Homework 3

Linked list, stack, queue

CS 5060 Intensive Programming, Fall 2012

85 points

Due: 3:59 pm September 26, 2012

Linked list, stack, queue (40 pts).

Implement a singly-linked list and use it to solve the following problems. Your `MyLinkedList` class should implement the `LinkedList` interface available in the Files section on Canvas.

Note: All input must be read from the standard input stream, and all output must be written to the standard output stream. For this assignment, assume the input is correct.

Problem 0a: Stack

Implement a linked list based stack using your singly-linked list implementation. Your `MyStack` class should implement the `Stack` interface available in the Files section on Canvas.

Input: The input begins with the number t of test cases in a single line ($t \leq 100$). Each of the next t lines starts with a number n ($1 \leq n \leq 1000$) followed by a list of n operations. There are two types of operations: push and pop. A push operation is represented by a string "`push`" followed by a number m ($1 \leq m \leq 1000000$) to push into the stack. A pop operation is represented by a string "`pop`".

Output: For each test case output the elements in the stack after all the operations have been performed. Consecutive elements should be separated by a single space.

Example:

Input:

```
2
4 push 5 push 0 push 6 push 0
7 push 1 push 2 pop push 3 push 4 pop push 5
```

Output:

```
5 0 6 0
1 3 5
```

Problem 0b: Queue

Implement a linked list based queue using your singly-linked list implementation. Your `MyQueue` class should implement the `Queue` interface available in the Files section on Canvas.

Input: The input begins with the number t of test cases in a single line ($t \leq 100$). Each of the next t lines starts with a number n ($1 \leq n \leq 1000$) followed by a list of n operations. There are two types of operations: enqueue and dequeue. An enqueue operation is represented by a string "`enq`" followed by a number m ($1 \leq m \leq 1000000$) to put into the queue. A dequeue operation is represented by a string "`deq`".

Output: For each test case output the elements in the queue after all the operations have been performed. Consecutive elements should be separated by a single space.

Example:

Input:

```
2
4 enq 5 enq 0 enq 6 enq 0
7 enq 1 enq 2 deq enq 3 enq 4 deq enq 5
```

Output:

```
5 0 6 0
3 4 5
```

Grading: The linked list is worth 10 pts. The stack and the queue are worth 15 pts each (if you don't use the linked list you get 5 pts only).

For each data structure, make sure you use the interface name when declaring instances (not the class name). For example, you should write:

```
LinkedList myList = new MyLinkedList();
Stack myStack = new MyStack();
Queue myQueue = new MyQueue();
```

This is worth 2 points in each case.

For each data structure you should use the `toString()` method to output the result. This is worth 2 pts in each case.

Solve the following problems (45 pts).

Each problem is worth 10 points. All of the problems can be solved by using a linked list, a stack, or a queue. There are 5 extra points for solving the problems using the data structures implemented in problem 0 (1 point for each problem plus 1 extra point if you use the data structures for all the problems).

Note: All input must be read from the standard input stream, and all output must be written to the standard output stream. For this assignment, assume the input is correct.

Problem 1: Josephus

There are n people, numbered $1, 2, ..., n$, in a circle. At each step, m people are skipped and the next person is executed, until there is only one person remaining. Help Josephus determine the position where he should be in to save his life. You can read more about this problem at
`http://en.wikipedia.org/wiki/Josephus_problem`.

Input: The input begins with the number t of test cases in a single line ($t \leq 100$). Each of the next t lines has two numbers: the number of people n ($1 \leq n \leq 1000$) and the number of people k ($1 \leq k \leq 1000$) to skip at each step.

Output: For each test case output the position where Josephus should be in to save his life.

Example:

Input:

```
2
5 1
5 2
```

Output:

```
3
4
```

Problem 2: Balance

A string S consisting of parenthesis `()` and square brackets `[]` is balanced if:

- S is the empty string,
- $S = AB$ and both A and B are balanced,
- $S = (A)$ or $S = [A]$.

Input: The input begins with the number t of test cases in a single line ($t \leq 100$). Each of the next t lines contains a string of length ℓ ($1 \leq \ell \leq 1000$) formed with parenthesis and square brackets.

Output: For each test case determine if the string is balanced.

Example:

Input:

```
3
([])
()[]
([)]
```

Output:

```
YES
YES
NO
```

Problem 3: Stack or queue?

Input: The input begins with the number t of test cases in a single line ($t \leq 100$). Each of the next t lines starts with a number n ($1 \leq n \leq 1000$) followed by a list of n operations. There are two types of operations that can be performed on an unknown data structure (stack or queue): add and remove. An add operation is represented by a string "`add`" followed by a number m ($1 \leq m \leq 1000000$) to add to the data structure. A remove operation is represented by a string "`rem`" followed by the number m ($1 \leq m \leq 1000000$) removed from the data structure.

Output: For each test case determine whether the data structure is a stack or a queue. If it is a stack output "`STACK`", if it is a queue output "`QUEUE`", and if it is not possible to know output "`UNKNOWN`".

Example:

Input:

```
4
4 add 5 add 0 add 6 add 0
6 add 1 add 2 add 3 rem 3 rem 2 rem 1
6 add 1 add 2 add 3 rem 1 rem 2 rem 3
4 add 2 add 2 rem 2 rem 2
```

Output:

```
UNKNOWN
STACK
QUEUE
UNKNOWN
```

Problem 4: RPN expression

Read about *Reverse Polish Notation* (RPN) at
`http://en.wikipedia.org/wiki/Reverse_Polish_notation`.

Input: The input begins with the number t of test cases in a single line ($t \leq 100$). Each of the next t lines contains an RPE expression. The expression only has integer numbers and it can have any of the following binary operators: `+, -, *, /`. The division is integer division (there is no division by zero).

Output: For each test case output the result of evaluating the expression (the result could be negative).

Example:

Input:

```
3
7 2 - 3 *
7 2 * 3 -
3 8 - 2 /
```

Output:

```
15
11
-2
```

Submission.

Submit a `zip` file with the following files:

1. Code files `MyLinkedList.java`, `MyStack`, and `MyQueue` with your implementations of the data structures; and code files `MyStackTest.java` and `MyQueueTest` with the solution to problems 0a and 0b. (do not submit the interface files).
2. Four code files `Josephus.java`, `Balance.java`, `StackOrQueue.java` and `RPNExpression.java` with the solutions to problems 1, 2, 3,and 4, respectively.

Include your name and A number at the top of each source file. Name the `zip` file `hw03_firstName_lastName.zip`. For example, if your name is John Smith, name the file `hw03_John_Smith.zip`.