Lab Session 12

Basic Binary Tree:

```
class BTree:  3 usages  new *
    def __init__(self, item):  new *
        self.data = item
        self.lc = None
        self.rc = None

    def addlc(self, item):  2 usages (2 dynamic)  new *
        assert self.lc is None, "Left child is already present"
        self.lc = BTree(item)

    def addrc(self, item):  2 usages (1 dynamic)  new *
        assert self.rc is None, "Right child is already present"
        self.rc = BTree(item)

    def dellc(self):  new *
        assert self.lc is not None, "Left child absent"
        assert self.lc.lc is None and self.lc.rc is None, "This node has children"

        x = self.lc.data
        self.lc = None
        return x

    def delrc(self):  new *
        assert self.rc is not None, "Right child absent"
        assert self.rc.lc is None and self.rc.rc is None, "This node has children"

        x = self.rc.data
        self.rc = None
        return x
```

Ex a (i):

INPUT

```
if __name__ == "__main__":
    root = BTree("log")

    fact_node = BTree("!")
    fact_node.addlc("n")

    root.lc = fact_node

    print("Prefix: ", end="")
    root.traversePreOrder()
    print()

    print("Postfix: ", end="")
    root.traversePostOrder()
    print()
```

OUTPUT

```
Prefix: log ! n
Postfix: n ! log
```

Ex a (ii):

INPUT

```
if __name__ == "__main__":
    root = BTree("*")

    root.addlc("+")
    root.lc.addlc("A")
    root.lc.addrc("B")

    root.addrc("-")

    root.rc.addlc("+")
    root.rc.lc.addlc("C")

    plus_node = BTree("+")
    plus_node.addlc("D")
    plus_node.addrc("E!")
```

```
log_node = BTree("log")
log_node.lc = plus_node

root.rc.lc.rc = log_node

div_node = BTree("/")
div_node.addlc("G")
div_node.addrc("H")

sin_node = BTree("sin")
sin_node.lc = div_node

root.rc.rc = sin_node

print("Prefix: ", end="")
root.traversePreOrder()
print()

print("Postfix: ", end="")
root.traversePostOrder()
print()
```

OUTPUT

```
Prefix: * + A B - + C log + D E! sin / G H
Postfix: A B + C D E! + log + G H / sin - *
```

Ex a (iii):

INPUT

```
if __name__ == "__main__":
    root = BTree("=")
    root.addlc("x")

    div_node = BTree("/")
    root.rc = div_node

    plusminus_node = BTree("±")
    div_node.lc = plusminus_node

    plusminus_node.addlc("-b")

    sqrt_node = BTree("√")
    plusminus_node.rc = sqrt_node

    minus_node = BTree("-")
    sqrt_node.lc = minus_node

    power_node = BTree("^")
    power_node.addlc("b")
    power_node.addrc("2")
    minus_node.lc = power_node

    mult1 = BTree("*")
    mult1.addlc("4")
    mult1.addrc("a")

    mult2 = BTree("*")
    mult2.lc = mult1
    mult2.rc = BTree("c")

    minus_node.rc = mult2

    denom_node = BTree("*")
    denom_node.addlc("2")
    denom_node.addrc("a")
    div_node.rc = denom_node

    print("Prefix: ", end="")
    root.traversePreOrder()
    print()

    print("Postfix: ", end="")
    root.traversePostOrder()
    print()
```

OUTPUT

```
Prefix: = x / ± -b √ - ^ b 2 * * 4 a c * 2 a
Postfix: x -b b 2 ^ 4 a * c * - √ ± 2 a * / =
```

Ex b:

INPUT

```
def TreeSize(self):  # Node Count function
    if self.lc is None and self.rc is None:
        return 1

    lnc = 0
    rnc = 0

    if self.lc is not None:
        lnc = self.lc.TreeSize()

    if self.rc is not None:
        rnc = self.rc.TreeSize()

    return lnc + rnc + 1
```

```
if __name__ == "__main__":
    root = BTree(1)
    root.addrc(2)
    root.rc.addlc(3)
    root.rc.lc.addrc(5)
    root.rc.lc.addlc(4)

    print("Tree Size: ", root.TreeSize())
```

OUTPUT

```
Tree Size:  5
```

Ex c:

INPUT

```
def clearTree(self):  3 usages (2 dynamic)  new *
    if self.lc is None and self.rc is None:
        self.data = 0
        return

    if self.lc is not None:
        self.lc.data = 0
        self.lc.clearTree()

    if self.rc is not None:
        self.rc.data = 0
        self.rc.clearTree()

    self.data = 0
```

```
if __name__ == "__main__":
    root = BTree(1)
    root.addrc(2)
    root.rc.addlc(3)
    root.rc.lc.addrc(5)
    root.rc.lc.addlc(4)

    print("Traverse:", end=" ")
    root.traversePreOrder()
    print()

    print("Clear Tree:", end=" ")
    root.clearTree()
    print()

    print("Traverse:", end=" ")
    root.traversePreOrder()
    print()
```

OUTPUT

```
Traverse: 1 2 3 4 5
Clear Tree:
Traverse: 0 0 0 0 0
```

Ex d:

INPUT

```python
def doubleOrderTraversal(self):  3 usages (2 dynamic)  new
    print(self.data, end=" ")

    if self.lc is not None:
        self.lc.doubleOrderTraversal()

    print(self.data, end=" ")

    if self.rc is not None:
        self.rc.doubleOrderTraversal()
```

```python
if __name__ == "__main__":
    root = BTree(1)
    root.addrc(2)
    root.rc.addlc(3)
    root.rc.lc.addrc(5)
    root.rc.lc.addlc(4)

    print("Double Order Traversal:", end=" ")
    root.doubleOrderTraversal()
    print()
```

OUTPUT

```
Double Order Traversal: 1 1 2 3 4 4 3 5 5 2
```