**Department of Information Technology UIT2201**

**Programming and Data Structures 2022 – 2023**

**Exercise — 11**

**Part A**

**1)Write a parser that takes an expression string in postfix notation (for eg, "ab+a\*cd-e+/afg-\*h+-) and constructs the corresponding expression tree. You may assume that only binary operators are used in the expression and all the identifiers are single characters only.**

**Code:**

from LinkedBinarytree import LinkedBinaryTree                #importing the LinkedBinaryTree class from LinkedBinaryTree file

class ExpressionTree(LinkedBinaryTree):

    """

       This class is used for creating the expressiontree,

       and it contains below methods.

    """

    def \_\_init\_\_(self, item=None, t\_left=None, t\_right=None):

        super().\_\_init\_\_(item, t\_left, t\_right)

    def construct(self, string):

        """Constructs an expression tree from a postfix expression string"""

        s = []

        for character in string:

            if character in "+-\*/":

                r\_child = s.pop()

                l\_child = s.pop()

                s.append(ExpressionTree(character, l\_child, r\_child))

            else:

                s.append(ExpressionTree(character))

        self.root = s.pop().getRoot()

        return self.root

#Driver Code

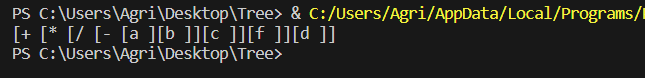
if \_\_name\_\_ == "\_\_main\_\_":

    E = ExpressionTree()

    E.construct("ab-c/f\*d+")

    print(E)

**Output:**

****

**2)Given a binary tree, write a Python code to convert the binary tree into its Mirror tree. Mirror of a Binary Tree T is another Binary Tree M(T) with left and right children of all non-leaf nodes interchanged.**

**Code:**

from LinkedBinarytree import LinkedBinaryTree               #importing the LinkedBinarytree class from LinkedBinarytree file

class mirrortree(LinkedBinaryTree):

    """

       This class is used for creating the mirrortree,

       and it contains below methods.

    """

    def \_\_init\_\_(self, item=None, t\_left=None, t\_right=None):

        "Initializing the Constructor"

        super().\_\_init\_\_(item, t\_left, t\_right)             #using the super()

    def Mirror(self,pos):

        """mirror function swaps the children of a node"""

        if pos is None:

            return None

        else:

            pos.left, pos.right = pos.right, pos.left           #Simultaneous Swappimg

            self.Mirror(pos.left)

            self.Mirror(pos.right)

#Driver Code

if \_\_name\_\_ == "\_\_main\_\_":

    t1=LinkedBinaryTree(11)

    t2=LinkedBinaryTree(22)

    t3=LinkedBinaryTree(1,t1,t2)

    x1=LinkedBinaryTree(1)

    x2=LinkedBinaryTree(2)

    x3=LinkedBinaryTree(10,x1,x2)

    y=LinkedBinaryTree(2,x3,t3)

    print("Orginal Tree")

    print(y)

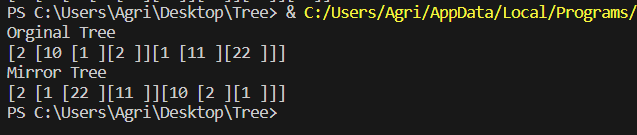
    M=mirrortree()

    M.Mirror(y.root)

    print("Mirror Tree")

    print(y)

**Output:**

****