Lab 3: Application for the Stack ADT

I. Create a new directory for this lab (call it *Lab3*). Copy your *StackLinkedList* class from *Lab2* into *Lab3*.

II. Design and implement a *String Checker* function called isBalanced(string):

- ***isBalanced***: Th method has one *String* type parameter and returns a *boolean* value – *true* if, in the parameter string, all brackets, braces, and parenthesis are balanced, and false otherwise. We say that brackets, braces, and parenthesis are balanced if every such right symbol has its corresponding left counterpart (and vice versa), and they are correctly arranged – for example {[ ]} is legal while {[}] is not. Note that the parameter string may have symbols other than brackets, braces, and parenthesis but those don't play any part in the decision making.

  The following simple algorithm requires a stack – **use your own *StackLinkedList* class created in Lab2**. You will need to create and use **a stack of *Characters*** for your algorithm.
  Note: let's agree to call brackets, braces, and parenthesis as "symbols of interest".

Algorithm:
Make an empty stack. Analyze characters of the parameter string one by one.
If a character is not an opening or closing symbol of interest, then move to the next character of the string.
If the character is an opening symbol, then push it onto the stack.
If the character is a closing symbol, then:

- If the stack is empty, that means there is no opening counterpart for that symbol, and you can make a conclusion that the string is not balanced
- If the stack is not empty, pop the top. If the popped element is not the opening counterpart of the symbol in consideration, then you can make a conclusion that the string is not balanced; otherwise (this is the case when the closing symbol had its opening counterpart) move to the next character of the parameter string.

If all characters of the input string are scanned, check the stack – if the stack is not empty, then you can make a conclusion that the string is not balanced since there were some opening symbols in the stack whose closing counterparts were never found; otherwise (i.e. the stack is empty) the string IS balanced.

III. Test your program thoroughly: Make sure the program works as expected. Give **long** strings with **many** symbols of interest (not just 2-3 pairs). Here are some test cases – test **each case several times** for different values:

a) balanced strings with no symbols of interest
b) balanced strings with several pairs of symbols of interest
c) unbalanced strings: there are opening symbols of interest that do not have closing counterparts; however, all paired symbols of interest are correctly arranged.
d) unbalanced strings: there are closing symbols of interest that do not have opening counterparts; however, all paired symbols of interest are correctly arranged.
e) unbalanced strings: all opening symbols have closing counterparts, but they are not correctly arranged.

f) unbalanced strings: there are opening symbols of interest that do not have closing counterparts and some paired symbols of interest are NOT correctly arranged.
g) unbalanced strings: there are closing symbols of interest that do not have opening counterparts and some paired symbols of interest are NOT correctly arranged.