"""

Thomas Morris

October 3, 2019

Write/Test a stack (called Min Stack) that supports

push, pop, top, and retrieving the minimum element in constant time.

"""

class MinStack:

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

self.\_data = []

self.\_minStack = []

def \_\_len\_\_(self):

return len(self.data)

def is\_empty(self):

return len(self.\_data) == 0

def push(self, e):

"""

use the second stack to hold the minimum value

only push a value if it is less than or equal to the top of the min

"""

if self.is\_empty():

self.\_minStack.append(e)

elif self.\_minStack[-1] >= e:

self.\_minStack.append(e)

self.\_data.append(e)

def top(self):

if self.is\_empty():

raise Exception('Stack is empty')

return self.\_data[-1]

def pop(self):

"""

Check if the element being removed from the main stack

is the same as the element in the min stack and remove it

when they are equal

"""

if self.is\_empty():

raise Exception('Stack is empty')

if self.\_data[-1] == self.\_minStack[-1]:

self.\_minStack.pop()

return self.\_data.pop()

def getMin(self):

if self.is\_empty():

raise Exception('Stack is empty')

return self.\_minStack[-1]

if \_\_name\_\_ == '\_\_main\_\_':

minStack = MinStack()

minStack.push(-2)

print(minStack.getMin())

minStack.push(0)

print(minStack.getMin())

minStack.push(-3)

print(minStack.getMin())

minStack.pop()

print(minStack.top())

print(minStack.getMin())

minStack.push(-5)

minStack.push(-5)

minStack.push(2)

minStack.push(7)

print(minStack.getMin())

minStack.pop()

minStack.pop()

minStack.pop()

print(minStack.getMin())

minStack.pop()

minStack.pop()

print(minStack.getMin())

"""

Thomas Morris

October 3, 2019

Write/Test a program which evaluate arithmetic expression using your

own stack class. Please follow the lecture slides Stack, pp. 15-17.

"""

class Stack:

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

self.\_data = []

def size(self):

return len(self.\_data)

def is\_empty(self):

return len(self.\_data) == 0

def push(self, e):

self.\_data.append(e)

def top(self):

if self.is\_empty():

raise Exception('Stack is Empty')

return self.\_data[-1]

def pop(self):

if self.is\_empty():

raise Exception('Stack is Empty')

return self.\_data.pop()

class Eval:

valStk = Stack()

opStk = Stack()

def doOp():

x = int(Eval.valStk.pop())

y = int(Eval.valStk.pop())

op = Eval.opStk.pop()

if op == "+":

Eval.valStk.push(y+x)

elif op == "-":

Eval.valStk.push(y-x)

elif op == "\*":

Eval.valStk.push(y\*x)

elif op == "/":

Eval.valStk.push(y/x)

elif op == "<":

Eval.valStk.push(y<x)

elif op == ">":

Eval.valStk.push(y>x)

elif op == "≤":

Eval.valStk.push(y<=x)

elif op == "≥":

Eval.valStk.push(y>=x)

else:

raise Exception('There is an error with op')

def prec(item):

if item == "$":

return 0

elif item == "<" or item == ">" or item == "≤" or item == "≥":

return 1

elif item == "+" or item == "-":

return 2

elif item == "\*" or item == "/":

return 3

else:

raise Exception('There is an error with prec')

def repeatOps(refOp):

while (Eval.valStk.size() > 1) and (Eval.prec(refOp) <= Eval.prec(Eval.opStk.top())):

Eval.doOp()

def evalExp(self, exp):

for i in range(0,len(exp)):

#check if it is digit

if exp[i].isdigit():

Eval.valStk.push(exp[i])

else:

#otherwise its an operator

Eval.repeatOps(exp[i])

Eval.opStk.push(exp[i])

#send the end operator

Eval.repeatOps("$")

#return the value left at the top of the stack

return Eval.valStk.top()

def reset(self):

Eval.valStk = Stack()

Eval.opStk = Stack()

if \_\_name\_\_ == '\_\_main\_\_':

print(Eval().evalExp("4+5-9>4"))

Eval().reset()

print(Eval().evalExp("6/6+8-4\*3+6\*4"))

Eval().reset()

print(Eval().evalExp("7/6+8-7\*3+7\*4"))

Eval().reset()

print(Eval().evalExp("7/6+8-7<3+7\*4"))

"""

Thomas Morris

October 3, 2019

Implement a program that can input an expression in postfix notation

and output its value.

"""

class Stack:

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

self.\_data = []

def \_\_len\_\_(self):

return len(self.\_data)

def is\_empty(self):

return len(self.\_data) == 0

def push(self, e):

self.\_data.append(e)

def top(self):

if self.is\_empty():

raise Exception('Stack is Empty')

return self.\_data[-1]

def pop(self):

if self.is\_empty():

raise Exception('Stack is Empty')

return self.\_data.pop()

def evalPost(self, exp):

for i in range(0,len(exp)):

#check if it is a digit

if exp[i].isdigit():

self.push(exp[i])

else:

#otherwise it is an operator so evaluate the expression

x = int(self.\_data.pop())

y = int(self.\_data.pop())

if exp[i] == "+":

self.push(y+x)

elif exp[i] == "-":

self.push(y-x)

elif exp[i] == "\*":

self.push(y\*x)

elif exp[i] == "/":

self.push(y/x)

else:

raise Exception('There is an error with op')

return self.pop()

if \_\_name\_\_ == '\_\_main\_\_':

post = Stack()

print(post.evalPost("52+83-\*4/"))

print(post.evalPost("11+"))

print(post.evalPost("11+25\*-5+6"))

print(post.evalPost("89/123\*-63+"))

print(post.evalPost("45/\*99+291"))