Gebze Technical University
Computer Engineering

CSE 222 - 2019 Spring

HOMEWORK 03 REPORT

Muhammed ÖZKAN
151044084

Course Assistant:Özgü Göksu

1 INTRODUCTION

1.1 Problem Definition

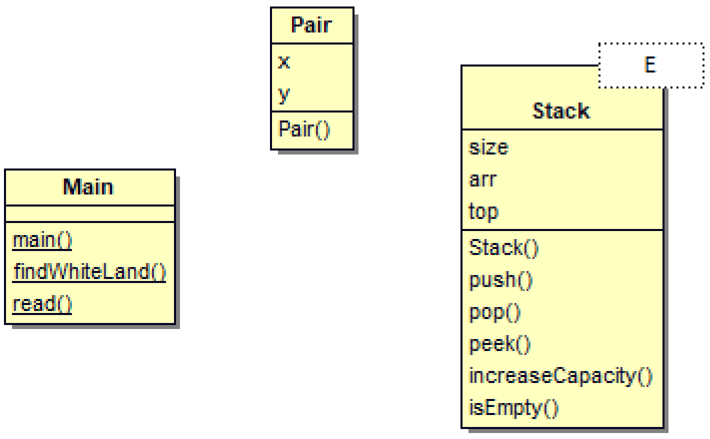
There are two problems we need to solve in this assignment. First, find out how many white areas are in an image represented by a binary number base from a text file. In doing this, we do not use the recursive algorithm. We calculate these operations using the Stack structure. Second problem we need to solve is to convert an infix expression and variables into a postfix expression format using a stack structure and then perform the numerical calculation of the expression.

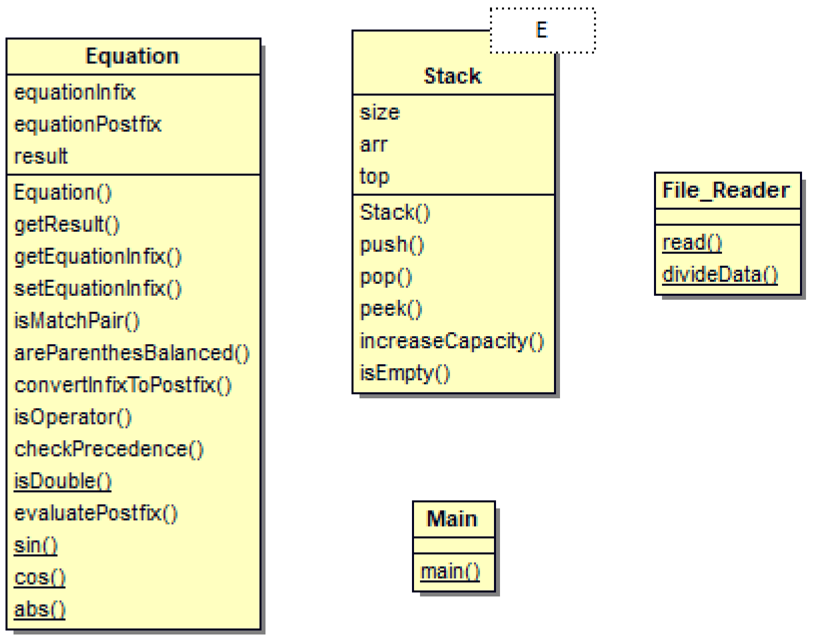
1.2 System Requirements

We will develop this program for all devices running Java. The classes used in the solution of the problems have been developed considering the minimum possible memory consumption. The size of the stack class on memory is to vary depending on the type of data to keep. Memory size shows a linear increase. In case of proper use, the prepared programs can be used in any environment, even on a smartphone or even on a smart watch.

2 METHOD

2.1 Class Diagrams





2.2 Use Case

The both software works on the console screen. The user must specify the path of the files to be used as the parameter to the program before running the programs.

For example:

-java program_name C:\test_file.txt

2.3 Problem Solution Approach

The stack data structure has been preferred because of the necessity to use this in the solution of the problems. In the first problem, I had to visit each element of the matrix only once. Therefore I used the stack structure to hold the neighbors of these elements then I visited the other members of this stack structure. In the second problem, I wanted to convert the mathematical expression to postfix and calculate the result. Here, I used the stack structure to keep the operators in the conversion process, and in the calculation part to keep the numeric data.

2.4 Complexity of Functions

The complexity of functions is calculated according to the number and structure of the loops they contain. Since the complexity calculations are considered infinite, the comparison, assignment and similar operations within the functions are not included in the calculations since they do not have any meaning in infinity.

Function Name	Complexity	Big O Notation
Stack.push()	$T_1(n)=1 \quad =1$	O(1)
Stack.pop()	$T_2(n)=1 \quad =1$	O(1)
Stack.isEmpty()	$T_3(n)=1 \quad =1$	O(1)
Stack.peek()	$T_4(n)=1 \quad =1$	O(1)
Stack.increaseCapacity()	$T_5(n)=n \quad =n$	O(n)
findWhiteLand()	$T_6(n)=n*m \quad =n*m$	O(n*m)
Equation. areParenthesBalanced()	$T_7(n)=n \quad =n$	O(n)
Equation. convertInfixToPostfix()	$T_8(n)=n+n+n \quad =3n$	O(n)
Equation.evaluatePostfix()	$T_8(n)=n \quad =n$	O(n)

3 RESULT

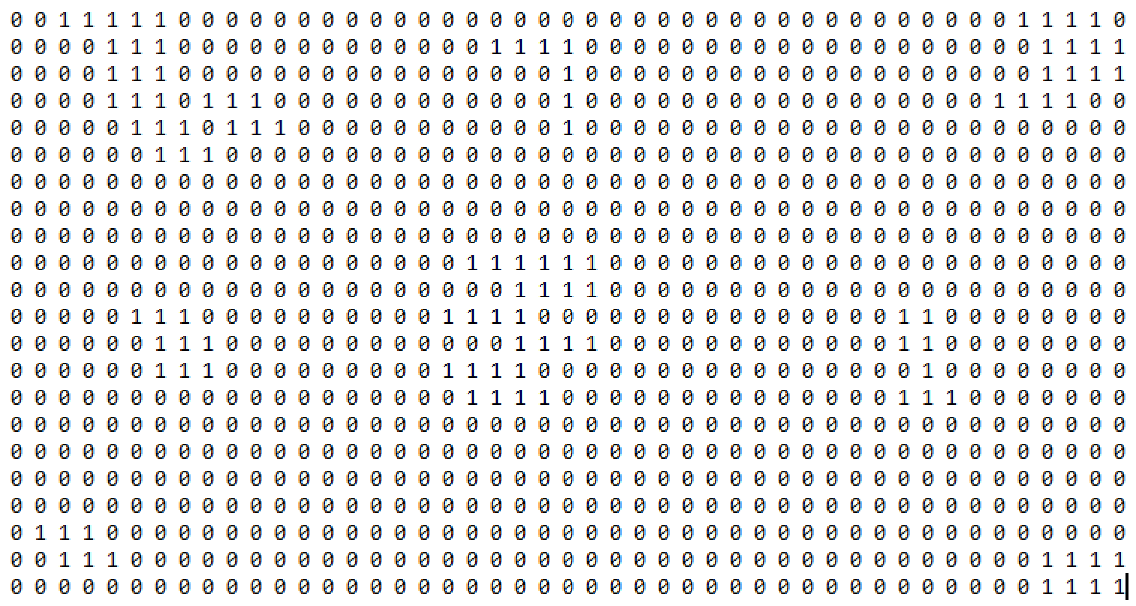
3.1 Test Cases

I tested the program with matrices of various sizes. These matrices included the worst case and the best cases. I tested the program with a 1 * 1 matrix. then 2 * 2, 3 * 3 4 * 4, etc., after testing all the probabilities in these matrices, I proceeded randomly on the larger matrix. As a result of these tests, each time the program gives the correct result, I have concluded that the program works correctly on any binary matrix.

Some types of matrix tested

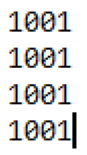
0000	1111	0000	1010	1111	1111	1001	0101	.....
0000	1001	0110	0101	1111	0000	1001	1010	.....
0000	1001	0110	1010	1111	1111	1001	0101	.....
0000	1111	0000	0101	1111	0000	1001	1010	.....
0	1	1	8	1	2	2	8	Result

3.2 Running Results



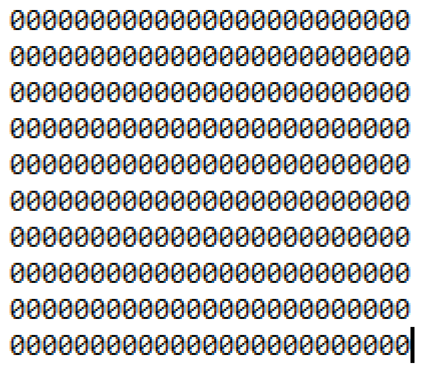
```
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 0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

Result is 9



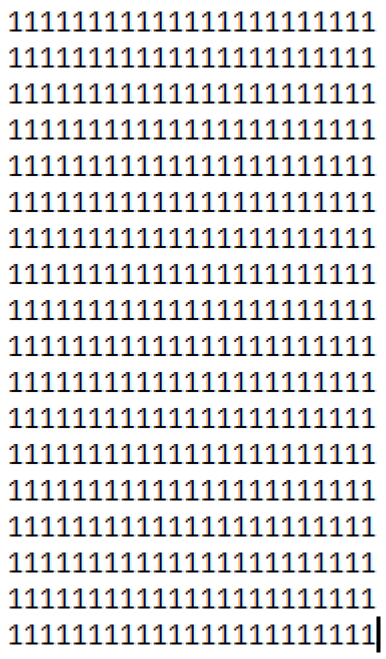
```
1001
1001
1001
1001
```

Result is 2



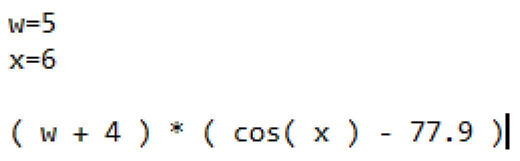
```
0000000000000000000000000
0000000000000000000000000
0000000000000000000000000
0000000000000000000000000
0000000000000000000000000
0000000000000000000000000
0000000000000000000000000
0000000000000000000000000
0000000000000000000000000
0000000000000000000000000
```

Result is 0



```
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
```

Result is 1



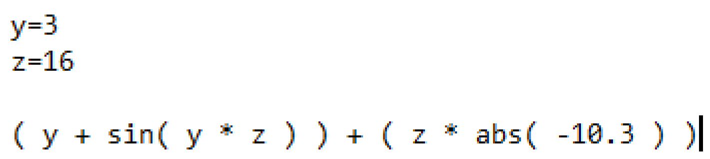
```
w=5
x=6

( w + 4 ) * ( cos( x ) - 77.9 )
```

Infix Form = (w + 4) * (cos(x) - 77.9)

Postfix Form = 5.0 4 + 6.0 c 77.9 - *

Result is -692,149303



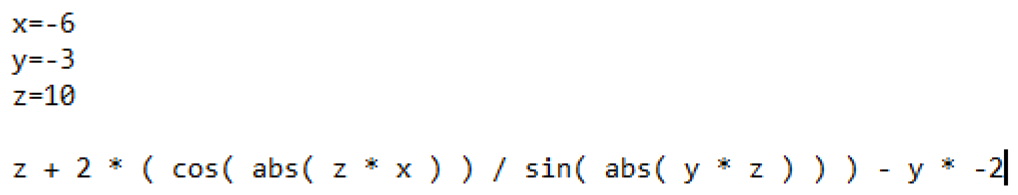
```
y=3
z=16

( y + sin( y * z ) ) + ( z * abs( -10.3 ) )
```

Infix Form = (y + sin(y * z)) + (z * abs(-10.3))

Postfix Form = 3.0 3.0 16.0 * s + 16.0 -10.3 a * +

Result is 168,543145



```
x=-6
y=-3
z=10

z + 2 * ( cos( abs( z * x ) ) / sin( abs( y * z ) ) ) - y * -2
```

Infix Form = z + 2 * (cos(abs(z * x)) / sin(abs(y * z))) - y * -2

Postfix Form = 10.0 2 10.0 -6.0 * a c -3.0 10.0 * a s / * + -3.0 -2 * -

Result is 6,000000