current.element = e)
   }

   Node<E> pointerB = head;
   return false;
}
```

a) What is the name of the method for operationX?

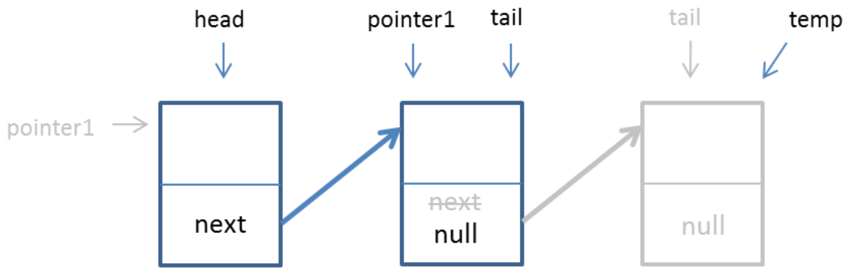
contains()

b) Correct the statements by rewriting them in the correct order and syntax.Write the correct/right method name to replace operationX.

```
public boolean contains(E e){
   Node<E> current = head;

   while (current != null) {
      if (current.element.equals(e)) {
         return true;
      }
      current = current.next;
   }

   return false;
}
```

Question 3

Given the following nodes. Answer the following:



a) Based on the above figure, what is the name of the method for above operation?
removeLast()

b) Write codes to represent the above figure. (Kindly use the variables stated in the figure)

```
public E removeLast(){
    if (size == 0){
        return null;
    }else if (size == 1){ // only 1 node
        Node<E> pointer1 = head;
        head = tail = null;
        // reset to know
        size = 0;
        return pointer1.element;
        // to know what we delete
    }else{
        Node<E> pointer1 = head;
        for (int i = 0; i < size - 2; i++) {
            pointer1 = pointer1.next;
        }
        // stop 1 node before tail
        Node<E> temp = tail;
        // copy tail to temp before delete
        tail = pointer1;
        // current become tail
        tail.next = null;
        // reset the next for tail to be null
        size--;
        return temp.element;
    }
}
```