

Assignment 2

Linear Data Structures

Release Time	Due Date
10/02/17	10/15/17

Objectives

- Comprehend and understand the mechanism of linked lists in depth
- Creating stacks and queues based on linked lists
- Use stacks and queues to solve practical problems

Problem Specification

Write a program to solve the following problem. *Note that in this assignment you are NOT allowed to use Java (or other languages) Collections API such as LinkedList, ArrayList, List, Stack, Queue, etc.*

- 1) Create Class **CharNode** with an attribute “**char myData**”, and create Class **CharList** that has at least two methods **insert()** and **delete()**.
- 2) Create Class **CharStack** that has at least two methods **push()** and **pop()**. This class must be implemented using **CharList**.
- 3) Create Class **CharQueue** that has at least two methods **enqueue()** and **dequeue()**. This class must be implemented using **CharList**.
- 4) Create Class **StackCheckBalancedParentheses** with an attribute “**String text**”. This class implements a method called **CheckBalancedParentheses()** to check whether “(“ is matched with “)” in **text**. The implementation of **CheckBalancedParentheses()** should be based on **CharStack**. For example, “sdb(erlj)((djre)lejr)” is balanced but “4+(5*4)” is unbalanced. The return value of **CheckBalancedParentheses()** is an integer that indicates the degree of unbalancing of **text**. For example,
 - a. **CheckBalancedParentheses** (“sdb(erlj)((djre)lejr)”) = 0 because the string is balanced
 - b. **CheckBalancedParentheses** (“4+(5*4)”) = 1 because there is an extra ‘)’.
 - c. **CheckBalancedParentheses** (“ere)(jre”) = 2 because there are extra ‘)’ and ‘)’.
 - d. **CheckBalancedParentheses** (“((()))”) = 3 because there are extra “(((“.
 - e. **CheckBalancedParentheses** (“(((()))”) = 4 because there are extra “((((“.

- 5) Create Class `QueueCheckBalancedParentheses` with an attribute "`String text`". This should implement a method called `CheckBalancedParentheses()` to check every "(" is matched with ")" in `text`. The implementation of `CheckBalancedParentheses()` should be based on `CharQueue`. (*Hint: you can use two queues to emulate a stack.*)
- 6) Create Class `StackQueueDemo` that takes a user string input (strings will be input from the file "balancedParenCheckInputs.txt" which will have various strings separated by blank lines). The main method should call `CheckBalancedParentheses()` of both `StackCheckBalancedParentheses` and `QueueCheckBalancedParentheses` classes, and make sure their outputs are consistent. No static members unless absolutely necessary (and if you use them, your report should justify their use)
- 7) Testing: test your program with several strings, design your test cases so that you cover most (if not all) cases of user-input (this implies your application should be robust and fail-safe as much reasonably as possible).

Design Requirements

Code Documentation

For this assignment, you must include documentation for your code as generated by Javadoc. You should have Javadoc comments for every class, constructor, and method. By default, Javadoc should output html documentation to a subfolder within your project (/dist/javadoc). Make sure this folder is included when you zip your files for submission. You do not need to submit a hard copy of this documentation.

Hint: <http://stackoverflow.com/questions/4468669/how-to-generate-javadoc-html-in-eclipse>

Coding Standards

You must adhere to all conventions in the CS 3310 Java coding standard. This includes the use of white spaces for readability and the use of comments to explain the meaning of various methods and attributes. Be sure to follow the conventions for naming classes, variables, method parameters and methods.

Testing

Make sure you test your application with several different values, to make sure it works.

Example input file: balancedParenCheckInputs.txt

"sdb(erlj)((djre)lejr)"

"4+(5*4)"

“ere)(jre”

“(((())(“

“)(((())(“

Assignment Submission

- Generate a .zip file that contains all your files, including:
 - Source code files
 - Including any input or output files
 - Documentation of your code – e.g. using Javadoc if using Java
 - A brief report (in a pdf file) on your observations of comparing theoretical vs empirically observed time complexities. Note this report will include (a) a brief description of problem statement(s), (b) algorithms descriptions (if these are standard, commonly known algorithms, then just mention their names along with customization to your specific solution(s), otherwise give the pseudo-code of your algorithms, (c) theoretically derived complexities of the algorithms used in your code, (d) table(s) of the observed time complexities, and (e) plots comparing theoretical vs. empirical along with your observations (e.g. do theoretical agree with your implementation, why? Why not?).

Don't forget to follow the naming convention specified for submitting assignments