Question 2 - Tokenizer, Parser [30 marks]

The theme of this question is developing a simple parser for LOGO programming language. LOGO controls the commands for movement and drawing of a pointer on the screen.

We consider a grid with 11 x 21 cells, where 11 is the number of rows and 21 is the number of columns. A pointer is represented by one of the following characters:

- “^”: The pointer is pointing to the upward direction
- “>”: The pointer is pointing to the right direction
- “<”: The pointer is pointing to the left direction
- “v”: The pointer is pointing to the downward direction

We can control the movement and drawing of the pointer by the following commands:

- **LEFT**: Turn the direction of the pointer by 90 degrees to the left
- **RIGHT**: Turn the direction of the pointer by 90 degrees to the right
- **PENUP**: Set the status of the pointer to be leaving no trail, when it moves
- **PENDOWN**: Set the status of the pointer to be leaving a trail, when it moves
- **FORWARD**(n): Move the pointer along the direction it is pointing by n cells
- **BACK**(n): Move the pointer in the reverse direction it is pointing by n cells

The language grammar of simplified LOGO is given by:

<Command> := **LEFT** | **RIGHT** | **FORWARD**(<num>) | **BACK**(<num>) | **PENUP** | **PENDOWN**

<Exp> := <Command>; <Exp> | <Command>;

where <num> is an integer literal.

Example:

- Initial screen: (the pointer is always initially positioned with upward direction at the center of the grid with PENUP status)

```
#####################
#####################
#####################
#####################
#####################
##########^##########
#####################
#####################
#####################
#####################
#####################
```

- Input:

  PENUP; **LEFT**; **FORWARD**(10); **PENDOWN**; **RIGHT**; **BACK**(5);

- Output:

```
#####################
#####################
#####################
#####################
#####################
.####################
.####################
.####################
.####################
.####################
^####################
```

where an empty cell is represented by character "#", a cell with the trail of the pointer is represented by character "`.`", and the pointer is initially with PENUP status. Note that you do not need to consider cases where FORWARD(n) and BACK(n) can go beyond the boundary of the grid.

Tasks:

1) [2 marks] Setup and use git within the Q2 directory for revision control and make at least 3 commits.
2) [3 marks] Write a correct Makefile for building the program with 3 targets: "*make*", "*make clean*", and "*make rundemo*", which will compile, delete the compiled class files and run "Demo.java" respectively.
3) [10 Marks] Complete "*`next`*()", "*`takeNext`*()" and "*`hasNext`*()" methods in "Tokenizer.java" to extract an expression into tokens.
4) [10 Marks] Complete "*`parse`*()" method in "Parser.java" to parse an input expression by computing the final position of the pointer and marking the pointer movement on the screen.
5) [5 Marks] Complete "*`trace`*()" method in "Screen.java" to return a string showing the trail of the pointer, its current position and direction.

For 2) - 5), you may only modify the required methods within Q2 directory. Do not modify anything else.

Question 3 - Binary Search Tree [20 marks]

Your Q3 directory contains code that implements a binary search tree with a set of string keys. The implementation has had the code for “*find*”, “*insert*”, and “*printOddNodes*” removed.

Consider a comparison of keys by alphabetical order. For example, “a” < “b”. “aaab” < “ab”. You only need to consider the lower-case alphabet.

You are required to complete the implementation replacing the missing code. Your answer must be placed in your Q3 directory.

Tasks:

1) [7 marks] Implement the “*find*” method.
2) [7 marks] Implement the “*insert*” method.
3) [6 marks] Implement the “*printOddNodes*” method to print the pre-order traversal of the nodes that have an odd number of direct children.

You may only modify the required methods within Q3 directory. Do not modify anything else.

Question 4 - Testing [20 marks]

Your Q4 directory contains code that implements two useful utilities. A parseInt algorithm to convert a string to an integer is implemented in *MyInteger.java*, and a method for replacing multiple newline characters by a single newline character is implemented in *StringUtil.java*.

Tasks:

1) [10 marks] Your first task for this question is to implement a **minimum number** of JUnit test cases for *MyInteger.parseInt* that is **code complete**. Write your test case(s) in the one *test*() method in *MyIntegerTest.java.* Use *assertEquals* and *assertThrows* to check the correctness of the implementation.

   Hint: This is the method signature of *assertThrows*:

   ```
   assertThrows(Class<T> expectedType, Executable executable)
   ```

2) [10 marks] Your second task for this question is to implement a **minimum number** of JUnit test cases for *StringUtil.collapseNewLines* that is **branch complete**. Write your test case(s) in the one *test*() method in *StringUtilTest.java.* Use *assertEquals* to check the correctness of the implementation.

Note that ALL test cases should be written in the one *test*() method in each of the two test classes. Any additional test methods created elsewhere in your answer will not be marked.