Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 3 REPORT

GÖKÇE NUR ERER
171044079

Course Assistant: Özgü Göksü

1 INTRODUCTION

1.1 Problem Definition

For part 1:

In part 1 the objective is to find the number white components in a matrix given with a text file. White components are sets of white matrix locations which between any two of them, there is a path of white matrix locations. If a white matrix coordinate has up, down, left or right neighbors which are white too, they are in the same component. Also, if a component has no white neighbors it is still counted as a component itself.

For part 2:

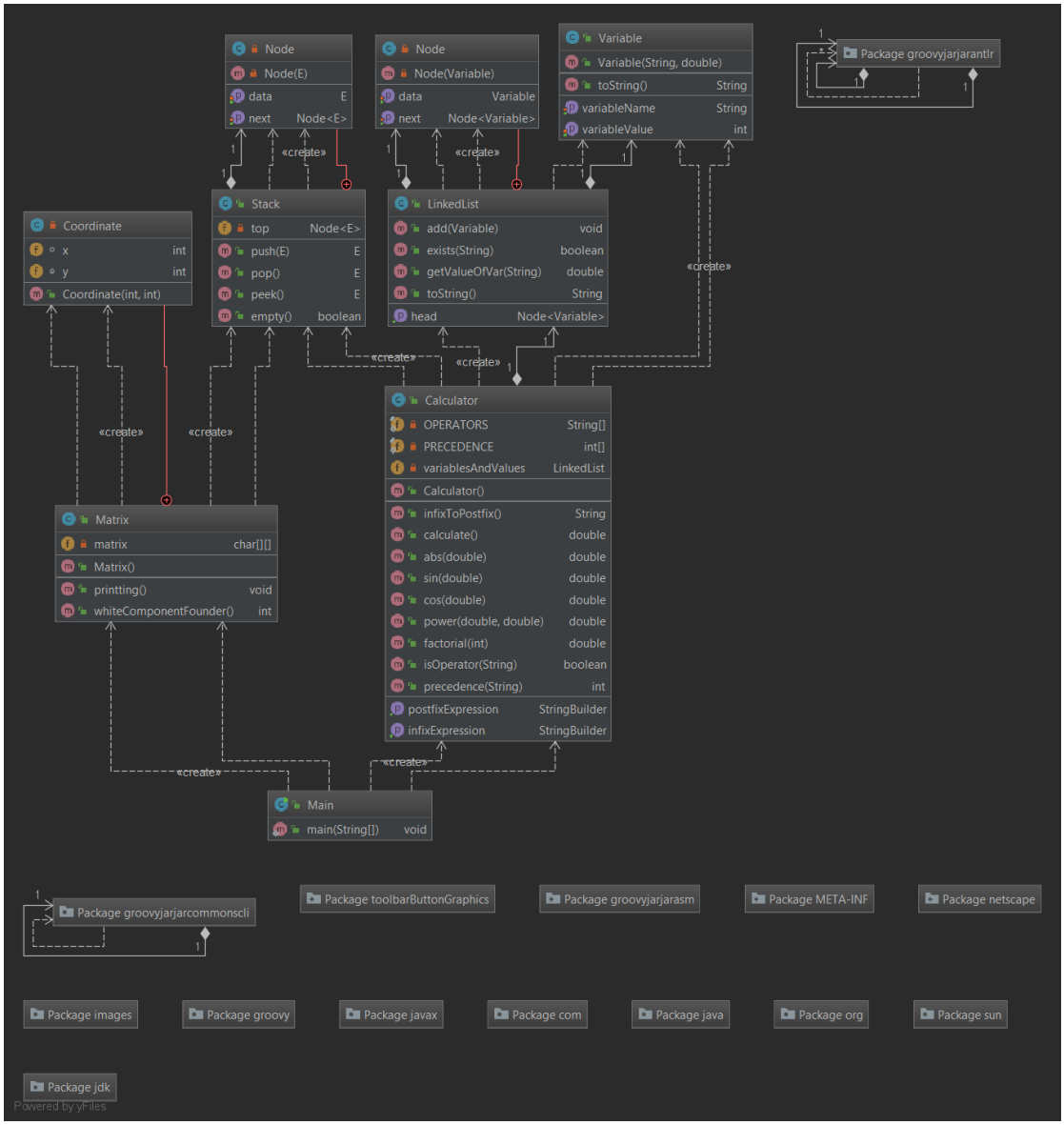
In part 2 the objective is to evaluate an infix expression. To do such, first it is required to convert the infix expression to a postfix expression and then evaluate that with a stack-based algorithm.

1.2 System Requirements

This project is capable of running on an average computer which has Java components like (JVM, SDK, JDK...) and has an IntelliJ Idea IDE since it is and IntelliJ project.

2 METHOD

2.1 Class Diagrams



2.2 Use Case Diagrams

-

2.3 Problem Solution Approach

For both parts stack implementation was required. So, a generic class Stack is implemented for use.

Stack class has a inner class called Node to keep the elements in. A stack should have push, pop, peek and empty functions originally, so they are implemented as well.
public E Push (E entry): pushes the given entry to the top of the stack, returns the pushed element.
public E pop () : pops the element at the top of the stack and returns it.
public E peek (): gets the element at the top of the stack but does not remove it.
public boolean empty () : checks if the stack is empty or not.

For part 1:

For this problem it is required to represent the matrix which was given with a file. Matrix class is exactly the representation of that. Matrix class reads the file line by line and initializes everything in a two-dimensional character array.
Also to get certain coordinates in a given matrix the coordinates has to be represented too. Coordinate inner class in matrix accomplishes this requirement. Coordinate class gets an x and y coordinate and keeps them inside for further access.
The method which solves the problem is called whiteComponentCounter(). It starts traversing the matrix one by one and whenever it sees a white matrix location it starts to check if it has a neighbor which is also a white matrix location. This function uses a stack to keep the white matrix locations' coordinates to keep track of the traversing way of the white component. Until there are no white matrix locations left to go it keeps on checking the neighbors of each component over and over. When there is no other way to go coordinates are popped from the stack one by one and
stack becomes empty. And whenever a stack becomes empty that means the traversal for that component ended so the total component count is increased. Also to mark the components letters such as A,B,C is used. To accomplish the character changing, first initialized a character with A then incremented it whenever total component count increases the character gets increased too.

Also printing method called printting() is implemented to check where is marked in the matrix for which component easily.

For part 2:
For this problem it is required to represent a calculator to do the calculation of the infix expression given in the file. Calculator class is the representation of that necessity.

Calculator class's constructor first reads the file and gets the necessary values to necessary places. It initializes infix expression's value with the last line of the file and reads the first lines and creates Variable objects to both keep their string value and numeric value. And to keep all these Variable objects a custom implemented LinkedList class is used. Their implementations will be provided later.
Calculator class has multiple methods to evaluate the infix expression which are:

public String infixToPostfix() : this function converts a infix expression to a postfix expression by using a stack. Stack is used for the operators and parenthesis mainly. If the read element in the infix expression is a operator, first it compares the precedence of the top of the stack and the element that will be added to the stack, if the element's precedence is bigger it gets pushed to the stack, else if the top of the stack's precedence is bigger than the element's precedence all the elements of the stack that have a bigger precedency than the element are popped from the stack, written to the postfix expression and then the element pushed. If the read element in the infix expression is a ")" , since the "(" are also pushed to the stack, all the operators that are in the stack until a "(" are popped and written to the postfix expression, lastly the "(" is popped too. If none of these conditions apply if the element is a "(" it is pushed to the stack, and if its a number or a variable it is added to the postfix expression only. After all these, all the remaining operators are popped from the stack, and get appended to the postfix expression.

public double calculate (): This function calculates a postfix notation using a stack. Everything which is not a operator is pushed to the stack, if an operator comes, if it is a "+" , "-" , "/" , "*", two operands are popped from the stack, if the operation is "sin", "cos", "abs", one operand is popped from the stack, calculation is done and then the calculation gets pushed to the stack again. Finally, the left element in the stack is popped and returned.

public double abs (double a): calculates the absolute value of a given double.

public double sin (double x): calculates the sine value of a given degree x with the Taylor expansion of sine.

public double cos (double x): calculates the cosine value of a given degree x with the Taylor expansion of sine.

public double power (double x , double y) : calculates the value of the first argument raised to the power of the second argument.

public double factorial (int x) : calculates the factorial of the given argument.

public boolean isOperator (String s) : checks if the given string is an operator depending on the final static String array OPERATORS defined in the Calculator class which has { " + " , " - ", " * " ," / "," sin "," cos "," abs "} in it.

public int precedence(String s) : returns the precedence of the given operator depending on the final static int array PRECEDENCE defined in the Calculator class which has { 1, 1, 2, 2, 3, 3, 3} which is parallel to the OPERATORS array.

Above, it was mentioned that a LinkedList class was implemented to keep the Variable objects. The Linked List class has these methods:

public void add (Variable data) : adds the given argument to the end of the list, if head is null it adds it as the head, else it finds the last element in the list and adds the argument after it.

public boolean exists (String x) : checks if a given string related variable exists in the list.

public double getValueOfVar (String x) : gets the value of the variable in the list , related to the given string.

Time Complexity

PART 1 FUNCTIONS		
Class	Function	Time Complexity
Matrix	Printting()	O(n) because overall element count is n and both loops traverse less than that.
Matrix	Matrix constructor: Matrix()	O(n): being the file line count
Matrix	whiteComponentFounder	O(n) because 2 inner for loops loop n times (n is the total array size) and at the worst case which is all elements being 1 1 1 ... while loop

		will loop n times for only 1 element since all the visited 1s are marked with a letter, they will not be visited later within the for loops, which will result in O(n+n) = O(2n) = O(n) In another case which the elements are 1 0 1 0 (no single 1 is matching with another one) for n/2 element the while loop will loop only 1 time so it will be, overall n/2 times so O(n/2+n) = O(3n/2) = O(n)
PART 2 FUNCTIONS		
Calculator	Calculator constructor: Calculator()	O(n): being the file line count
Calculator	infixToPostfix	O(n): n being the infix expression length
Calculator	calculate	O(n): n being the postfix expression length
Calculator	abs	O(1)
Calculator	sin	O(1)
Calculator	cos	O(1)
Calculator	power	O(n): n being the power
Calculator	factorial	O(n): n being the number
Calculator	isOperator	O(1): since length of the array OPERATORS are fixed
Calculator	precedence	O(1): since length of the array OPERATORS are fixed
LinkedList	add	O(n): n being the size of the list
LinkedList	exists	O(n): n being the size of the list
LinkedList	getValueOfVar	O(n): n being the size of the list

FEW EXTRA NOTES

-For part 1, the whole program works in a way that at each line of the file there is no extra space! If there is a space program crash.

-For a more flexible use, the file names are taken from the user for both parts.

3 RESULT

3.1 Test Cases

Tested the program with the input files given by the instructors, and with few test files I have written. The running results I will provide will only have the instructor provided files' outputs.

3.2 Running Results

Outputs of the provided files are below;

```
PART 1 - COUNTING COMPONENTS
Enter the file name:
test_file_part_1.txt
Component Map:
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 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 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
The number of the white components are:9
PART 2 - A CALCULATOR
Enter the file name:
test_file_part_2-1.txt

w=5.0
x=6.0

The Infix Notation: ( w + 4 ) * ( cos( x ) - 77.9 )
The Postfix Notation is: w 4 + x cos 77.9 - *
Evaluated value is : -692.1492530110093

Process finished with exit code 0
```

```
PART 2 - A CALCULATOR
Enter the file name:
test_file_part_2-2.txt

y=3.0
z=16.0

The Infix Notation: ( y + sin( y * z ) ) + ( z * abs( -10.3 ) )
The Postfix Notation is: y y z * sin + z -10.3 abs * +
Evaluated value is : 168.54286057359306

Process finished with exit code 0
```

- Main titles -> 16pt , 2 line break
- Subtitles -> 14pt, 1.5 line break
- Paragraph -> 12pt, 1.5 line break