**Expression Evaluation and Notation Conversions using Stack**

1. **Infix to Postfix:**

for each token in infix expression:

if token is operand:

output.append(token)

else if token is '(':

stack.push(token)

else if token is ')':

while stack.top() != '(':

output.append(stack.pop())

stack.pop() // Remove '('

else if token is operator:

while stack is not empty and precedence(token) <= precedence(stack.top()):

output.append(stack.pop())

stack.push(token)

while stack is not empty:

output.append(stack.pop())

1. **Infix to Prefix:**

reverse the infix expression

for each token in reversed expression:

if token is operand:

output.append(token)

else if token is ')': // reversed '('

stack.push(token)

else if token is '(': // reversed ')'

while stack.top() != ')':

output.append(stack.pop())

stack.pop() // Remove ')'

else if token is operator:

while stack is not empty and precedence(token) < precedence(stack.top()):

output.append(stack.pop())

stack.push(token)

while stack is not empty:

output.append(stack.pop())

reverse the output to get prefix

1. **Postfix to Infix:**

for each token in postfix expression:

if token is operand:

stack.push(token)

else if token is operator:

operand2 = stack.pop()

operand1 = stack.pop()

expression = "(" + operand1 + token + operand2 + ")"

stack.push(expression)

return stack.top()

1. **Postfix to Prefix:**

for each token in postfix expression:

if token is operand:

stack.push(token)

else if token is operator:

operand2 = stack.pop()

operand1 = stack.pop()

expression = token + operand1 + operand2

stack.push(expression)

return stack.top()

1. **Prefix to Infix:**

for each token in prefix expression (scan right to left):

if token is operand:

stack.push(token)

else if token is operator:

operand1 = stack.pop()

operand2 = stack.pop()

expression = "(" + operand1 + token + operand2 + ")"

stack.push(expression)

return stack.top()

1. **Prefix to Postfix:**

for each token in prefix expression (scan right to left):

if token is operand:

stack.push(token)

else if token is operator:

operand1 = stack.pop()

operand2 = stack.pop()

expression = operand1 + operand2 + token

stack.push(expression)

return stack.top()