Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 3 REPORT

FURKAN ÖZEV
161044036

Course Assistant: Özgü GÖKSU

1 INTRODUCTION

1.1 Problem Definition

Part 1:

In this part, There is a binary digital image is represented through a matrix of integers, each element of which is either 1 or 0. 1 represent the white, 0 represent the black. Each sequential white pair that is contiguous, indicate that the path continues. We're asked to figure out how many different white paths are. We need to check if every element is white. Then, we need to check the white element's neighbors (through their top, left, right or bottom neighbor) to determine if the path continues. We have to repeat the same process for the neighbors. So, we need to visit every matrix element at least once in order to accomplish this goal. But we also want the time complex to be low. We need to develop an algorithm that can find out how many different white paths in the least complexity and use the stack data structure. In addition, we have to write the data structures ourselves.

Part2 :

In this part, We need a program that evaluates an infix expression. In order to do this, we first need to convert the infix expression to postfix and then calculate it. We will assume that the expression only consists of spaces, operands, and operators. The space character will be used as a delimiter between tokens.. All operands that are numbers begin with a digit ; all operands that are identifiers begin with a letter or some character. The ability to convert expressions with parentheses is important in postfix conversion. Parentheses are used to separate expressions into subexpressions. Then we need evaluate postfix expression. The postfix expression will be a string containing digit characters and operator form. Same way, the space character will be used as a delimiter between tokens. We need to use the stack data structure when writing the algorithms that do these operations. In addition, we have to write the data structures ourselves.

1.2 System Requirements

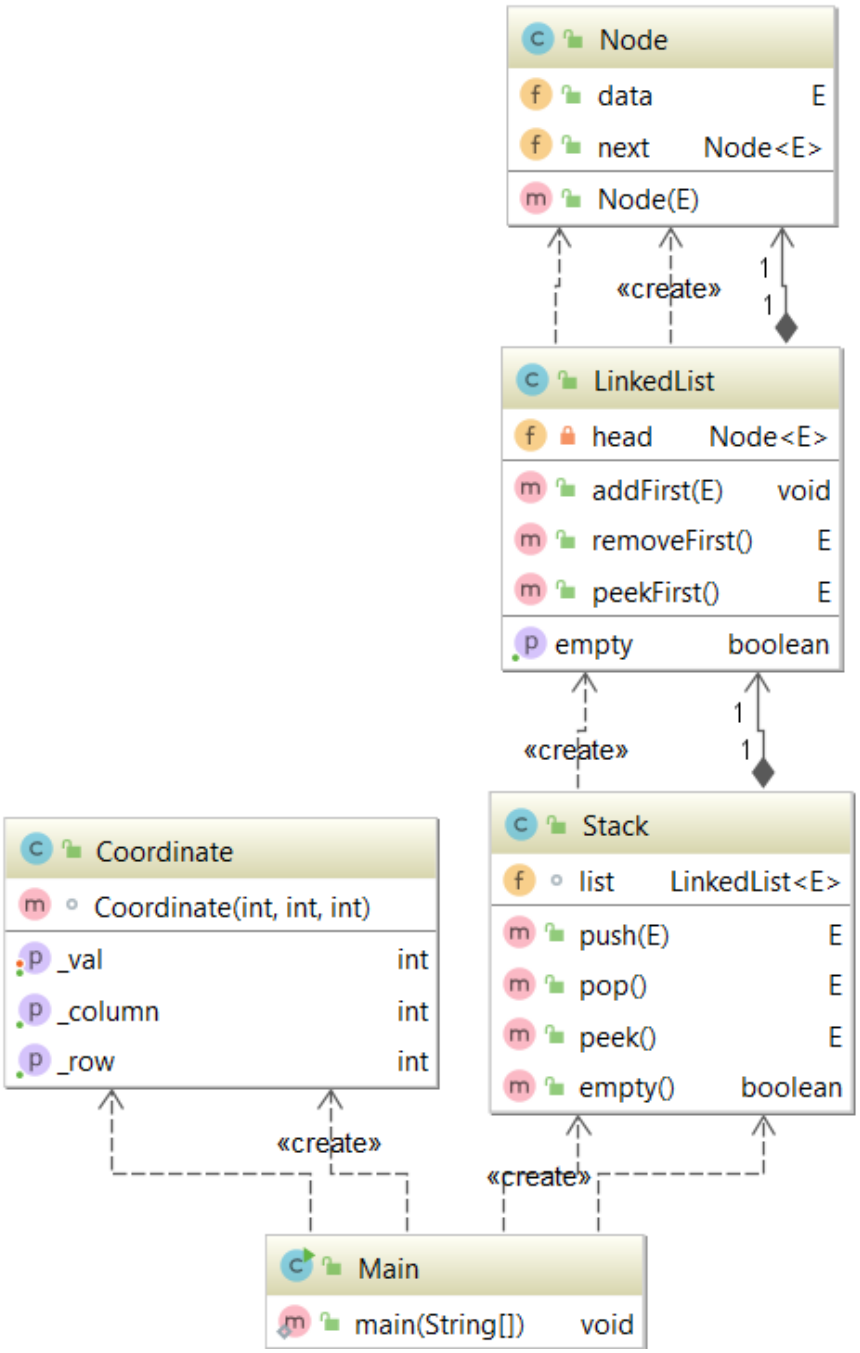
You must have java installed on your operating system to run programs. Running programs with the current version of Java will be useful for efficiency. Programs can work in both windows, linux and other operating systems installed java. There is a need for a file that the program can take as an argument for its operation. For the programs to work properly, the format and content of the file must be appropriate for the program. You must enter the path

of the file when running the program. The program was written using jdk-11.0.2 in IntelliJ IDEA. Part 1 requires 6.52 kb memory to keep source files. Part 1 's project size is 1.56 mb with javadoc. Part 2 requires 16.7 kb memory to keep source files. Part 2 's project size is 1.63 mb with javadoc.

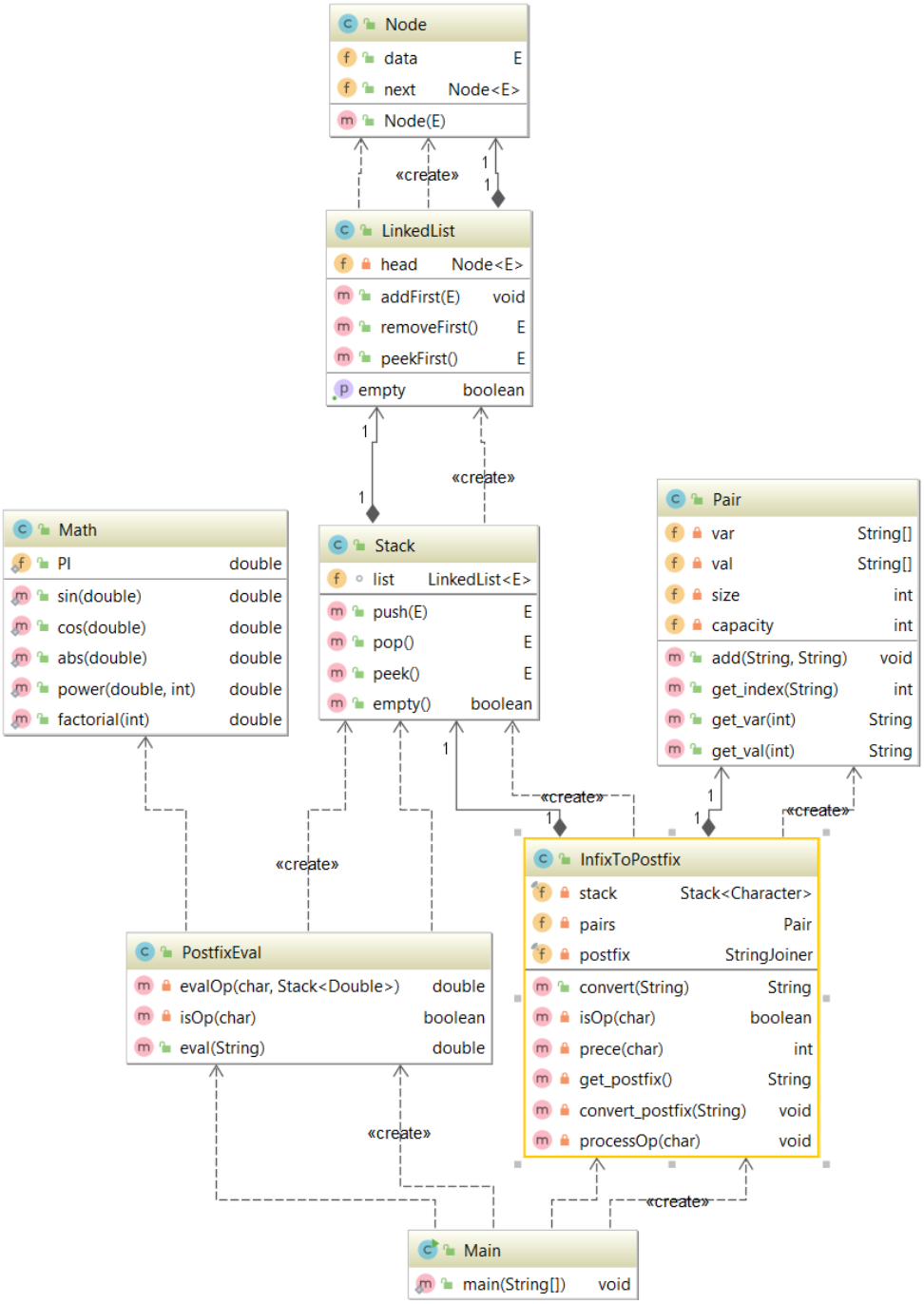
2 METHOD

2.1 Class Diagrams

Part 1:



Part 2:



2.2 Use Case Diagrams

Not required.

2.3 Problem Solution Approach

Part1: First I need to read the elements of the matrix from the file in order to perform the necessary operations. I've read the file line by line to find out how many rows are there. Then I learned with the lenght method how many elements in a row. After all of these operations, I was able to create a 2-dimensional array because I knew the number of rows and columns. I've created a class called Coordinate to keep both the coordinates and the value of each element. I created a 2-dimensional Coordinate array. Then, starting from the beginning of the file, I passed the elements of the matrix to this array. After that, I switched to the algorithm section. I checked if each element of Array is 1. If the element is 1, then I set the value to 0, and I added the value of the neighbors whose value is 1 to the stack data structure. I repeated the same operation until there were no neighbors with a value of 1. Then I increased the counter 1. I repeated the same process until the last element of the array. I used the stack as a data structure because I have to provide the last in first out logic. I have to go back to the first location in order not to lose the paths and not to repeat them. So in short, I used a stack. I used the linked list structure, which provides ease of processing to implement Stack. I wrote the linked list structure myself. Addfirst, removefirst, peek, empty methods have O (1) time complexity. Because they only process on the first element. The algorithm of the problem has Oc(n) time complexity. Since the algorithm will run through each element, it must remain in the loop until (row x column). Since it will control the neighbors of the elements with value 1, and because they will repeat the same process in their neighbors, they have Oc (n) time complexity. The best case is that all elements are 0. So, time complexity is O (n). The worst case is that there are many paths. So, time complexity is approximately Oc(n).

Part 2: First I need to read the expressions from the file in order to perform the necessary operations. I put the statements I read from the file into a string. Then I converted the expressions from infix to postfix. Then I examined each character. Character can be number operator or letter. I had to determine what the character was and do the necessary procedures. If the character is a number, I add the part to the next space, so even if the number is double, it will be added to the string array correctly. If the character is the operator, I have called the process operator method and the process is done according to the precedence of the operator. But here is a special case '-' after the character if there is a number, and this is a negative number to control it according to the transaction. In other cases, the character or string is added directly to the string array, but in methods like cos, sin, abs, the inner part of the parentheses are first coverted and then added to the string array. Then, sin, cos, abs will add in string array. The same process is done until the last character. After converting to Postfix, evaluation process starts. It takes an element from the string array sequentially. If the first element of the string is a number, it converts it into a double number and assigns it to a stack where operands are held. If the first element of the string is a operator, first check whether it represents the negative number and the necessary action is taken. In other cases, Method pop 2 numbers from the stack in which operands are kept, process them with the operator and assigns the result to the operand stack. If the string is cos, sin, abs, it pop 1 number from the operand stack and calls the required method and adds the result to the operand stack. Finally returns the calculated value. I used the stack as a data structure because I have to provide the last in first out logic. I have to pop number sequentially. So in short, I used a stack. I used the linked list structure, which provides ease of processing to implement Stack. I wrote the linked list structure myself. Addfirst, removefirst, peek, empty methods have O (1) time complexity. Because they only process on the first element. The methods used in the conversion and calculation part have Oc (n) time complexity. All other methods have O (1) time complexity. Because they don't contain loop.

3 RESULT

3.1 Test Cases

Part 1: When I was testing this part, I first used all states of a four-by-four matrix.

Like That:

```
1001    0000    1111    1101    1101    1111
0000    0000    1111    0000    0000    0000
0000    0000    1111    1101    1111    1111
1001    0000    1111    1101    1111    1111


1001    1000    0000    1111    1001    1101
0110    0100    0100    1110    1010    0010
0110    0010    0000    1111    1101    1101
1001    0001    0000    0111    0101    1101
```

After the results came out correctly, I started to try random matrices.

Like That:

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

The correct results in the matrices I tried were enough to make the program valid.

Part 2: When I was testing this part, I first used the tests given to us.

Like That:

```
w = 5
x = 6

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

OR

y = 3
z = 16

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

After the results came out correctly, I started to try hard infix expressions.

Like That:

```
x = 5
y = 12
z = 9
t = 3

( cos( z + t ) + t ) * ( 4 * sin( x + ( x * y ) ) ) + ( abs( x * y ) + 3.2 ) + x - y * -2
```

The correct postfix expression and evalution results in infix expression I tried were enough to make the program valid.

3.2 Running Results

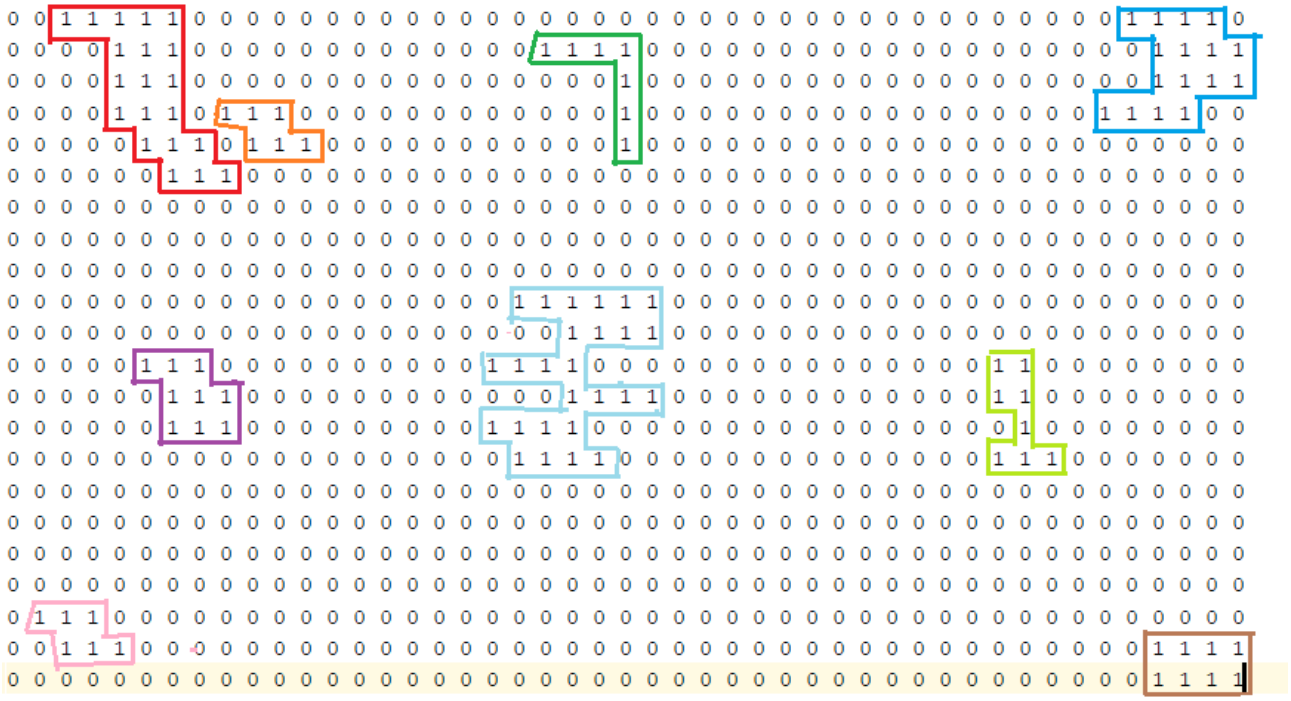
Part 1:

Matrix:

```
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:

```
White components: 9
```



Part 2:

Test 1:

```
Entered Infix Expression:
w = 5
x = 6

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

Converted Postfix Expression:
5 4 + 6 cos 77.9 - *

Evaluated Postfix Expression: -692,149303
```

Test 2:

```
Entered Infix Expression:
y = 3
z = 16

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

Converted Postfix Expression:
3 3 16 * sin + 16 -10.3 abs * +

Evaluated Postfix Expression: 168,543144
```