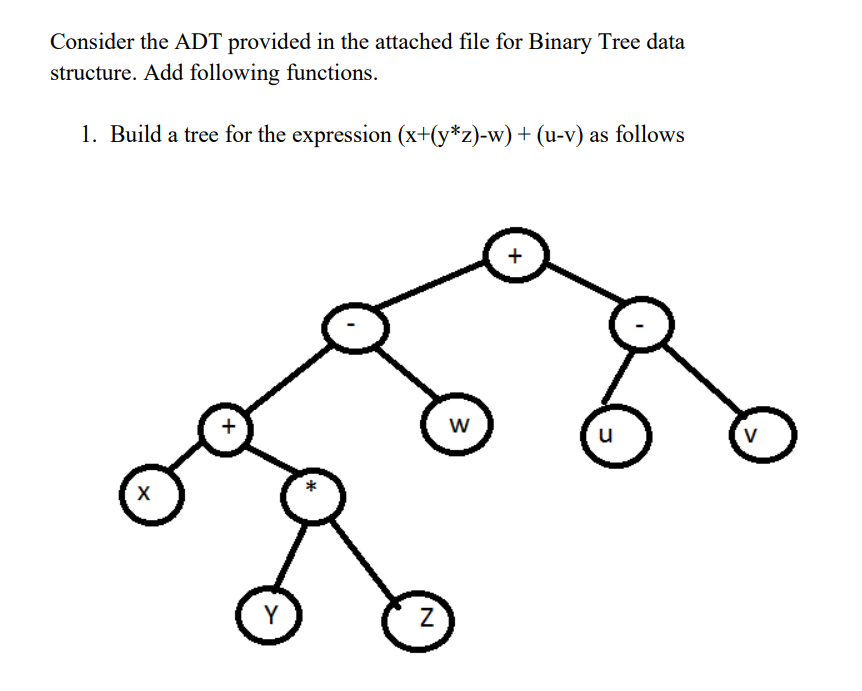
**S Abhishek AM.EN.U4CSE19147**

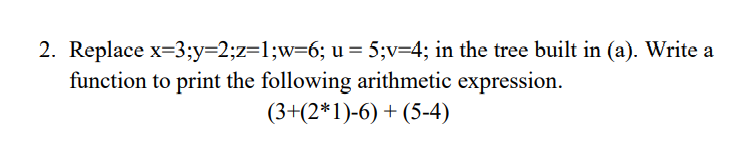
**Data Structures**

****

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)

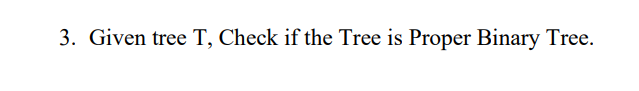
****

****

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)

****

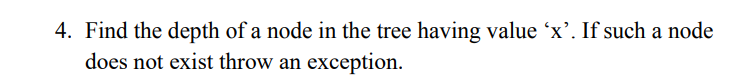
****

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")

****

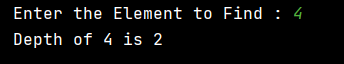
class Node:  
 def \_\_init\_\_(self**,** data**,** left=None**,** right=None):  
 self.data = data  
 self.left = left  
 self.right = right  
  
def full\_or\_not(head):  
 if head is None:  
 return True  
  
 elif head.left is None and head.right is None:  
 return True  
  
 elif head.left is not 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")

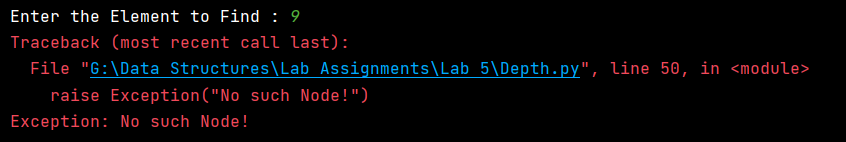
****

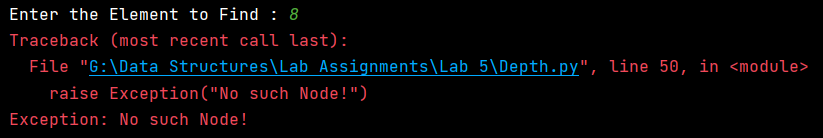
****

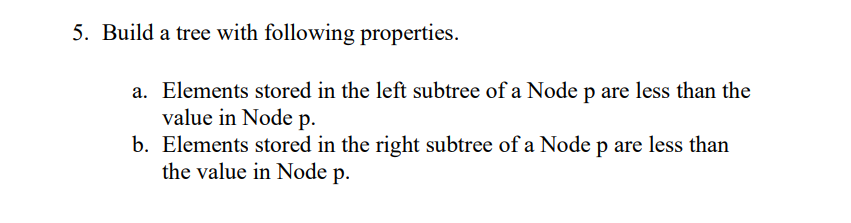
class Node:  
 def \_\_init\_\_(self**,** data**,** left=None**,** right=None):  
 self.data = data  
 self.left = left  
 self.right = right  
  
  
def get\_node(node**,** data**,** depth):  
 if node is None:  
 return **0** if node.data == data:  
 return depth  
  
 node\_pos = get\_node(node.left**,** 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!")

****

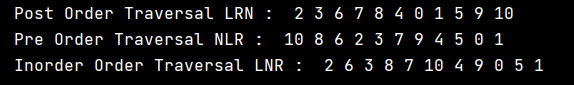
****

****

****

****

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)

****

**One Drive :** [**Click Me!!**](https://amritauniv-my.sharepoint.com/:f:/g/personal/sabhishek_am_students_amrita_edu/ErgE-zO5wK1Iok_prm_xlUEBkSprQ1VlLgO9sUPKzU_z3w?e=SbOhM0)

**Thankyou!!**