Mini Project: Symbol Balance

Joshua Canlas
Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, NJ
jcanlas1@stevens.edu

I. Design and Algorithm

A. *Pseudocode*

Algorithm 1 parseInput

```
Set the input string to a member variable
Call checkSyntax
if checkSyntax returns Symbol Balanced then
    Remove comments from input, if any
end if
Return parsed input
```

Algorithm 2 checkSyntax

```
Create an empty stack
for each element in the string do
    if element is an open symbol then
        push in the stack
    end if
    if element is a closing symbol then
        if stack is empty then
            EmptyStack error, print closing symbol
        else
            if returned is not correct symbol then
                Mismatch Error, print mismatched symbols
            else
                pop the stack
            end if
        end if
    end if
end for
if the stack is not empty after the for loop then
    NonEmptyStack error, print top symbol in stack
else
    Symbol Balanced
end if
```

Algorithm 3 postfixExpress

```
Create an empty stack
for each item in parsedInput do
    if operand then
        print as an output
    end if
    if left symbol then
        push in stack
    end if
    if right symbol then
        pop all until the corresponding left symbol
    end if
    if operator then
        pop all until a symbol of lower priority, then push the operator
    end if
end for
```

B. *Data Structures*

I have implemented my own stack data structure in Python, with the following member variables and functions:

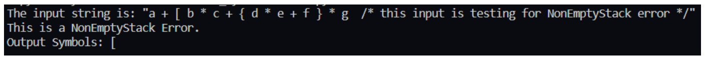
- Member variables: array
- Member functions: is_empty, push, pop, size, peek

C. *API Specifications*

- Stack()
- Stack.is_empty()
- Stack.push(char item)
- Stack.pop()
- Stack.size()
- Stack.peek()
- SymbolBalance(str input)
- SymbolBalance.parseInput()
- SymbolBalance.checkSyntax()
- SymbolBalance.postfixExpress(str parsedInput)
- runSymbolBalance(str input)

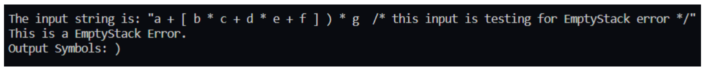
II. Results

A. Example Screenshots



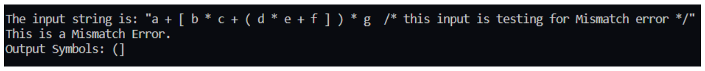
```
The input string is: "a + [ b * c + { d * e + f } * g  /* this input is testing for NonEmptyStack error */"
This is a NonEmptyStack Error.
Output Symbols: [
```

Fig. 1. NonEmptyStack test



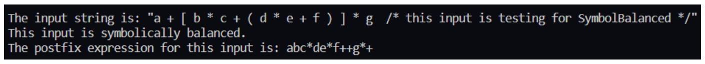
```
The input string is: "a + [ b * c + d * e + f ] ) * g  /* this input is testing for EmptyStack error */"
This is a EmptyStack Error.
Output Symbols: )
```

Fig. 2. EmptyStack test



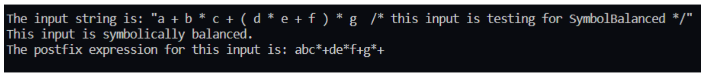
```
The input string is: "a + [ b * c + ( d * e + f ] ) * g  /* this input is testing for Mismatch error */"
This is a Mismatch Error.
Output Symbols: (]
```

Fig. 3. Mismatch test



```
The input string is: "a + [ b * c + ( d * e + f ) ] * g  /* this input is testing for SymbolBalanced */"
This input is symbolically balanced.
The postfix expression for this input is: abc*de*f++g*+
```

Fig. 4. SymbolBalanced test #1.



```
The input string is: "a + b * c + ( d * e + f ) * g  /* this input is testing for SymbolBalanced */"
This input is symbolically balanced.
The postfix expression for this input is: abc*+de*f+g*+
```

Fig. 5. SymbolBalanced test #2.

B. Additional Information

I am assuming that the user input will not include non-alphanumeric values for the infix expression. I have also written some tests that will check to see if the SymbolBalance class works properly. If the user wants to add additional tests, they will need to go into the Python script and define their own input strings, then call runSymbolBalance.

I have updated my branch on GitHub (Canlas9875). The Python script is located under the SymbolBalance folder (Canlas9875_SymbolBalance.py). Please contact me if you have any questions or feedback.