Name: Nour Ahmed Mahmoud Khalil

ID: 23010913

DS Sheet 2

Question 1:

- Singly (Assume the list has a header node)

```
insertAtTail(Node V)
{
    V.setNext(null)
    Node T <- head
    While (T.getNext ≠ null)
            T <- T.getNext()
    T.setNext(V)

}

deleteLast()

{

    Node T <- head
    Node U <- T.getnext()
    If T = null
            Indicate an error, list is empty
            return
    While (U.getNext ≠ null)
            U <- U.getNext()
            T <- T.getNext()
    T.setNext(null)

}

insertAtHead(Node V)

{

    V.setNext(head.getNext)

    head.setNext(V)

}
```

```
deleteFirst()
{
        If head.getNext() = null
                Error, list is empty
                Return
        Node T <- head.getNext()
        Head.setNext(T.getNext())
        T.setNext(null)
}
```

- Doubly(Assume header and trailer)

```
insertAtTail(Node V)
{
        Node U <- trailer.getPrev()
        U.setNext(V)
        V.setPrev(U)
        V.setNext(trailer)
        Trailer.setPrev(V)
}
deleteLast()
{
        If header.getNext() = trailer
                Error, empty list
                Return
        Node U <- trailer.getPrev()
        Node V <- U.getPrev()
        V.setNext(trailer)
        trailer.setPrev(V)
        U.setNext(null)
```

```
        U.setPrev(null)
}
insertAtHead(Node V)
{
        Node U <- header.getNext()
        V.setNext(U)
        V.setPrev(header)
        header.setNext(V)
        U.setPrev(V)
}
deleteFirst
{
        If header.getNext() = trailer
                Error, empty list
                Return
        Node U <- header.getNext()
        Node V <- U.getNext()
        Header.setNext(V)
        V.setPrev(header)
        U.setNext(null)
U.setPrev(null)
}
```

Question 2:

- Recursive

```
search(data d)
{
    If head = null
            Return not found
     if head.getData() = d
            return head
    head <- head.getNext()
    return search(d)
}
```

- Iterative

```
search(data d)
{
    While head ≠ null
            If head.getData = d
                    Return head
            head <- head.getNext()
    Return not found
}
```

Question 3:

1. insertAtHead(Node Y)

```
{
        Node U <- head.getNext()
        Y.setNext(U)
        Y.setPrev(head)
        Head.setNext(Y)
        U.setPrev(Y)
}
```

2. insertInSorted(Node Y)

```
{
        Data data = Y.getData()
        Node p <- head
        Node q <- head.getNext()
        While q ≠ null and q.getData < data
                q <- q.getNext()
                p <- p.getNext()
        p.setNext(Y)
        Y.setNext(q)
}
```

3. insertAtKth(int K, Node Y)

```
{
        Node q <- head
        Node p <- head
        For (int i=0; i<k; i++)
                p <- q
                q  <- q.getNext()
        p.setNext(Y)
        Y.setNext(q)
}
```

4. append(Data element)

```
{
        Node q <- head
        Node n <- createNode(element)
        While q.getNext ≠ null
                q <- q.getNext()
        q.setNext(n)
        n.setNext(null)
}
```

5. deleteVal(Data Val)
```
{
        Node q <- head.getNext()
        Node p <- head
        While q ≠ null
                If q.getData()  = Val
                        p.setNext(q.getNext)
                        q.setNext(null)
                        return
                q <- q.getNext()
                p <- p.getNext()
        error, value not found
}
```
6. deleteAllOccurences(val)
```
{
        Node q <- head.getNext()
        Node p <- head
        While q ≠ null
                If q.getData()  = Val
                        p.setNext(q)
                        q.setNext(q.getNext())
                q <- q.getNext()
                p <- p.getNext()
        error, value not found
}
```
7. deleteK(int K)
```
{
        Node q <- head
        Node p <- head
        For (int i=0; i<k; i++)
                p <- q
                q <- q.getNext()
        if q.getNext = null
                error, k out of bound
                return
        p.setNext(q.getNext())
        q.setNext(null)
}
```

8. makeCopy()
```
{
        Node p <- F1.getNext()
        Node q <- F2
        While p ≠ null
                Node newNode <- createNode(p.getElement())
                q.setNext(newNode)
                q <- newNode
                p <- p.getNext()
}
```

9. reverseOrder(node a, node b)
```
{
        if b = null
                return a
        node c = b.getNext()
        b.setNext(a)
        return reverseOrder(b, c)
}
```
10. testOrder()
```
{
        Node p <- head
        Node q <- head.getNext()
        While q ≠ null
                If q.getData() < p.getData()
                        Return not ordered
                p <- p.getNext()
                q <- q.getNext()
        Return ordered
}
```
11. firstLast()
```
{
        Node q <- head.getNext()
        If q or q.getNext = null
                Error
        Val = q.getElement()
```

```
        While q.getNext() ≠ null
            q <- q.getNext()
        val2 = q.getElement()
        q.setElement(val)
        head.getNext().setElement(val2)
    }
```

12. removeDuplicates()

```
    {
        Node q <- head.getNext()
        If q = null
            Error, empty list
            Return
        Node p <- q.getNext()
        While p ≠ null
            If q.getElement() = p.getElement()
                q.setNext(p.getNext())
            q <- q.getNext()
            p <- p.getNext()
    }
```

Question 4:

1. testEquality()

```
    {
        Node p <- head1.getNext()
        Node q <- head2.getNext()
        While p and q ≠ null
            If q.getData() ≠ p.getData()
                Return not equal
            p <- p.getNext()
            q <- q.getNext()

        if p or q = null
            return not equal
        return equal
    }
```

2. concatenate()

```
    {
        p <- head1
```

```
        q <- head2
        while p.getNext() ≠ null
            p <- p.getNext()
        p.setNext(q.setNext)
}
```

3. copy()

```
{
        Node p <- F1.getNext()
        Node q <- F2
        While p ≠ null
            Node newNode <- createNode(p.getElement())
            q.setNext(newNode)
            q <- newNode
            p <- p.getNext()
}
```

Question 5:

1. deleteLast()

```
{
        If R = null
            Return error, list is empty
        Node U <- R
        R <- R.getPrev()
        If R = null
            F = null
            return
        R.setNext(null)
        U.setPrev(null)
}
```

2. insertLast(Node V)

```
{
        V.setNext(null)
        If F = null
            F <- V
            R <- V
            Return
        R.setNext(V)
```

```
        V.setPrev(R)
        R <- V
}
```