Seoul National University

Data Structure

Fall 2020, Kang

Programming Assignment 1: Lists, Stacks, and Queues (Chapter 4)

Due: October 12, 23:59, submit at eTL

Reminders

- The points of this homework add up to 100.
- Like all homework, this has to be done individually.
- Lead T.A.: Tairen Piao (piaotairen@snu.ac.kr)
- Write a program in Java.
- Do not use Java Collection Framework and third-party implementation from the Internet.

1. How to submit the programming assignment

1) Create a ***JAR*** file including 'src' folder that contains your sources files but without 'release' folder. (Refer to '1 – Introduction.pptx' in the first lab session)

   We will run your ***Main*** class in the JAR file to grade your programming assignment. Before submitting the JAR file, please check if your Main class in the JAR file works correctly.

   You **MUST** obey the I/O specification of the programming assignment, and rules for the submission of the programming assignment.

   Before submitting, check if your JAR file runs properly in your terminal with the following commands:

   "`java -classpath PA_##_(StudentID).jar Main`".

2) Submit the jar file to the eTL (http://etl.snu.ac.kr/).

NOTE: You must import the "skeleton" directory when you start this assignment. Please check if you can read the "sample_input.txt" without any modification.

NOTE: If you want to check your jar file working, you have to make jar file and the input file located in the same directory (i.e. make the relative path of the input file to be "sample_input.txt")

2. How to grade your programming assignment

1) We made a grading machine to automatically grade your programming assignment. The machine will run your program and compare the answers and outputs that your program generates for given inputs. If your program cannot generate correct answers for an input file, it will not give you the point corresponding to the input. Our machine will consider the following scenarios:

   (***Accept***) When your program generates exact outputs for an input file, the machine will give you the point of the input.

   (***Wrong Answer***) When your program runs normally but generates incorrect outputs for an input file, including typos, the machine will not give you the point of the input.

   (***Run Error***) When your program does not run or is terminated suddenly for some reason, the machine will not give you the point of an input file because it cannot generate any outputs.

   (***Time Limit***) When your program runs over a predefined execution time for an input file, our machine will stop your program, and it will not give you the point of the input. The time limit of the execution is ***5 seconds***.

2) We will generate 10 input files, and assign 10 points for each input file. For example, if your program gets 9 accepts, and 1 wrong answer by the machine, the total point will be 90 points. <u>Hence, before submitting your programming assignment, please be sure that your program makes correct answers in a reasonable time for any input case.</u>

3. Problem

Stack and **Queue** are simple but very powerful data structures when you make applications.

1) **LinkedStack**

   A stack is an abstract data type that serves as a collection of elements, with the following main principal operations.

   - *Push*: Adds an element to the collection.
   - *Pop*: Removes the most recently added element.
   - *Clear*: Clear the stack.
   - *Length*: Return the length of the queue.
   - *isEmpty*: Determine if the stack is empty.

2) **LinkedQueue**

   A queue is a collection of entities that are maintained in a sequence and can be modified by the addition of entities at one end of the sequence and the removal of entities from the other end of the sequence, with the following main principal operations.

   - *Enqueue*: Adds an element to the collection.
   - *Dequeue*: Removes the oldest element in the collection.
   - *Clear*: Clear the queue.
   - *Length*: Return the length of the queue.
   - *isEmpty*: Determine if the queue is empty.

3) **Valid Parentheses**

   Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid **using stack that you implemented.**

   An input string is valid if:

   - Open brackets must be closed by the same type of brackets.
   - Open brackets must be closed in the correct order.

   Here are two small examples that help you understand the problem. (Detailed I/O descriptions are in section 6)

   Input: "()[]{}", **Input:** "[)"
   Output: true, **Output:** false

4. ADT of Data Structures

1) Stack

Function
void clear()
Description
- **This function removes all of the elements from the stack.**

Function
void push(E item)
Description
- **This function pushes an (item) onto the top of the stack.**

Function
E pop()
Description
- **This function removes the item at the top of the stack and returns that item as the value of this function.** - **If the stack is empty, this function returns null.**

Function
int length()
Description
- **This function returns the number of elements in the stack.**

Function
boolean isEmpty()
Description
- **This function returns true if the stack contains no elements, otherwise returns false.**

2) Queue

Function
void clear()
Description
- **This function removes all of the elements from the queue.**

Function
void enqueue(E item)
Description
- **This function inserts the specified element into the queue.**

Function
void dequeue()
Description
- **This function retrieves and removes the head of the queue.** - **if the queue is empty, this method returns null.**

Function
int length()

Description
- **This function returns the number of elements in the queue.**

Function
boolean isEmpty()
Description
- **This function returns true if the queue contains no elements, otherwise returns false.**

Function
void reverse()
Description
- **This function reverses the elements in the queue.**

3) Parentheses

Function
boolean isValid()
Description
- **This function returns if the input string is valid**

5. Specification

1) Stack

   You must implement the **linked** stack.

   Note that, we will not pop an element when a stack is empty.

2) Queue

   Like the stack, you must implement the **linked** queue.

   Note that, We will not dequeue an element when a queue is empty.

3) Valid parentheses

   Please confirm that your program can pass all test cases, and we will not give an empty string in this problem.

Important

You need to fill the blanks of functions in the LinkedQueue.java, LinkedStack.java, and Parentheses.java from the skeleton codes. Check the interface files (Queue.java and Stack.java) to see the description of functions should be implemented.

6. Specification of I/O

The program should accept only the inputs listed below and print the listed outputs. You **must** use standard I/O operations in Java, not file operations.

Sample Input	Sample Output
`stack,clear`	`clear success`
`stack,isempty`	`true`
`stack,push,1`	`push 1`
`stack,push,2`	`push 2`
`stack,isempty`	`false`
`stack,length`	`2`
`stack,pop`	`pop 2`
`stack,pop`	`pop 1`
`stack,isempty`	`true`

Sample Input	Sample Output
`queue,clear`	`clear success`
`queue,isempty`	`true`
`queue,enqueue,1`	`enqueue 1`
`queue,enqueue,2`	`enqueue 2`
`queue,isempty`	`false`
`queue,length`	`2`
`queue,dequeue`	`dequeue 1`
`queue,dequeue`	`dequeue 2`
`queue,isempty`	`true`

Sample Input	Sample Output
`isvalid,}}`	`false`
`isvalid,()()`	`true`
`isvalid,([)]`	`false`
`isvalid,()[]`	`true`
`isvalid,[`	`false`
`isvalid,{[]}`	`true`
`isvalid,(}`	`false`
`isvalid,()(]`	`false`