Data Structures Assignment 1 - Ruth Dooley 19300753

Question 1

3 Real World Uses for Circularly Linked Lists

- Queuing (Circularly) → A circularly linked list queue is similar to a singularly linked list queue other than that the last node of the singularity linked lists queue points to nothing compared the circularly linked list queue whose final node leads to the first node. It is a queue that works on a first in and first out basis.

Benefits: Only one pointer is required. Do not need to maintain two pointers one at the front and one at the rear, the front node can always be accessed from the rear node. One pointer need only be maintained to store the last inserted node marking rear and front and therefore can easily be found. Using a regular linked list implementation with a pair of pointers makes the program more susceptible to bugs, such as form pointer or reference inconsistency, and this risk can be eliminated using this technique.

- Personal computer → Circular linked lists are useful in the personal computer when multiple applications are running. The applications are given a node in the linked list and each are allotted a time slot to allow it to run. This distributes the time required for each of the programs. The operating system continues to iterate over the linked list until all applications are complete.

Benefits: A continuous queue can be maintained and a continuous loop can be run. This is useful for the allocation of resources that must be kept up continuously until completed. Using this method various other application queuing can also be implemented such as priority queue, rather than the typical waiting queue, all using this data structure.

- Multiplayer Games → In a multiplayer game a circular linked list can be used to represent the players in the game. The pointer iterates over the list, moving when one player's turn ends continuing until instructed. This allows the game to traverse the node indefinitely, which is uncommon for other data structures.

Benefits: Easily increment over the players in a continuous loop, when the last player is reached the loop begins repeats again. Any node/player can be the first in the loop, as they are all lined up and linked there is no endpoint or start point

*Highlighted yellow throughout are the base cases

Question 2

```
Node class:
    data
    node next

    constructor (data)
        data = data
        next = null

Question 2 class:
    Node head

    reverseNumber (node)
        if (node or node.next is empty)
            return node
        temp = reverseNumber (node.next)
        node.next.next is equal to original node
        node.next is equal to empty
        return temp

    actualAdditioFunc (node1, node2)
        carryIn is equal to 0

        headNode is equal to null
        tempNode is equal to null
        tempSum is equal null
        newNode is equal null
        total is equal to 0
        iterator1 is equal to 0

        while (either node1 or node2 is not empty)
            iterator1 ++
            total is equal carryIn

            if (node1 is not empty)
                sum is equal itself plus the data of node1
                node1 is equal the next node

            if (node2 is not empty)
                sum is equal itself plus the data of node2
                node2 is equal the next node

            carryIn is equal to sum divided by 10
            sum is equal to itself modulus 10

            if (iterator1 is equal 1)
                tempNode data is equal to the sum
                headNode is equal to temp node
            else
                tempSum data is equal to the sum
                tempNode.next is equal to tempSum
                tempNode is equal to tempNode.next


        if (carry is not equal to 0)
            newNode is equal to carryIn
            tempNode.next is equal to new node

        return headNode
```

*Have to reverse each number first then perform addition function then reverse the answer

Question 3

```
Node class:
    data
    node next
    node child

    constructor (data, next)
        data = data
        next and child = null

Question 3 class:

//Helper method create list
newList (array, number){
    node = null
    temp = null;

    for (i = 0; i < n; ++i){
        if (node is null)
            node is equal to temp is equal to a new node and the data is array[i]
        else
            temp.next is equal to a new node and the data is array[i]
            temp is equal to temp.next

        temp.next is equal to temp.child is equal to null
    }
    return node
}

flattenList (node){
    tempNode = null
    if (node is null)
        return

    last = node
    while (last.next is not equal null)
        last is equal last.next

    current = node
    while (current is not equal to last)
        if (current has a child)
            last.next is equal current.child
            tempNode = current.child
            while (tempNode.next is not equal to null)
                tempNode is equal to tempNode.next
            last = tempNode
    current = current.next
}
```

```
//How to make tiered list like in the question
listToFlatten (){
          int arr1[] = new int[]{9, 8, 4, 7, 13};
    int arr2[] = new int[]{5, 12, 3};
    int arr3[] = new int[]{16, 21};
    int arr4[] = new int[]{10};
    int arr5[] = new int[]{14};
    int arr6[] = new int[]{23, 41};
    int arr7[] = new int[]{9};
    int arr8[] = new int[]{34, 30};

    Node head1 = createList(arr1, arr1.length);
    Node head2 = createList(arr2, arr2.length);
    Node head3 = createList(arr3, arr3.length);
    Node head4 = createList(arr4, arr4.length);
    Node head5 = createList(arr5, arr5.length);
    Node head6 = createList(arr6, arr6.length);
    Node head7 = createList(arr7, arr7.length);
    Node head8 = createList(arr8, arr8.length);

    head1.child = head2;
    head1.next.next.next.child = head3;
    head2.next.child = head4;
    head2.next.next.child = head5;
    head3.child = head6;
    head5.child = head7;
    head6.child = head8;

    return head1;
}
```

```
Main:
        Make a new linked list newList
        Node head is equal newList.listToFlatten()
        newList.flattenList(head)
        Print list to console
```

Question 4

Question 4 class:

```
class Node
        data
        Node left, right

nodeData (data)
        node = new node
```

```
    node left and right equal null
    node data equal data
    return node

Node last = null

flattenBinaryTree (node)
    Node left, right
    if (node is not equal null)
        return

    left equal node.left
    right equal node.right

    if (node is not equal last)
        last.right is equal node
        last.left is equal to null
        last is eqaul node

    flattenBinaryTree (left)
    flattenBinaryTree (right)

    if (left is null and right is null)
        last is the root

Main:
    //Set up binary tree like this:
    Node root = AllocNode(1);
    root.left = AllocNode(2);
    root.left.left = AllocNode(3);
    root.left.right = AllocNode(4);
    root.right = AllocNode(5);
    root.right.right = AllocNode(6);
    etc..

    last = root;
    flattenBinaryTree(root);
```

Question 5

```
Node class:
    data
    Node left, right, nextRight
    Node(int item)
  data = item;
  left = right = null;

Question 5 class:
    sum ()
```

```
        if (node is null)
            return 0
        return sum (node.left) + node.data + sum (node.right)

    isDiskUsage (node)
        left, right

        if (node is null or node left and node right are null)
                return true

        left = sum (node.left)
        right  = sum (node.right)

         if (node.data is equal left + right  and isDiskUsage (node.left) not equal 0 and
    isDiskUsage (node.right) not equal 0)
            return 1;
         return 0;

    Main:
        //Set up like
        Question6 diskUsageTree = new Question6()
        tree.root=new Node(12);
        tree.root.left=new Node(1);
        tree.root.right=new Node(7);
        tree.root.left.left=new Node(4);
        Etc….

        If (isDiskUsage (node)) the tree is a disk usage tree
        Else the tree is not a disk usage tree
```

Question 6

```
Node class:
    data
    Node left null
    Node right null

    Node (data)
        data = data

Question 6 class:
nodeData
    data
    level
    Node parent is null

    nodeData (data, level, parent)
```

```
        data = data
        level = level
        parent = parent

order (root, parent, level, nodeData i, nodeData j)
    if (root is null) return

    order (root.left, root, level + 1, i, j)

    if (root.data is equal i.data)
        i.level = level;
        i.parent = parent

    if (root.data is equal j.data)
        j.level = level;
        j.parent = parent

    order (root.right, root, level + 1, i, j)

areCousins (root, one, two)
    if (root is null) return false

    level = 1
    parent = null

    nodeData x = new nodeData(one, level, parent);
    nodeData y = new nodeData(two, level, parent);

 order (root, null, 1, x, y)

 if (x.level is not equal y.level or x.parent is equal y parent)
    //Not cousins
    return false

 return true

Main:
    //Set up as following
    Node root = new Node('a');
    root.left = new Node('b');
    root.right = new Node('e');
     root.left.left = new Node('d');
     root.left.right = new Node('c');
     root.right.right = new Node('f');

    If (areCousins(root, one, two)) two nodes are cousins
    Else two nodes are not cousins
```