Seoul National University

Data Structure

Fall 2017, Kang

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

Due: October 12, 02:00 pm, submit at eTL

Reminders

- The points of this homework add up to 100.
- Like all homeworks, this has to be done individually.
- Lead T.A.: Beunguk Ahn (beunguk.ahn@gmail.com)
- Write a program in Java.
- Do not use Java Collection Framework and the 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 assignments. Before submitting the JAR file, please check if your Main class in the JAR file works correctly.
   - The submitted JAR should contain your sources files. If you don't use Eclipse, please see https://www.clear.rice.edu/comp201/05-spring/info/jar.shtml to create a proper JAR file.
   - 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 -jar ./PA_01_2015-12345.jar`".

2) Compress **the JAR file, a readme file and the project folder.** The readme file needs to include your student id, name, how to execute your program, and any other information TA needs to know.
   - The format of the JAR file and the readme file are *"PA_##_(StudentID).jar"/ "PA_##_(StudentID).txt"*, respectively, where '##' is the programming assignment ID, and *(StudentID)* is your student ID.
     - ex) PA_01_2016-12345.jar
     - ex) PA_01_2016-12345.txt
   - All documents should be written in English.

3) The name of the compressed file (zip file) should be '*PA_##_(StudentID).zip*' where '##' is the programming assignment ID, and (StudentID) is your student ID.
   - ex) PA_01_2016-12345.zip
     - ➢ *01: the first programming assignment*
     - ➢ *2016-12345: your student ID*

4) Submit the compress file to the eTL (http://etl.snu.ac.kr/) .

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 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 reasons, 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. Hence, before submitting your programming assignment, please be sure that your program makes correct answers in reasonable time for any input case.

3. Problem

Stack is one of the widely used data structures. It has two principle operations: *push* and *pop*. With these operations, the order of element come off a stack is also called LIFO (last in, first out).

- *Push*: Adds an element to the collection
- *Pop*: Removes the most recently added element that was not yet moved

Stack is useful when we calculate the sequence of numbers with parenthesis. For example, if we want to calculate (1 + 2 + 3) + (4 + 5), we need to evaluate (1 + 2 + 3) first then (4 + 5). Although the order of adding numbers does not affect the result, we need follow the order of operations anyway.

In this assignment, we will make calculator called “BabyCalculator”, which only handles parenthesis and digits (0, 1, …, 9) with only addition function using stack. Here are some examples.

- For input ‘123’, it means ‘1+2+3’.
- For input ‘(12(34))5’, it means ‘(1+2+(3+4)+5)’
- For input ‘12(2(02)2)’, it means ‘1+2+(2+(0+2)+2)

Inputs consists of parentheses and digits, and you can assume inputs are always syntactically correct. BabyCalculator must follow LIFO order on adding numbers. BabyCalculator also holds a current value as previous result. It means the calculator add to the current value when a new polynomial is given. (Please check sample input and output at the end of this document.).

Fill in the ‘LinkedList.java’, ‘LinkedStack.java’, ‘BabyCalculator.java’ and ‘Main.java’ in ‘PA01’ java project. First, implement LinkedList and LinkedStack. Then using the LinkedStack, implement BabyCalculator and Main.

Several rules that you **must** follow are as follows:

- Make your program run through Main class which uses Stack implemented in the first step. The main class should handle inputs and outputs using standard I/O in JAVA. (input from a keyboard, output to a monitor)

- You have to implement all functions listed in the following ADT and BabyCalculator.

4. ADT of Data Structures

1) List

Function

```
void clear()
```

Description

- This function removes all of the elements from the list.

Function

```
void insert(int pos, E item)
```

Description

- This function inserts the specified element at the specified position in the list.
- (pos) is an index at which the specified element is to be inserted.
- (item) is an element to be inserted.
- The range of (pos) is between 0 and (the current length of list).
- If (pos) is not valid, the function ignores the request.

Function

```
E remove(int pos)
```

Description

- This function removes the element at the specified position in the list, shifts any subsequent elements to the left, and returns the element that was removed from the list.
- (pos) is an index of the element to be removed.
- The range of (pos) is between 0 and (the current length of list - 1).

- If the value of (pos) is not valid, this function ignores the request and returns null.

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

Function
`E getValue(int pos)`
Description
- This function returns the element at the specified position in the list. - (pos) is an index of the element to return. - The range of (pos) is between 0 and (the current length of list - 1). - If (pos) is not valid, the function returns null.

2) 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.

5. BabyCalculator

1) BabyCalculator

Function
`int murmurAdd (String polynomial)`
Description

- (polynomial) is a sequence of parentheses and digits to evaluate - This function prints and returns (current value + evaluated polynomial).

Function
`int getValue ()`
Description
- Return current value of the calculator.

Function
`void setValue(int newValue)`
Description
- Set the current value of the calculator to newValue.

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.

1) ADD

Input form	**Output form**

`ADD (polynomial)`	`(currValue)[+(digits) … ]`
	`RESULT (new currValue)`

Description	
- value of (polynomial) will be added to a current value of the calculator. - (polynomial) doesn't contain whitespace characters. - (currValue) is the current value of the calculator. - [+(digits) …] is a sequence of popped out digits from the stack in the calculator. - (new currValue) is new value of the calculator.	
Example Input	**Example Output**
`ADD 134`	5+4+3+1

	`RESULT 13`
	
	`* (the first term 5 is current value of the calculator)`

2) SHOW

Input form	Output form
`SHOW`	`(currValue)`
Description	
- Show the current value of the calculator	
Example Input	**Example Output**
`SHOW`	`VALUE 11`

3) CLEAR

Input form	Output form
`CLEAR`	`VALUE CLEARED`
Description	
- This command clears (set to 0) the value.	
Example Input	**Example Output**
`CLEAR`	`VALUE CLEARED`

4) SET

Input form	Output form
`SET (newValue)`	`VALUE TO SET (newValue)`
Description	
- Set the value of the calculator to (newValue) - (newValue) does not have to be one digit	
Example Input	**Example Output**
`SET 24`	`VALUE SET TO 24`

5) EXIT

Input form	Output form
`EXIT`	`FINAL VALUE (currValue)`
Description	
- Show the current value and exit the program.	
Example Input	**Example Output**

`EXIT`	`FINAL VALUE 5`
	
	`* (5 is current value of the calculator)`

7. Sample Input and Output

Keep in mind the order of digits printed on ADD operations.

Sample Input	Sample Output
SHOW	VALUE 0
ADD 5	0+5
	RESULT 5
ADD (4123)	5+3+2+1+4
	RESULT 15
ADD 2((13)45)	15+3+1+5+4+2
	RESULT 30
ADD 2(37)	30+7+3+2
	RESULT 42
CLEAR	VALUE CLEARED
SHOW	VALUE 0
SET 10	VALUE SET TO 10
ADD 2((10)45)	10+0+1+5+4+2
	RESULT 22
EXIT	FINAL VALUE 22