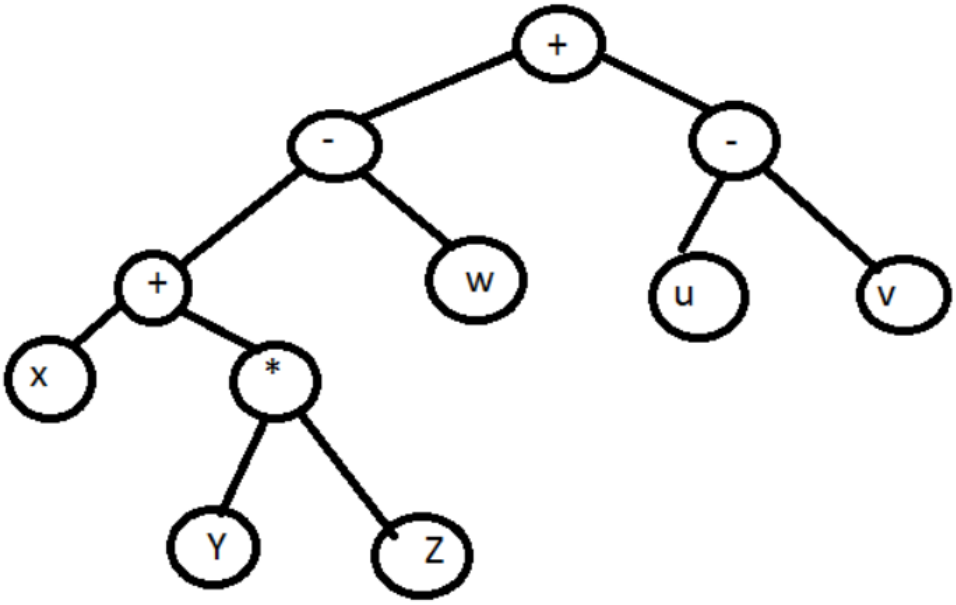
S Abhishek AM.EN.U4CSE19147

Data Structures

Consider the ADT provided in the attached file for Binary Tree data structure. Add following functions.

1. Build a tree for the expression (x+(y*z)-w) + (u-v) as follows




```
class Node:

    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right

def isop(op):
    return op == '+' or op == '-' or op == '*' or op == '/'

def inorder(head):
    if head is None:
        return
```

```
    if isop(head.data):
        print("(", end=' ')

    inorder(head.left)
    print(head.data, end=' ')
    inorder(head.right)

    if isop(head.data):
        print(")", end=' ')

def tree(exp_value):
    global node
    s = []

    for i in exp_value:
        if isop(i):
            x = s.pop()
            y = s.pop()
            node = Node(i, y, x)
            s.append(node)
        elif i == " ":
            continue
        else:
            s.append(Node(i))

    return node

if __name__ == '__main__':
    exp = input("Enter the Postfix Expression : ")
    root = tree(exp)

    print("Arithmatic Expression : ",end=" ")
    inorder(root)
```

```
Enter the Postfix Expression : x y z * + w - u v - +
Arithmatic Expression :  ( ( ( x + ( y * z ) ) - w ) + ( u - v ) )
```

2. Replace x=3;y=2;z=1;w=6; u = 5;v=4; in the tree built in (a). Write a function to print the following arithmetic expression.

(3+(2*1)-6) + (5-4)

```
class Node:

    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right

def isop(op):
    return op == '+' or op == '-' or op == '*' or op == '/'

def inorder(head):
    if head is None:
        return
    if isop(head.data):
        print("(", end=' ')

    inorder(head.left)
    print(head.data, end=' ')
    inorder(head.right)

    if isop(head.data):
        print(")", end=' ')

def tree(exp_value):
    global node
    s = []

    for i in exp_value:
        if isop(i):
            x = s.pop()
            y = s.pop()
            node = Node(i, y, x)
```

```
            s.append(node)
        elif i == " ":
            continue
        else:
            s.append(Node(i))

    return node

if __name__ == '__main__':
    exp = input("Enter the Postfix Expression : ")
    root = tree(exp)

    print("Arithmatic Expression : ",end=" ")
    inorder(root)
```

```
Enter the Postfix Expression : 3 2 1 * + 6 - 5 4 - +
Arithmatic Expression :  ( ( ( 3 + ( 2 * 1 ) ) - 6 ) + ( 5 - 4 ) )
```

3. Given tree T, Check if the Tree is Proper Binary Tree.

```
class Node:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right

def full_or_not(head):
    if head is None:
        return True

    elif head.left is None and head.right is None:
        return True

    elif head.left is not None and head.right is not None:
```

```
        return full_or_not(head.left) and full_or_not(head.right)

    else:
        return False


if __name__ == '__main__':
    root  = Node(1)
    nod1 = Node(2)
    nod2 = Node(3)

    root.left = nod1
    root.right = nod2

    nod3 = Node(4)
    nod4 = Node(5)

    nod2.left = nod3
    nod2.right = nod4

    if full_or_not(root):
        print("It is a FULL Binary tree")
    else:
        print("It is not a FULL Binary tree")
```

```
It is a FULL Binary tree
```

```
class Node:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right
```

```
def full_or_not(head):
    if head is None:
        return True

    elif head.left is None and head.right is None:
        return True

    elif head.left is not None and head.right is not None:
        return full_or_not(head.left) and full_or_not(head.right)

    else:
        return False


if __name__ == '__main__':
    root  = Node(1)
    nod1 = Node(2)
    nod2 = Node(3)

    root.left = nod1
    root.right = nod2

    nod3 = Node(4)
    nod4 = Node(5)

    nod2.left = nod3
    #nod2.right = nod4

    if full_or_not(root):
        print("It is a FULL Binary tree")
    else:
        print("It is not a FULL Binary tree")
```

```
It is not a FULL Binary tree
```

4. Find the depth of a node in the tree having value ‘x’. If such a node does not exist throw an exception.

```
class Node:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right


def get_node(node, data, depth):
    if node is None:
        return 0

    if node.data == data:
        return depth

    node_pos = get_node(node.left, data, depth + 1)

    if node_pos != 0:
        return node_pos

    node_pos = get_node(node.right, data, depth + 1)
    return node_pos

if __name__ == '__main__':

    root = Node(1)
    nod1 = Node(2)
    nod2 = Node(3)

    root.left = nod1
    root.right = nod2

    nod3 = Node(4)
```

```
  nod4 = Node(5)

  nod1.left = nod3
  nod1.right = nod4

  nod5 = Node(6)
  nod6 = Node(7)

  nod2.left = nod5
  nod2.right = nod6

  ele = int(input("Enter the Element to Find : "))

  if get_node(root, ele, 1):
      print("Depth of", ele, "is", get_node(root, ele, 0))
  else:
      raise Exception("No such Node!")
```

```
Enter the Element to Find : 1
Depth of 1 is 0
```

```
Enter the Element to Find : 4
Depth of 4 is 2
```

```
Enter the Element to Find : 9
Traceback (most recent call last):
  File "G:\Data Structures\Lab Assignments\Lab 5\Depth.py", line 50, in <module>
    raise Exception("No such Node!")
Exception: No such Node!
```

```
Enter the Element to Find : 8
Traceback (most recent call last):
  File "G:\Data Structures\Lab Assignments\Lab 5\Depth.py", line 50, in <module>
    raise Exception("No such Node!")
Exception: No such Node!
```

5. Build a tree with following properties.

   a. Elements stored in the left subtree of a Node p are less than the value in Node p.
   b. Elements stored in the right subtree of a Node p are less than the value in Node p.

```
class Node:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right


def Inorder(root):
    if root:
        Inorder(root.left)
        print(root.data, end=" ")
        Inorder(root.right)


def Postorder(root):
    if root:
        Postorder(root.left)
        Postorder(root.right)
        print(root.data, end=" ")


def Preorder(root):
    if root:
        print(root.data, end=" ")
        Preorder(root.left)
        Preorder(root.right)


if __name__ == '__main__':
    head = Node(10)
```

```
nod1 = Node(8)
nod2 = Node(9)

head.left = nod1
head.right = nod2

nod3 = Node(6)
nod4 = Node(7)

nod1.left = nod3
nod1.right = nod4

nod5 = Node(4)
nod6 = Node(5)

nod2.left = nod5
nod2.right = nod6

nod7 = Node(2)
nod8 = Node(3)

nod3.left = nod7
nod3.right = nod8

nod9 = Node(0)
nod10 = Node(1)

nod6.left = nod9
nod6.right = nod10

print("Post Order Traversal LRN : ", end=" ")
Postorder(head)
print("\nPre Order Traversal NLR : ", end=" ")
Preorder(head)
print("\nInorder Order Traversal LNR : ", end=" ")
Inorder(head)
```

```
Post Order Traversal LRN :  2 3 6 7 8 4 0 1 5 9 10
Pre Order Traversal NLR :  10 8 6 2 3 7 9 4 5 0 1
Inorder Order Traversal LNR :  2 6 3 8 7 10 4 9 0 5 1
```

One Drive : Click Me!!

Thankyou!!