Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 3 REPORT

YUSUF PATOGLU
151044027

1 INTRODUCTION

1.1 Problem Definition

For the first part we had to find component labels in a given matrix. Simply, we have to count the component label counts in a given matrix. There are couple ways to do that. Most popular way is using recursion, but it's forbidden in homework. So I had to mimic that recursive traverse method somehow.

For the second part we needed to implement simple calculator that calculates expression. Expression input is in human readable format. It's also contains sin, cos and abs functions. These are the keys making the assignment tricky.

1.2 System Requirements

In order to execute my solution you need to have an essential JVM. You can run my Project in Eclipse or Intellij IDEA or whatever you want. It doesn't use any third party software libraries. If your computer can run a simple Java program, surely it will run my solution.
Just in case JVM system requirements are noted here on Java's website:

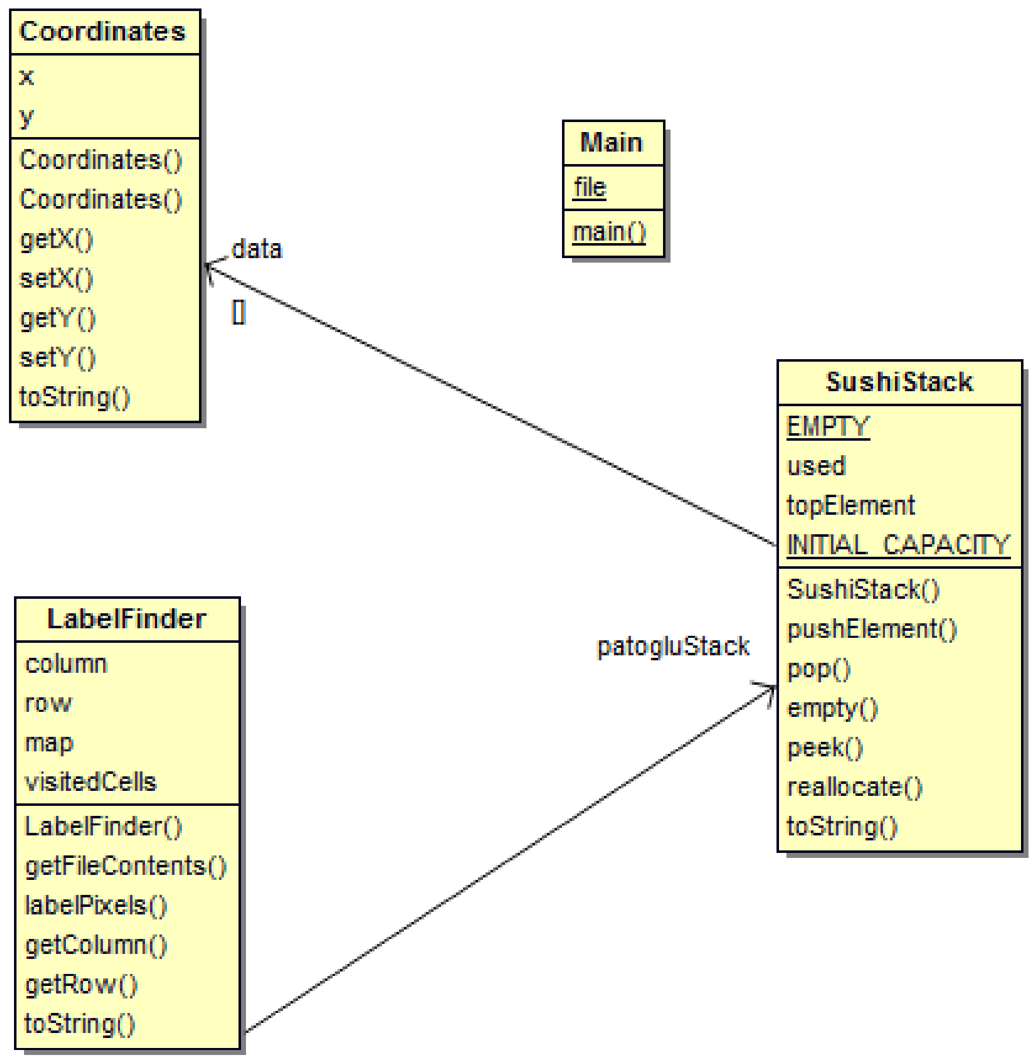
The minimum and recommended memory and disk space requirements are as follows:

- Minimum memory: 1 GB
- Recommended memory: 2 GB for Windows platforms, 1 GB for non-Windows platforms
- Minimum disk space: 250 MB
- Recommended disk space: 500 MB

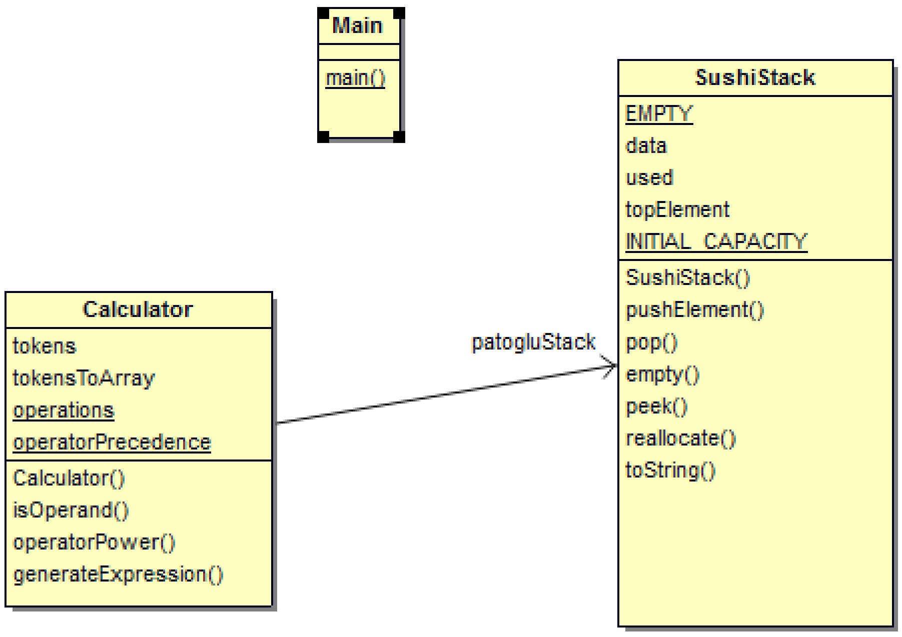
2 METHOD

2.1 Class Diagrams

For Part1:



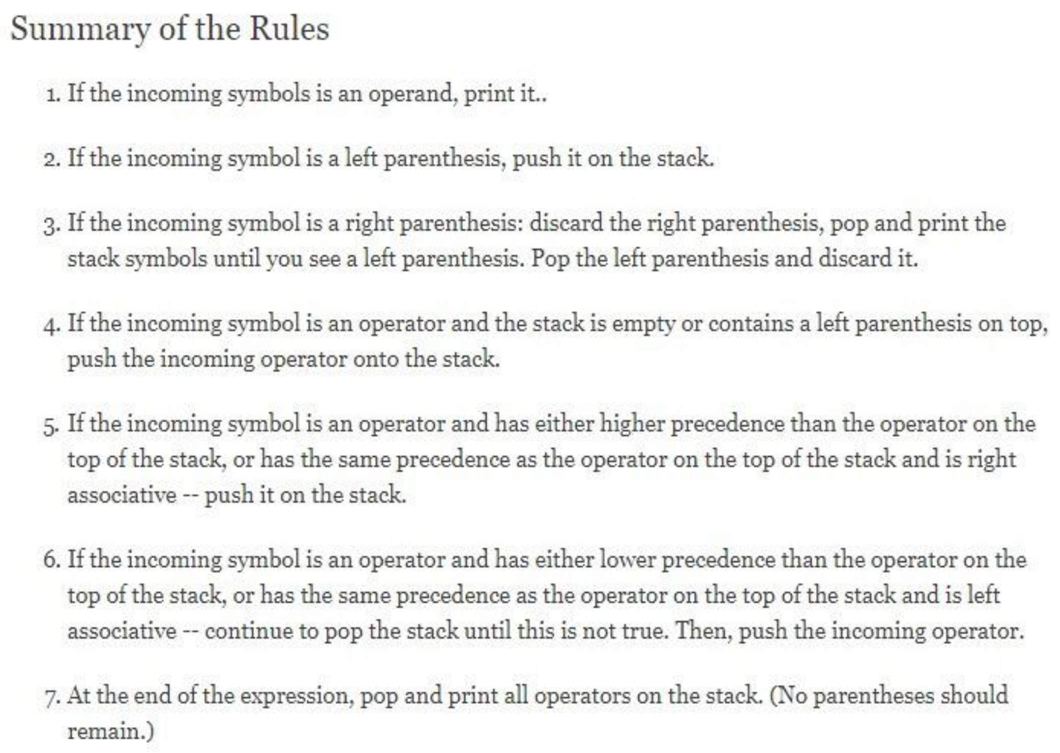
For Part2:


Main
main()
SushiStack
EMPTY
data
used
topElement
INITIAL_CAPACITY
SushiStack()
pushElement()
pop()
empty()
peek()
reallocate()
toString()
Calculator
tokens
tokensToArray
operations
operatorPrecedence
Calculator()
isOperand()
operatorPower()
generateExpression()
patogluStack

2.2 Problem Solution Approach

For part1 I had to traverse all matrix without using good of recursion. Since this week we finished with stacks it was obvious to use stacks to solve problem. Firstly I understand the depth first search recursive method. Then I mimicked recursions runtime stack with an expilicit stack. In order to do that I wrote my own stack data structure named with Sushi Stack(With the help of Koffman Data Structures book stack implementation.) Instead calling functions over, over and again I used my own stack as a physical memory. With this way I achived to mark all 1's in given matrix and counted label number. If we assume the matrix length as n my complexity will be O(n^2) in worst case.

For part2 simply I converted infix form to postfix form. I didn't get any help from Koffman book. I researched more robust algorithm for that. I've found algorithm named with Shunting Yard Algorithm. I applied all these steps in my code:



Summary of the Rules

1. If the incoming symbols is an operand, print it..
2. If the incoming symbol is a left parenthesis, push it on the stack.
3. If the incoming symbol is a right parenthesis: discard the right parenthesis, pop and print the stack symbols until you see a left parenthesis. Pop the left parenthesis and discard it.
4. If the incoming symbol is an operator and the stack is empty or contains a left parenthesis on top, push the incoming operator onto the stack.
5. If the incoming symbol is an operator and has either higher precedence than the operator on the top of the stack, or has the same precedence as the operator on the top of the stack and is right associative -- push it on the stack.
6. If the incoming symbol is an operator and has either lower precedence than the operator on the top of the stack, or has the same precedence as the operator on the top of the stack and is left associative -- continue to pop the stack until this is not true. Then, push the incoming operator.
7. At the end of the expression, pop and print all operators on the stack. (No parentheses should remain.)

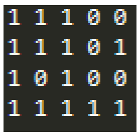
My goal was adding special functions like sin, cos and abs but I couldn't make that. Things got messy when nested sin functions appeared. I lost in if and else statements so I decided to leave the running version of it atleast.

3 RESULT

3.1 Test Cases

For Part1:

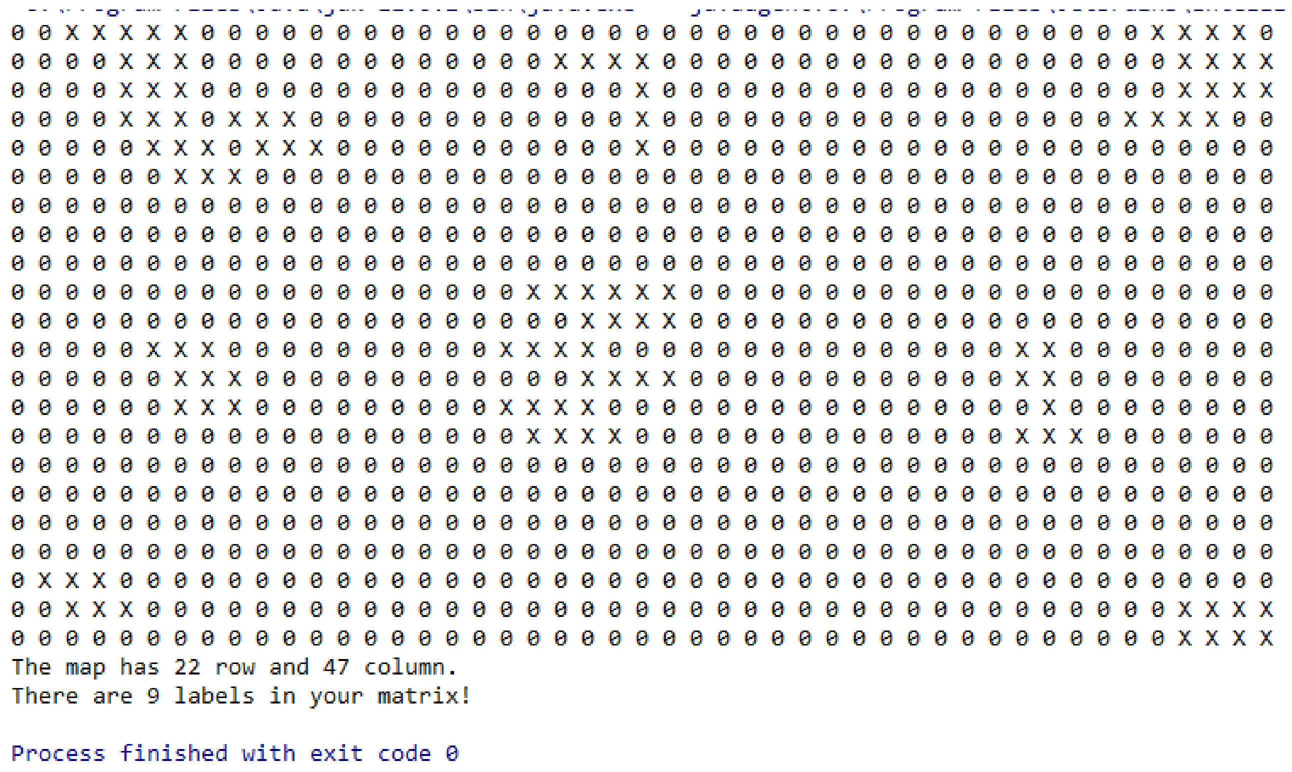
I’ve tested my code with several matrices.



```
1 1 1 0 0
1 1 1 0 1
1 0 1 0 0
1 1 1 1 1
```

Also I tested it with an empty matrix.

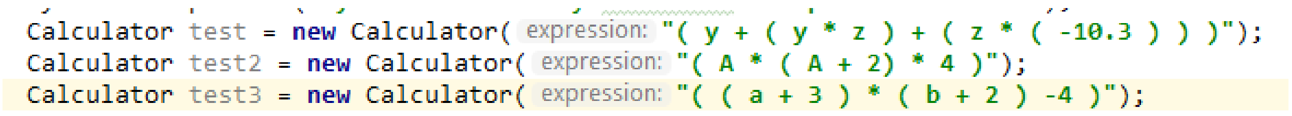
This is the input matrix that given to us:



```
0 0 X X X X X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X X X X 0
0 0 0 0 X X X 0 0 0 0 0 0 0 0 0 0 0 0 0 X X X X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X X X X
0 0 0 0 X X X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X X X X
0 0 0 0 X X X 0 X X X 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X X X X 0 0
0 0 0 0 0 X X X 0 X X X 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 X X X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X X X X X X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X X X X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 X X X 0 0 0 0 0 0 0 0 0 0 X X X X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X X 0 0 0 0 0 0 0 0
0 0 0 0 0 0 X X X 0 0 0 0 0 0 0 0 0 0 0 0 X X X X 0 0 0 0 0 0 0 0 0 0 0 0 X X 0 0 0 0 0 0 0 0
0 0 0 0 0 0 X X X 0 0 0 0 0 0 0 0 0 X X X X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X X X X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X X X 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 X X X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 X X X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X X X X
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X X X X
The map has 22 row and 47 column.
There are 9 labels in your matrix!

Process finished with exit code 0
```

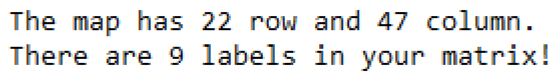
Part2 TestCases:



```
Calculator test = new Calculator( expression: "( y + ( y * z ) + ( z * ( -10.3 ) ) )");
Calculator test2 = new Calculator( expression: "( A * ( A + 2) * 4 )");
Calculator test3 = new Calculator( expression: "( ( a + 3 ) * ( b + 2 ) -4 )");
```

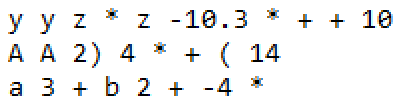
3.2 Running Results

Part1:



```
The map has 22 row and 47 column.
There are 9 labels in your matrix!
```

Part2:



```
y y z * z -10.3 * + + 10
A A 2) 4 * + ( 14
a 3 + b 2 + -4 *
```

References:

Koffman Data Structures Book for lecture.