CS3310: Assignment I
Balanced Parentheses with Stacks and Queues
(Due: 9/28/2023 @11:59pm)

Concepts

- Linked Lists
- Queue
- Stack

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 may be implemented either in Python, Java, or C++. The program implemented in either language **MUST** be 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.

If you implement the program in Python, you must write a Class Queue, a Class Stack, a Class Node, a Class LinkedList, and a Class StackParenthesesChecker.

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

        # constructor for Node class
        def __init__(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
```

If you implement the program in Java, the main class should be named “Application” and the following Interfaces must be implemented.

```
public Interface IParenthesesChecker{
        // return truth of whether the string s has balanced parentheses
        boolean isBalanced(String s);
}

public Interface INode<T>{
        // set the data item for the node
        void setData(T data);
        // return the data item reference stored in the node
        T getData();
        // set the pointer to the next linked node to this one
        void setSucc(INode<T> succ);
        // return the pointer to the next linked node to this one
        INode getSucc();
}

public Interface IList<T>{
        // return the pointer to the head node of the list
        INode<T> getHead();
        // set the pointer to the head node of the list
        void setHead(INode<T> head);
        // set the pointer to the tail node of the list
        void setTail(INode<T> tail);
        // return the pointer to the tail node of the list
        INode<T> getTail();
        // return the number of items in the list
        int getSize();
```

```
        // set the number of items in the list
        void setSize(int size);
        // return the max number of items the list can hold
        int getCapacity();
        // set the max number of items the list can hold
        void setCapacity(int capacity);
        // return the truth of whether the list is full
        boolean isFull();
        // return the truth of whether the list is empty
        boolean isEmpty();
        // remove item at index i from the list
        T remove(int i);
        // add item x to list at index i
        boolean add(int i, T x);
}

public Interface IStack<T>{
        // set the pointer to the top node of the stack
        void setTop(INode<T> top);
        // return the pointer to the head node of the list
        INode<T> getTop();
        // add new item x to the top of the stack
        boolean push(T x);
        // remove an item from the top of the stack
        T pop();
        // set the pointer to the list used as the stack
        void setList(IList<T> list);
        // return the pointer to the list used as the stack
        IList<T> getList();
        // clear all items from the stack
        void clear();
        // looks at the item at the top of the stack without removal
        T peek();
}

public Interface IQueue<T>{
        // set the pointer to the front node of the queue
        void setFront(INode<T> front);
        // return the pointer to the front node of the queue
        INode<T> getFront();
        // set the pointer to the rear node of the queue
        void setRear(INode<T> rear);
        // return the pointer to the rear node of the queue
        INode<T> getRear();
        // add new item x to the front of the queue
        boolean enqueue(T x);
```

```
        // remove an item from the rear of the queue
        T dequeue();
        // set the pointer to the list used as the queue
        void setList(IList<T> list);
        // return the pointer to the list used as the stack
        IList<T> getList();
        // clear all items from the queue
        void clear();
        // looks at the item at the rear of the queue without removal
        T poll();
}
```

For implementing interface INode, the expected implementing class should be a doubly-linked node, as in the Python version. INode provides only the getter method for next. The implemented Node class will also require getters and setters for a **prev** attribute, and you will need to cast in your linked list implementation to access the method. Just as in the Python version, two ParenthesesChecker implementing classes must be implemented. Setters and getters are not written for stacks or queues in the interface due to that information only being in context of the class. So, when setting the data structures in the main method, class casting must be used as well to connect the queues to QueueParenthesesChecker and the stack with StackParenthesesChecker.

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

In Python:

main.py file…

```
# add import statements here

def main():
        checker1 = StackParenthesesChecker()
        checker2 = QueueParenthesesChecker()
        stack = Stack()
        queue1 = Queue()
        queue2 = Queue()
        setAttribute(stack, '_Stack__linkedList', LinkedList())
        setAttribute(queue1, '_Queue__linkedList', LinkedList())
```

```
        # more setting statements here

        userString = None

        # 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()
```

In Java:

```
Public static void main(String[] args){

        IParenthesesChecker checker1 = new StackParenthesesChecker();
        IParenthesesChecker checker1 = new QueueParenthesesChecker();
        IStack<String> stack = new Stack<String>();
        IQueue<String> queue1 = new Queue<String>();
        IQueue<String> queue2 = new Queue<String>();
        String userInput = null;
        stack.setList(new LinkedList<String>());

        //more setting statements here

        //get user input for continuing program from console

        While(user wants to continue program){
                userString = get user string via console
                //  add more code here to set up checkers and their data structures

                //note, this is in partial pseudocode, not complete Java syntax
                If (checker1.isBalanced(userString) and checker2.isBalanced(userString)){
                        print("The input string " + userString+ " has balanced parentheses.");
                }Else{
                        print("The input string " + userString+ "  does not have balanced parentheses.");
                }

                // get user continuation of program via console
        }
}
```

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.