95-771 – Data Structures and Algorithms for Information Processing
Project 3

Due Monday, October 14 Midnight 11:59:59 PM

The material covered in this project will appear on the midterm exam.

Topics: Stacks, Red Black Trees and Reverse Polish Notation (RPN)

(1) 30 Points. Write a stack class called DynamicStack.java. It will be implemented in an array with a top index initially set to -1. Each push operation will add one to the top index and then add a new element at that location. Each pop operation will return the value pointed to by the top pointer and it will decrease the top pointer by 1. The stack will hold Java objects. Thus, it will be able to contain such Java objects as Strings and BigIntegers. You should also provide a method called isEmpty. This method returns true if and only if the top index is -1.

The array, within which the stack "lives", must grow if it needs to. The array will begin with a size of 6. If the array is full and a push operation is executed, create a new array of twice the size as the old and copy the elements within the old array over to the new array. In this way, we will run out of stack space only when we have too little memory to accommodate this doubling of capacity. In this program, we will not be using that much stack space (that is, not enough stack space that would overwhelm the heap manager in the JVM) and so overflow will not be treated as a concern. When the stack grows smaller, there is no need to copy data over to a smaller array. Our array will only grow as needed – never shrink.

Other methods may be added to your stack class as needed. All additional methods will preserve class invariants (which should be described with comments) and will be true to the nature of a stack.

Your DynamicStack class will have main routine that tests it. The test will include a loop that pushes 1000 values to the stack. Another loop will pop and display all 1000 values pushed.

Within your DynamicStack code, describe the worst and best case behavior of your push operation – in terms of Big Theta.

(2) 40 Points. Recall the Red Black Tree that you wrote in Project 2. The tree was used to hold a set of words for spell checking. The CLR pseudocode provided for the insertion and lookup of a single key. In this project, your solution will provide for the insertion and lookup of a key, value pair. The key will always be a Java String (representing the name of a variable) and the value will always be a Java BigInteger. You will need to modify your Red Black Tree appropriately.

Provide a main routing that tests your new Red Black tree. The test code will add key and value pairs to the tree. It will perform lookups for the key, retrieving the value – a BigInteger. Note, the tree is ordered on the key (a String) and not on the value (the BigInteger). Each key in the tree will be unique – no duplicate keys are permitted. If you insert twice with the same key, the second will simply overwrite the first.

Your main routine (for testing) should add twenty String, BigInteger pairs to the tree. The first string will be the string "var1" with value 1. The second string will be the string "var2" with value 2 and so on. It will then search the tree for two of the BigIntegers (given the keys). It will then display the value of the BigIntegers found.

(3) 30 Points. Write a class called ReversePolishNotation.java that reads and then evaluates postfix expressions involving BigIntegers and variables. It reads a line from the user, evaluates the expression and displays the result. It continues to do this until the user hits the return key with no input or it encounters an error. We will use the standard binary operators (+, -, *, / , %, =) with their usual meanings. The assignment operator "=" requires that the left hand side of the expression be a variable. We will also use a unary minus. The unary minus will be represented with the tilde character "~". Finally, we will use a ternary operation - powerMod. powerMod computes x to the y modulo z. It will be represented by the circumflex character "^". All results will be integers. Below is an example execution. User input is in red. The program's output is in black.

```
java ReversePolishNotation
1  2  +
3
1 99999999999999999999999 +
100000000000000000000000
2 4 *
8
1 3 /
0
1 3 %
1
13 10 %
3
10000000000000000000 2 /
5000000000000000000
99999999999999999999 ~
-99999999999999999999
3 4 5 ^
1
12 2 3 ^
0
12 ~ ~ ~ ~ 2  3  ^
```

```
0
<return>
terminating
```

The example execution above only needs a stack of BigIntegers (and the BigInteger API provided by Java). Your program will also use the Red Black Tree that you wrote in (2) to store and retrieve variables and their values. Here is another execution that uses variables.

```
java ReversePolishNotation
x 4 =
4
y 5 =
5
x y +
9
x x 20 + =
24
lowerVal 1 =
1
upperVal 10 =
10
interval upperVal lowerVal - 1 + =
10
```

Note that the assignment operator returns the value assigned. So, we can run the following:

```
a 4 = 2 +
6
```

Note too that a value or variable may be entered by itself and its value will be printed:

```
a 4 = 2 +
6
a
4
```

With respect to error handling, you may assume that the user always enters a space between tokens. You may also assume that all integer values are entered correctly and that all variable names begin with a letter. The only error checking that you are required to do is throw an exception and halt the program when the stack underflows, a variable does not appear on the left of an assignment (a so-called “lvalue violation”), or a variable that has not been given a value is being dereferenced.

Here are three examples:

```
a 1 =
1
a b +
```

Exception in thread "main" java.lang.Exception: error: no variable b
2 3 ^
Exception in thread "main" java.lang.Exception: error: stack underflow exception
3 4 =
Exception in thread "main" java.lang.Exception: error: 3 not an lvalue

Finally, be sure that your calculator works for large integer input:

coolBigNum 999999999999999999999999999999999999 =
999999999999999999999999999999999999
anotherBigNum 1 coolBigNum + =
1000000000000000000000000000000000000
c coolBigNum anotherBigNum + =
1999999999999999999999999999999999999

Summary

Submit a single zipped file to Canvas. The zipped file will be named <yourAndrewID>Project3.zip. A complete project will contain at least the following files:

DynamicStack.java
RedBlackTree.java (uses DynamicStack)
ReversePolishNotation.java
At least two screenshots showing example runs