# CS3310: Assignment I
# Balanced Parentheses with Stacks and Queues
# (Due: 9/26/2025 @11:59pm)

## Concepts

- Linked Lists
- Stack
- Queue

## Background

Strings from context-free languages are recognized with the use of stack structures. Such languages, as in the set of all finite strings with balanced parentheses can be recognized. It can be determined whether or not a string fits within a particular language.

## Problem Specification

Write a program to read random strings from a user, consisting of any number of "(" and ")" in any combination, and determine whether they contain balanced parentheses until the user wishes to end the program. A string with balanced parentheses is one where each "(" is paired with a ")". For instance, the string "()((()()))" has balanced parentheses, but the strings "(", ")", "(()", "))((" and "()(()))()()" do not have balanced parentheses. Given the data structures from the course material, there are two ways you can implement a technique for checking balanced parentheses.

1. Implement a class that uses a stack to determine if a string has balanced parentheses
2. Implement a class that uses queues to determine if a string has balanced parentheses (Hint: two queues can be used to simulate a stack's behavior).

Instead of using arrays for the underlying structures of stacks and queues, use linked list representations that do not use built-in list classes. The program must be implemented in C++ and well-commented, i.e. use block comments for describing each method in a class and give some lines of comments to explain statements. Programs with very few comments (as in just commenting on one of two methods only) or no comments at all will receive a small penalty.

You must write Class Node, a Class LinkedList, a Class Stack, a Class Queue, a Class StackParenthesesChecker, and a Class QueueParenthesesChecker.

```
class Node(object):
        __data = None
        __prev = None
        __next = None

        # constructor for Node class
        def __init__(self):
                # code goes here
```

```
class LinkedList(object):
        __head=None
        __tail= None
        __capacity = 0
        __size=0

        # constructor for LinkedList class
        def __init__(self):
                # code goes her

        # add item x to list at index i
        def add(self, i, x):
                # code goes here

        # remove item at index i from the list
        def remove(self, i):
                # code goes here (should return item from list or None if item is not in the list)

class Stack(object):
        __linkedList = …
        __top = …

        # constructor for stack class
        def __init__(self):
                # code goes here

        # push item onto stack
        def push(self, x):
                # code goes here

        # pops item from top of stack
        def pop(self):
                # code goes here (should return item from top of stack or None if stack is empty)

        # returns Boolean of whether stack is currently empty
        def isEmpty(self):
                # code goes here

        # returns Boolean of whether stack is currently full
        def isFull(self):
                # code goes here

        # clears the stack
        def clear(self):
                # code goes here
```

```
        # looks at the top item of the stack without removing it
        def peek(self):
                # code goes here

class Queue(object):
        __linkedList = …
        __front = …
        __rear = …

        # constructor for Queue class
        def __init__(self):
                # code goes her

        # adds item to front of queue
        def enqueue(self, x):
                # code goes here

        # removes item from rear of queue
        def dequeue(self):
                # code goes here (should return item from end of queue or None if queue is
                  empty)

        # returns Boolean of whether queue is currently empty
        def isEmpty(self):
                # code goes here

        # returns Boolean of whether queue is currently full
        def isFull(self):
                # code goes here

        # clears the queue
        def clear(self):
                # code goes here

        # looks at the item at the end of the queue without removing it
        def poll(self):
                # code goes here

class StackParenthesesChecker(object):
        __stack = …

        # constructor for StackParenthesesChecker class
        def __init__(self):
                # code goes here
```

```
        # Check if string s has balanced parenthesis
        def isBalanced(self, s):
                # code goes here

class QueueParenthesesChecker(object):
        __queue1 = …
        __queue2 = …

        # constructor for QueueParenthesesChecker class
        def __init__(self):
                # code goes here

        # Check if string s has balanced parenthesis
        def isBalanced(self, s):
                # code goes here
```

**Create several string examples to check functionality of your program. Please see Testing Phase below.**

## Implementation Phase

You must work on the assignment *individually*. If any external source code or information from a website is applied to your implementation, you **MUST** acknowledge the source with comments in your code.

## Testing Phase

```
def main():
        checker1 = StackParenthesesChecker()
        checker2 = QueueParenthesesChecker()
        stack = Stack()
        queue1 = Queue()
        queue2 = Queue()
        setAttribute(stack, '_Stack__linkedList', LinkedList())
        setAttribute(queue1, '_Queue__linkedList', LinkedList())
        userString = None

        # more setting statements here

        # a loop to ask input via console for new string to check with both checkers
        While user wants to continue program:
                userString = get user string via console
                //  add more code here to set up checkers and their data structures

                If checker1.isBalanced(userString) and checker2.isBalanced(userString)):
                        print('The input string %s has balanced parentheses.', userString)
                Else:
                        print('The input string %s does not have balanced parentheses.', userString)
```

```
            # get user continuation of program via console

If name == '__main__':
        main()
```

**Expected Output:**
Accurate determination of balanced parentheses in input strings, for both string checking techniques. The if-statement where the methods are called for checking balance determines whether checks for both techniques are equivalent.

## Assignment Submission

- Submit a .zip file with all your source, input, and output files to the dropbox designated for Assignment 1 in E-learning.