**6. Psuedocode: reverse a sentence/string using stack.**

Function reverseSentence(sentence):

# Step 1: Initialize an empty stack

stack = []

# Step 2: Iterate over each character in the sentence

for each character in sentence:

# Push each character onto the stack

stack.push(character)

# Step 3: Initialize an empty string to store the reversed sentence

reversed\_sentence = ""

# Step 4: Pop characters from the stack until it's empty

while stack is not empty:

# Pop each character and add it to the reversed\_sentence

reversed\_sentence += stack.pop()

# Step 5: Return the reversed sentence

return reversed\_sentence

**7. Psuedocode: check for balanced parenthesis in a given expression.**

Function isBalanced(expression):

# Step 1: Initialize an empty stack

stack = []

# Step 2: Iterate over each character in the expression

for each character in expression:

# Step 3: If the character is an opening bracket, push it onto the stack

if character is '(' or '{' or '[':

stack.push(character)

# Step 4: If the character is a closing bracket

elif character is ')' or '}' or ']':

# Step 5: Check if the stack is empty (means there's no matching opening bracket)

if stack is empty:

return False # Unbalanced

# Step 6: Pop the top element from the stack

top = stack.pop()

# Step 7: Check if the popped element is the matching opening bracket

if (character is ')' and top is not '(') or

(character is '}' and top is not '{') or

(character is ']' and top is not '['):

return False # Unbalanced

# Step 8: After processing all characters, check if the stack is empty

# If stack is empty, all brackets were matched; if not, they were unbalanced

return stack is empty

**8. Psuedocode: check for balanced parenthesis in a given expression.**

**1. Infix to Postfix Conversion**

**Function infixToPostfix(expression):**

# Step 1: Initialize an empty stack for operators and an empty string for the postfix expression

stack = []

postfix = ""

# Step 2: Define a function to check precedence of operators

Function precedence(op):

if op is '+' or '-':

return 1

if op is '\*' or '/':

return 2

if op is '^':

return 3

return 0

# Step 3: Iterate over each character in the expression

for each character in expression:

# Step 4: If character is an operand (e.g., letter or number), add it to postfix

if character is operand:

postfix += character

# Step 5: If character is '(', push it onto the stack

elif character is '(':

stack.push(character)

# Step 6: If character is ')', pop from stack until '(' is found

elif character is ')':

while stack is not empty and stack.top() is not '(':

postfix += stack.pop()

stack.pop() # Pop '('

# Step 7: If character is an operator

else:

while (stack is not empty and

precedence(character) <= precedence(stack.top())):

postfix += stack.pop()

stack.push(character)

# Step 8: Pop all remaining operators in the stack

while stack is not empty:

postfix += stack.pop()

# Step 9: Return the postfix expression

return postfix

1. **Infix to Prefix Conversion**

Function infixToPrefix(expression):

# Step 1: Reverse the infix expression

expression = reverse(expression)

# Step 2: Swap '(' with ')' and vice versa

for each character in expression:

if character is '(':

replace character with ')'

elif character is ')':

replace character with '('

# Step 3: Convert modified infix to postfix using infixToPostfix function

reversed\_postfix = infixToPostfix(expression)

# Step 4: Reverse the postfix expression obtained to get prefix

prefix = reverse(reversed\_postfix)

# Step 5: Return the prefix expression

return prefix