Postfix Notation

LAB # 8

By

Corey Henry

And

Aaron Shepard

***“On my honor, as a Mississippi State University student, I have neither***

***given nor received unauthorized assistance on this academic work.”***

Signature:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CSE-1384-06-201430 Intermediate Computer Programming

Class Section # 6

Jesse Farek

4/7/2015

**Analysis and Conclusions**

The purpose of this lab was to to read a file and take the numbers and put them into a stack and perform postfix math on them. The lab overall was a little harder then I thought it was going to be. If functions were a major part of the lab, as well as repetition. If I could go back and do it I would create a function that did the repetition and just call it so the code would have been more neat and required less space. Knowledge of opening the file and stripping the lines was also needed. Trying to get the loops to end was a major problem in our code, we eventually had to add a Flag into the code and only print when the flag was true. The break function was also used to exit the loop to stop the flag from being reedited. Overall I think the code is a little rough and with another try could be a lot shorter and would require less factors to prevent the constant errors we had.

Source Code:

#Corey Henry & Aaron Shepard #Date Assigned: 17 mar 15

# #

#Course CSE 1384 Sec 06 #Date Due: 7 apr 15

#File name: Lab

#

#Program description- takes a creates a stack and does basic math interpitations

from Stack import\*

#creates main and gets file name and adds.txt if needed

def main():

file\_name = input("Please enter the file you wish to open. ")

if not file\_name.endswith(' .txt'):

file\_name += '.txt'

#opens the file, creates a object called my\_stack

file = open(file\_name)

my\_stack = Stack()

#loops to split the file into lines the splits them up

for each\_line in file:

each\_term = each\_line.strip()

lists = each\_term.split()

flag = False

for each in lists:

#if statement for addition

if each == '+':

if my\_stack.is\_empty() == False:

x= my\_stack.pop()

if my\_stack.is\_empty() == False:

y= my\_stack.pop()

value = x+y

my\_stack.push(value)

#invalid statement

if my\_stack.is\_empty()== True:

print('ERROR: ', each\_term , ' is an invalid postfix expression')

flag = True

print('\n')

break

#if statement for minus

elif each == '-':

if my\_stack.is\_empty() == False:

x= my\_stack.pop()

if my\_stack.is\_empty() == False:

y= my\_stack.pop()

value = y-x

my\_stack.push(value)

#invalid statement

if my\_stack.is\_empty()== True:

print('ERROR: ', each\_term, ' is an invalid postfix expression')

flag = True

print('\n')

break

#if statement for multiplication

elif each == '\*':

if my\_stack.is\_empty() == False:

x= my\_stack.pop()

if my\_stack.is\_empty() == False:

y= my\_stack.pop()

value = x\*y

my\_stack.push(value)

#invalid statement

if my\_stack.is\_empty()== True:

print('ERROR: ', each\_term, ' is an invalid postfix expression')

flag = True

print('\n')

break

#if statement for division

elif each == '/':

if my\_stack.is\_empty() == False:

x= my\_stack.pop()

if my\_stack.is\_empty() == False:

y= my\_stack.pop()

value = y/x

my\_stack.push(value)

#invalid statement

if my\_stack.is\_empty()== True:

print('ERROR: ', each\_term , ' is an invalid postfix expression')

flag = True

print('\n')

break

else:

try:

my\_stack.push(int(each))

except ValueError as err:

print('ERROR: ', each\_term , ' is an invalid postfix expression')

flag = True

print('\n')

break

#conditions for printing out the strings

if flag == False:

x= my\_stack.pop()

if my\_stack.is\_empty() == True:

print('Expresssion: ',str(each\_term))

print('Answer = ',x)

print('\n')

else:

print('ERROR: ' ,each\_term , 'is an invalid postfix expression')

print('\n')

main()

