CS 3310 Data and File Structure

Instructor Dr.Ajay K. Gupta

Name: Mariam Ghali

**Linear Data: Stack and Queue**

**Phase 1: objective**

1. The main goal of this assignment is to get the number of the unbalanced parentheses from a string by using the idea Stack and queue
2. Comprehend and understand the mechanism of linked lists in depth
3. Creating stacks and queues based on linked lists
4. 3. Use stacks and queues to solve practical problems

The program should do the following:

1. Read an input file
2. Call the method named QueueCheckedBalanced Parentheses
3. Call the method names StackCheckedBalancedParentheses
4. Time both two methods
5. Return the number of the unbalanced parentheses in an output file

**Phase 2: Description**

The program consists of 7 classes:

1. StackQueueDemo
2. StackQueueDemo will be the main class for this project and it will manage to read an input file that has text string and read the file line by line using the buffer and write the number of the parentheses into and output file.
3. It will also check the time of each method StackCheckBalancedParentheses and QueueCheckBalancedParenthesesthat will take for executing those methods to see the difference between stack and queue.
4. It will create the objects and instance of the class CharStack, charQueue, QueueCheckBalancedParentheses, StackCheckBalancedParentheses.
5. CharNode
   1. This class will be responsible for creating the nodes whenever we call it
   2. The method CharNode will create the value by using the the attribute myDate and a pointer to the ode
6. CharStack
   1. This class will create the main nodes of the list which are Head Tail and the temp nodes.
   2. Then it will create the insert and delete method and insert method
   3. Insert method
      1. get the value from the StackQueueDemo class the one the user wants to insert
      2. there are 2 situations if the linkedlist is empty then create a node that will hold this value in the data attribute in charNode and assigen this node to be the head and the tail of the linked list
      3. case 2 if the insert at the end make temp = tail and move through the whole linked list until we find the next one is null so we create a node and make it the tail of the new linked list
   4. Delete method
      1. get the value we need to delete then we move through the list
      2. we have 4 cases
      3. first if the list is empty return null
      4. second if the list has only one element and the value is the same with the one we need to delete so we delete and return the value that is delete if it is not the value return null
      5. third if required value is in the head of the list we return the value and move the pointer to the next by using head = temp.next
      6. Forth is to go through the whole list if we found it move the pointer to the next node and it will be deleted directly and return the value if not return null
7. CharStack
   1. Char stack implements charlist class and has 2 methods push() and pop()
   2. Pop() return the last value we add FILO
      1. 3 cases first if the list is empty return null
      2. second if it has one element return the value and set both head and tail to null
      3. third since stack it FILO so we go through the whole lint until we find the tail return the value of the tail and move the tail one step backword and we keed doing it until we reach the first element
      4. return the last char of the list the value at "tail'
   3. Push() add starts from the beginning
      1. use the same method of the insert
      2. push starts at the beginning of the linked list
8. CharQueue
   1. it is implemented by CharList and has 2 methods enqueue() and dequeue()
   2. engueue() similar to insert which means we ass to the beginning of the list
      1. using the same method as insert which is basicaly is adding the element once we receive it
   3. Dequeue() returns the first element that added to the list FIFO
      1. 3 cases first if the list is empty return null
      2. Second if it has one element return the value and set head = tail
      3. the last one is get the value of the head then move to the next node and get the value and so one return the value of the first node
9. StackCheckBalancedParentheses class
   1. this class has a method called CheckBalancedParentheses to check the balance by using the idea stack with the aid of it is properties push() and pop() by the attribute text
   2. The method CheckBalancedParentheses will do the following
      1. we get the string and then convert it to char and push only the parentheses to the list
      2. however when we find an open parentheses we push it as a closed one and when it is a closed change it to open
      3. To make it easier when we pop it and get the right order and the valid number of the unbalanced parentheses
      4. After we get the stack full of the parentheses I use a while loop so it will go through the whole stack
      5. then we pop it until we reach empty
      6. so, we take the first node it is it open pop it from the stack and increase the counter open
      7. if it is close, check first if it the counter open == 0 then increase the close counter
      8. if it's a closed parentheses and open !=0 then decrement open by 1.then get the total by adding open and close and return total
      9. return will return the total number of the unbalanced parentheses
10. QueueCheckBalancedParentheses class
    1. This class will have the text attribute and it will convert it at a char
    2. Call the method CheckBalancedParentheses to get the number of the unbalanced parentheses by using Queue
       1. we get the string and then convert it to char and enqueue only the parentheses to the list
       2. dequeue char by char and check until we reach and empty list
       3. we take the first node it is it open dequeue it from the queue and increase the counter open
       4. if it is close , check first if it the counter open == 0 then increase the close counter
       5. if it's a closed parentess and open !=0 then decrement open by 1.then get the total by adding open and close and return total
       6. return will return the total number of the unbalanced parentheses

**Pseudocode**

StackQueueDemo

Text, line, queue, stack, startTime, endTime, totalTime,SecondsForStack, secondsForQuese

TRY

Open fie

IF !file.exist

Create file name outputfile.txt

Read file

WHILE Line = bufferedReader.readLine() !=Null

Read the file line by line

Call the Stack method

Call Queue method

Write to an output.txt

Close the buffer

Close the file

CATCH

File not found

2. CharNode

CharNode

Value

Next

3. CharStack

Pop

IF head == null

Return null

IF head==tail

Get the value

Set head and tail to null

Return the value

WHILE true

IF temp.next is tail

Get the value of the tail

Set the tail to null

Set temp.net ot be the tail

ELSE

Move to the next node

Break at the end of the list

Push

Call inset method

1. CharQueue
   1. Enqueue

Insert();

* 1. Dequeue

IF head == null

Return null

IF head==tail

Return the value

Set both head and tail to null

ELSE

Get the next pointer

Return the Value

1. StackCheckBalancedParentheses

Repeat i time “the length of the string ’

IF i==(

Push(‘)’)

IF i== )

Push(‘(’)

WHILE stack not empty

IF c==(

Open++

ELSE IF c==) and open!=0

Open—

ELSE IF c==) and open = = 0

Close++

Return open+close

1. QueueCheckBalancedParentheses

Repeat i time “the length of the string ’

IF i==(

Enqueue(“(”)

IF i== )

Enqueue(“)”)

WHILE Queue not empty

IF c==(

Open++

ELSE IF c==) and open!=0

Open—

ELSE IF c==) and open = = 0

Close++

Return open+close

**Thermotical analysis**

In the function StackCheckBalancedParentheses it takes o(n) to go through the string and it takes o(i)” i is the number of the parentheses ”push only the parentheses form the string. Then we exchange the open with the close it takes also o(i). Then we get the number of the open and the close parentheses which take o(i). Then the total complexitity of this function is o(n+i+i) which is o(n)

In the function QueueCheckBalancedParentheses it takes o(n) to go through the string and it takes o(i) )” i is the number of the parentheses “enqueue only the parentheses form the string. Then we get the number of the open and the close parentheses which take o(i). Then the total complexitity of this function is o(n+i) which is o(n)

**The empirical analyses**

It aggress with the thermotical analysis by showing that when we increase the length of the string, it takes more time to get the number of the parentheses in both methods Stack and Queue

In the Graph below it shows the stack is the yellow graph and the Queue is the blue one. The x-axis implies the length of the string and the y-axis implies the time it takes the method to execute.