**Evaluate Arithmatic**

Problem Statement:

Take a sufficiently long arithmetic expression with +,-, \*, /,^ operators including parenthesis. Evaluate the expression.

* Input example:

Enter the expression: (5+6)-(8-7)^2

* Output Example:

10

Algorithm:

DEFINE FUNCTION eval(postfix):

SET stack TO []

FOR token IN postfix:

IF token EQUALS '+':

SET a TO stack.pop()

SET b TO stack.pop()

stack.append(a + b)

ELSEIF token EQUALS '-':

SET b TO stack.pop()

SET a TO stack.pop()

stack.append(a - b)

ELSEIF token EQUALS '\*':

SET a TO stack.pop()

SET b TO stack.pop()

stack.append(a \* b)

ELSEIF token EQUALS '/':

SET b TO stack.pop()

SET a TO stack.pop()

stack.append(a // b)

ELSEIF token EQUALS '^':

SET b TO stack.pop()

SET a TO stack.pop()

stack.append(pow(a,b))

ELSE:

stack.append(token)

RETURN stack[-1]

DEFINE FUNCTION calculate( s: str) -> int:

SET stack TO []

SET output TO []

SET rank TO {

'+': 1,

'-': 1,

'\*': 2,

'/': 2,

'^': 3,

}

SET num TO None

FOR c IN s:

IF c EQUALS ' ':

Continue

IF c IN "0123456789":

IF num is None:

SET num TO 0

SET num TO num \* 10 + int(c)

Continue

IF num is not None:

output.append(num)

SET num TO None

IF c EQUALS '(':

stack.append('(')

ELSEIF c EQUALS ')':

WHILE stack[-1] != '(':

output.append(stack.pop())

stack.pop()

ELSE:

WHILE stack and stack[-1] != '(' and rank[stack[-1]] >= rank[c]:

output.append(stack.pop())

stack.append(c)

IF num != None:

output.append(num)

WHILE stack:

output.append(stack.pop())

RETURN output

IF \_\_name\_\_ EQUALS "\_\_main\_\_":

math=INPUT("Enter the expression: ")

OUTPUT(eval(calculate(math)))

Proposed Python Code:

/\* ------- main.py ------- \*/

def eval(postfix):

stack = []

for token in postfix:

if token == '+':

a = stack.pop()

b = stack.pop()

stack.append(a + b)

elif token == '-':

b = stack.pop()

a = stack.pop()

stack.append(a - b)

elif token == '\*':

a = stack.pop()

b = stack.pop()

stack.append(a \* b)

elif token == '/':

b = stack.pop()

a = stack.pop()

stack.append(a // b)

elif token == '^':

b = stack.pop()

a = stack.pop()

stack.append(pow(a,b))

else:

stack.append(token)

return stack[-1]

def calculate( s: str) -> int:

stack = []

output = []

rank = {

'+': 1,

'-': 1,

'\*': 2,

'/': 2,

'^': 3,

}

num = None

for c in s:

if c == ' ':

continue

if c in "0123456789":

if num is None:

num = 0

num = num \* 10 + int(c)

continue

if num is not None:

output.append(num)

num = None

if c == '(':

stack.append('(')

elif c == ')':

while stack[-1] != '(':

output.append(stack.pop())

stack.pop()

else:

while stack and stack[-1] != '(' and rank[stack[-1]] >= rank[c]:

output.append(stack.pop())

stack.append(c)

if num != None:

output.append(num)

while stack:

output.append(stack.pop())

return output

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

math=input("Enter the expression: ")

print(eval(calculate(math)))

/\* ---------------------- \*/

Conclusion:

The proposed algorithm has a runtime of O(n), where n is the length of the string.

Limitations and assumptions for this algorithm include:

1.This program is valid for only integer numbers.

2.The numbers in this program should be within integer range.