CSC 310 Data Structures

Homework 5:

CSC310_Hw5_1:

```
class LinkedStack{
    private String stackName;
    private LinkedList<Integer> l;
    private int n;

    LinkedStack(String name){
        stackName=name;
        l=new LinkedList<>();
        n =0;
    }

    public void push(int num){
        l.add(num);
        n++;
    }

    public int size(){
        return n;
    }

    public int top(){
        return l.get(n-1);
    }

    public int pop(){
        if(n>=0){
            int out = top();
            l.remove(n-1);
            n--;
            return out;
        }else
            return -1;
    }

    public String name(){
        return stackName;
    }
}
```

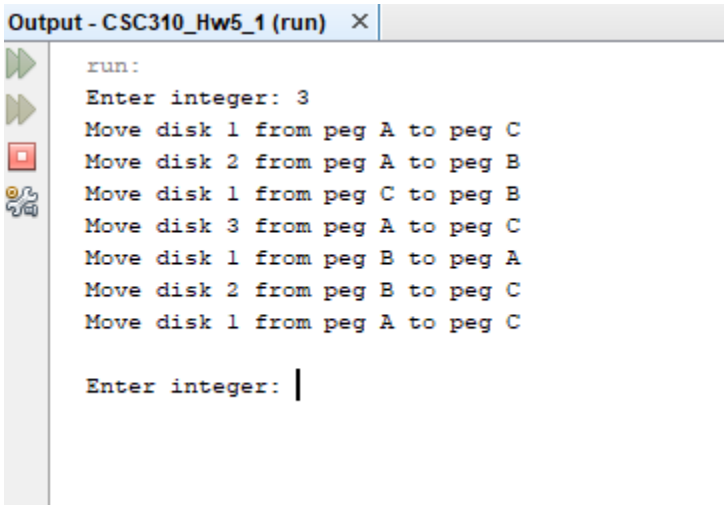
I decide to use a LinkedStack similar to previous Homework. It has basic LinkedStack functions.

```java
public class CSC310_Hw5_1 {
    static void towerOfHanoi(int n, LinkedStack first,LinkedStack mid,LinkedStack last){
        if(n==1){
            System.out.println("Move disk "+first.top()+" from peg "+first.name()+" to peg "+last.name());
            last.push(first.pop());
            return;
        }

        towerOfHanoi(n-1,first, last, mid);

        System.out.println("Move disk "+first.top()+" from peg "+first.name()+" to peg "+last.name());
        last.push(first.pop());

        towerOfHanoi(n-1,mid, first, last);
    }
```

This function is the tower of Hanoi that prints out the moves needed to solve the puzzle. This works must like a branching problem. Basically you move one on the mid move the next larger one to the last and basically using the mid as a transition point.

```java
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    while(true){
        try{
            LinkedStack a= new LinkedStack("A");
            LinkedStack b= new LinkedStack("B");
            LinkedStack c= new LinkedStack("C");

            System.out.print("Enter integer: ");
            int num = input.nextInt();

            for (int i = num; i > 0; i--) {
                a.push(i);
            }

            towerOfHanoi(num, a, b, c);
            System.out.println("");

        }catch(Exception e){
            System.out.println("Goodbye!");
            break;
        }
    }
}
```

I had to set up three stacks to act as the “pegs” I have to prompt the user for how many discs and then stack them in the first peg. I used the while true try catch comb to catch exceptions and let the user end the program.



CSC310_Hw5_2:

```
3   class Node{
4       int value;
5       Node parent;
6       Node left;
7       Node right;
8
9       public Node(int x){
10          value=x;
11          parent=null;
12          right=null;
13          right=null;
14      }
15  }
16
17  class Tree{
18      Node root;
19
20      Tree(){
21          root=null;
22      }
23
24      public void inOrder(Node t){
25          if (t==null)
26              return;
27          inOrder(t.left);
28          System.out.print(t.value+" ");
29          inOrder(t.right);
30      }
31
32      public void preOrder(Node t){
33          if (t==null)
34              return;
35          System.out.print(t.value+" ");
36          preOrder(t.left);
37          preOrder(t.right);
38      }
39
40      public void postOrder(Node t){
41          if (t==null)
42              return;
43          inOrder(t.left);
44          inOrder(t.right);
45          System.out.print(t.value+" ");
46      }
47  }
```

I had to set up a node class with all of the components needed to transition up and down the tree.

I set up a tree class that would be needed to keep track of the root of the tree.

In order I had to first get the lefts then the middles then the rights of the tree and it would need to be put in this order.

In order I had to first get the middle then the lefts then the rights of the tree and it would need to be put in this order.

In order I had to first get the lefts then the rights then the middles of the tree and it would need to be put in this order.

I Decided to add this for it to be ore expandable in the future and to test the program better. Plus it wasn't too hard to change since it's just the order of which that needed to be changed around.

```
50      static void toTree(Node current, LinkedList<Integer> list, int n){
51          int left=n*2+1;
52          int right=n*2+2;
53
54
55          if(left<=list.size() && list.get(left)!=null){
56              Node leftN = new Node(list.get(left));
57              current.left=leftN;
58              toTree(leftN, list, left);
59          }
60
61          if(right<list.size() && list.get(right)!=null){
62              Node rightN = new Node(list.get(right));
63              current.right=rightN;
64              toTree(rightN, list, right);
65          }
66      }
67
68
69      public static void main(String[] args) {
```

This is where I turned the list into the tree in a recursion very similar to the tree class. I had to calculate the children then call the children of the class if they aren't over the list size and if its not null.

```
49  public class CSC310_Hw5_2 {
50      public static void main(String[] args) {
51          Scanner input = new Scanner(System.in);
52          LinkedList<String> list = new LinkedList<String>();
53
54          while(true){
55              try{
56                  System.out.print("Enter integer: ");
57                  String s =input.nextLine();
58                  if (!s.equals("null"))
59                      Integer.parseInt(s);
60                  list.add(s);
61
62              }catch(Exception e){
63                  break;
64              }
65          }
```

I decided to use a linked list to store the user input into so that it was easier to store and I wouldn't need to resize an array every time to keep it dynamic. I decided to use the while true try catch combination to let the user input the read in information from the user and let the user stop input by entering input that would be considered invalid.

```
92          System.out.print("Inorder: ");
93          t.inOrder(t.root);
94          System.out.println("");
95          System.out.print("Preorder: ");
96          t.preOrder(t.root);
97          System.out.println("");
98      }
```

I then had to call the functions of the trees to get the inorder and preorder traversals of the tree for the user.

```
Output - CSC310_Hw5_2 (run)
run:
Enter integer: 1
Enter integer: null
Enter integer: 2
Enter integer: 3
Enter integer: asdf
Inorder: 1 3 2
Preorder: 1 2 3
BUILD SUCCESSFUL (total time: 16 seconds)
```