NAME- Vaibhav Budh

ROLL NO- 07

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.left = None

self.right = None

class ExpressionTree:

def \_\_init\_\_(self):

self.root = None

# Function to construct expression tree from prefix expression

def construct\_tree(self, prefix):

stack = []

for char in reversed(prefix): # Traverse from right to left

if char.isalpha(): # Operand (e.g., 'a', 'b', 'c')

stack.append(Node(char))

else: # Operator (e.g., '+', '-', '\*', '/')

node = Node(char)

node.left = stack.pop() # Pop two operands from stack

node.right = stack.pop()

stack.append(node) # Push the subtree back to stack

# The root of the tree will be the last remaining node

self.root = stack[-1]

# Non-recursive post-order traversal

def post\_order\_traversal(self):

if not self.root:

return

stack1 = [self.root]

stack2 = []

while stack1:

node = stack1.pop()

stack2.append(node)

# Push left and right children to stack1

if node.left:

stack1.append(node.left)

if node.right:

stack1.append(node.right)

# Print nodes in post-order

while stack2:

print(stack2.pop().data, end=" ")

# Delete the entire tree (simulated by clearing the root reference)

def delete\_tree(self):

self.root = None

print("\nTree deleted.")

# Main program

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

# Take input for prefix expression

prefix\_expression = input("Enter a prefix expression: ")

# Create expression tree and construct it

tree = ExpressionTree()

tree.construct\_tree(prefix\_expression)

# Perform non-recursive post-order traversal

print("Post-order Traversal (Non-Recursive):")

tree.post\_order\_traversal() # Should print post-order of the expression tree

# Delete the entire tree

tree.delete\_tree()

