Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 03 REPORT

AHMED SEMİH ÖZMEKİK
171044039

Course Assistant:

1 INTRODUCTION

1.1 Problem Definition

1.1.1 Problem 1 (Counting Components)

We have a bitmap which happens to be the representation of some kind of image. Of course, we are not dealing with whole and concrete image, but the idea is similar. And we are expected to find: How many “neighbourhood“ are there, in these bitmap? Or maybe we can think of this problem as to find the island in the map of lands. Of course, in this manner, if we take two piece of lands and see that they are adjacent with each other, then it means they exist in the same island. As we can see, we can simulate this problem in too many ways. Returning to the real case, in our bitmap we have either “1” which represents the “white” or “0” which represents the “black”. And our problem is to find the “white” component in this map.

1.1.2 Problem 2 (A Calculator)

We are given an infix expression and we are expected to find the result of this expressions. Of course, as humans we are convenient with the infix expression, since because we are used to it. But for the computers, the expression itself is nothing but a useless byte sequence. But if we go one level higher, maybe be can say the expression is character sequence, yet again, it is useless. To be able to calculate the expression, we need to come up with a solid algorithm for a strict syntax. For this problem, we need to take the problem of precedence of operations and the existence of the parenthesis into account. Of course brackets mean a lot in terms of a delimeter in our expression sequence and we must think them as a weapon. Plus, there are some functions in our expression such as “sin()” or “cos()” and it consists variables belong to user. In essence, we have an mathematical expression which happens to have variables, functions, brackets; armed with a strict syntax which we can and will be make use of.

1.2 System Requirements

1.2.1 Requirements for Part I (Counting Components)

For this part, before the requirement of a solid algorithm, I need a data structure to store the bitmap properly so that I can work with the bits in a good way. Since the bitmap is a two dimensional map, it could be a good idea to pick the “array” structure for this requirement. Of course, picking a data structure for the bit sequence was belong to part which is the algorithm requirement. Because, I should solve the problem with iterating the bit sequence only once. To achieve this and with taking advantage of simplicity of the map, I decided to go on with an “array” structure. I need a O(n) complexity for the algorithm, so I can satisfy this requirement within this circumstances. For the algorithm part, we are suggested to use stack structure. If we think about the algorithm, we can realize that we will check the neighbours of one bit, then say we find a “white” again, hence we will go on with that one and so on. The flow of this fragmentary algorithm seems like it needs a recursion for to solve. Yet, using recursion was forbidden and we know that stack solutions are very similar to recursion solutions. So, for this requirement I picked the stack solution. Once I accomplish the algorithm, only remains the concrete command-line program to serve this algorithm which is our last requirement for this problem. User will/must give the “.txt” file as input and he or she will get the “white component” number for that map.

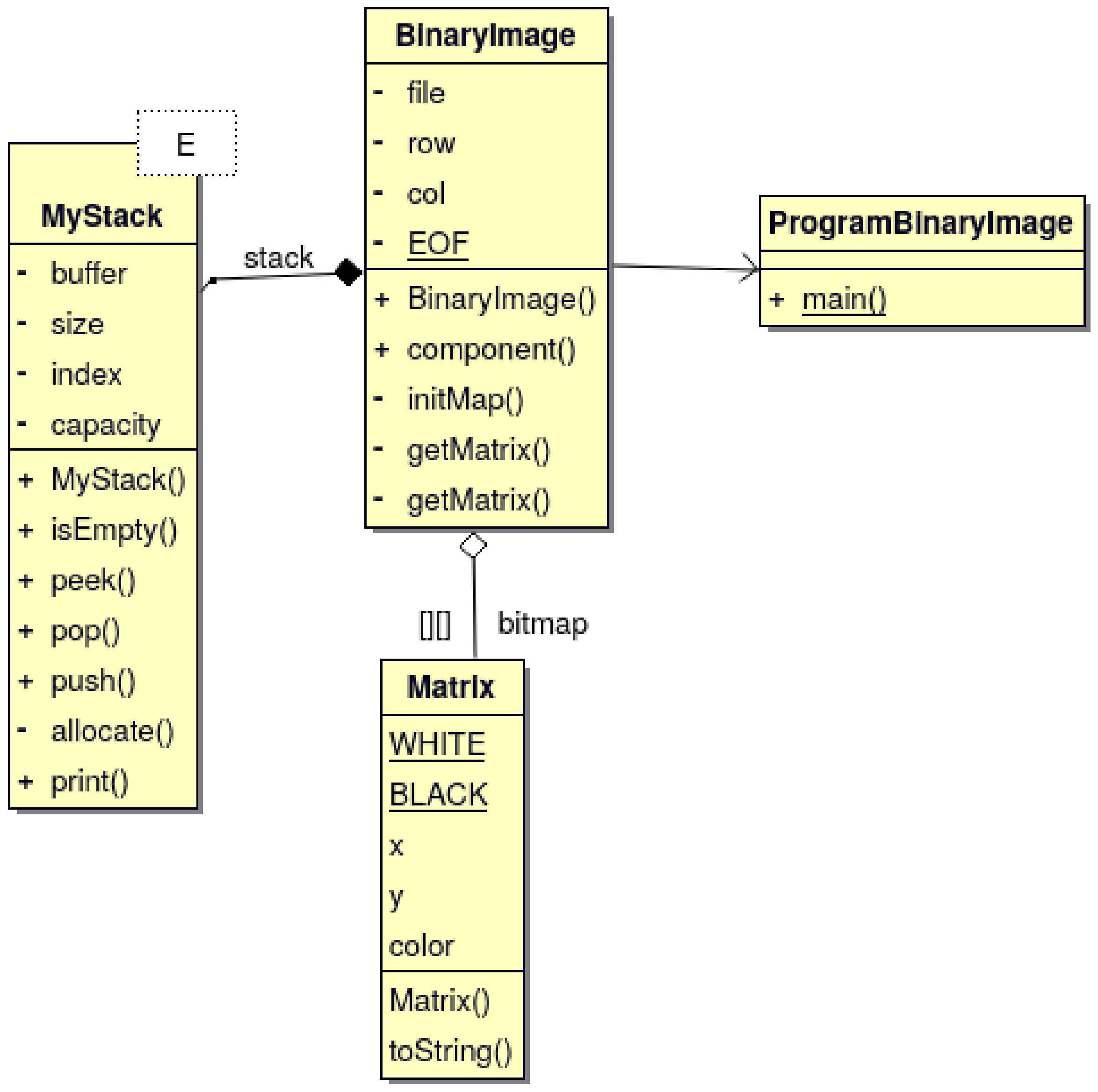
1.2.2 Requirements for Part II (A Calculator)

We can have variables, functions, and brackets as I mentioned. For the variables, I need a data structure to store them. Of course, I must know in time, what value I should use when I encounter a variable used in the expression. Meaning that, I should be able to get a value from the structure when I give the proper key for it. Of course, usage of “HashMap” structure could have been useful but I am not supposed to use that one, rather instead I require a “Variable” class such that having one key which happens to be the letter for that certain variable and one value for that variable. And then, I need a data structure to store these variables. The other serious requirement of this problem was a solid algorithm which uses a stack solution. But for the necessity I required two stack for this problem which I will be mentioning the “problem solution approach” section. The last requirement for this problem is to have a concrete program which uses all this functionality and serves to user. User will/must give the “.txt” which has the infix expression with proper syntax and afterwards he/she will get the result of that expression.

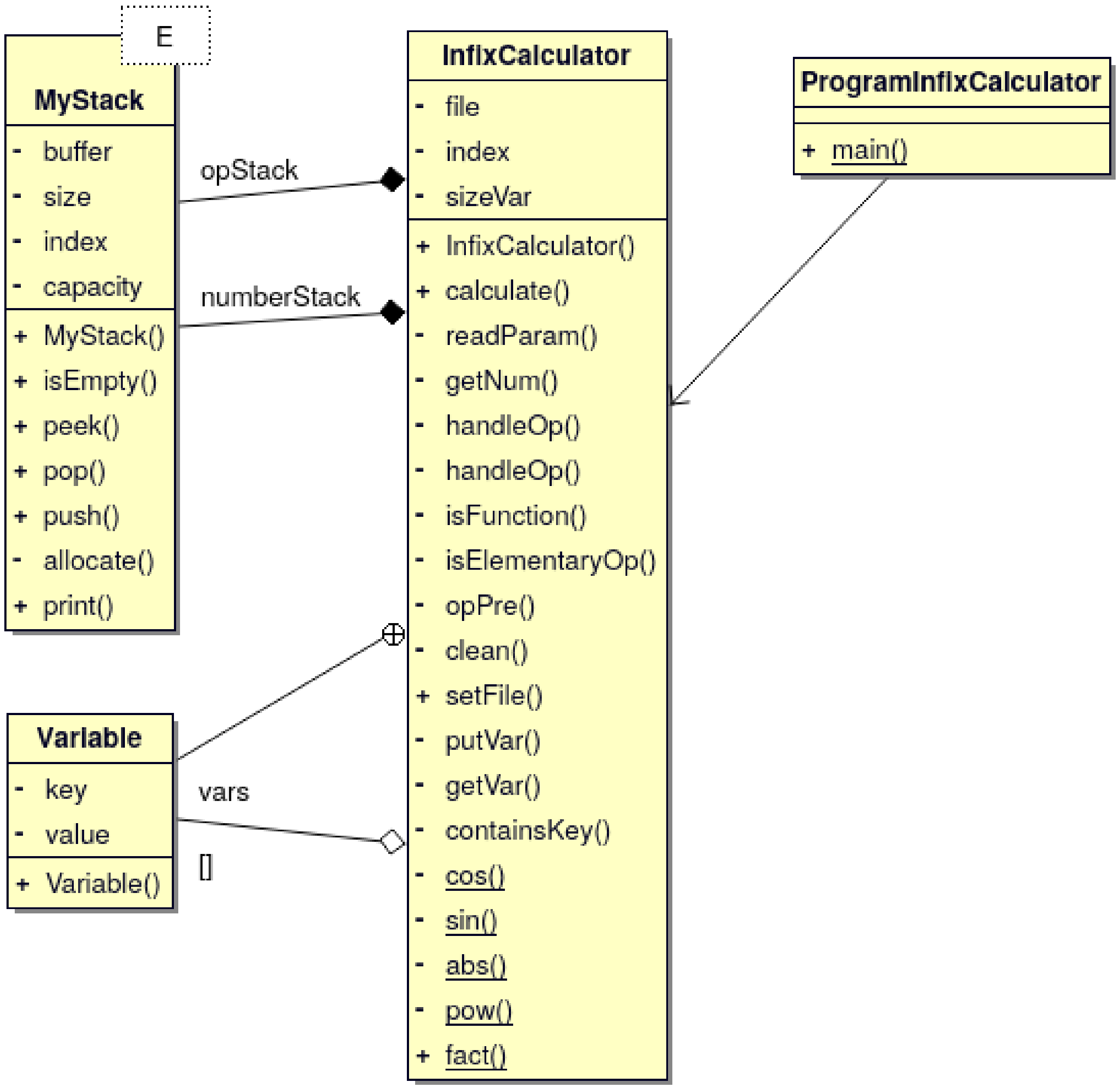
2 METHOD

2.1 Class Diagrams

2.1.1 Class Diagram for Part I (Counting Components)

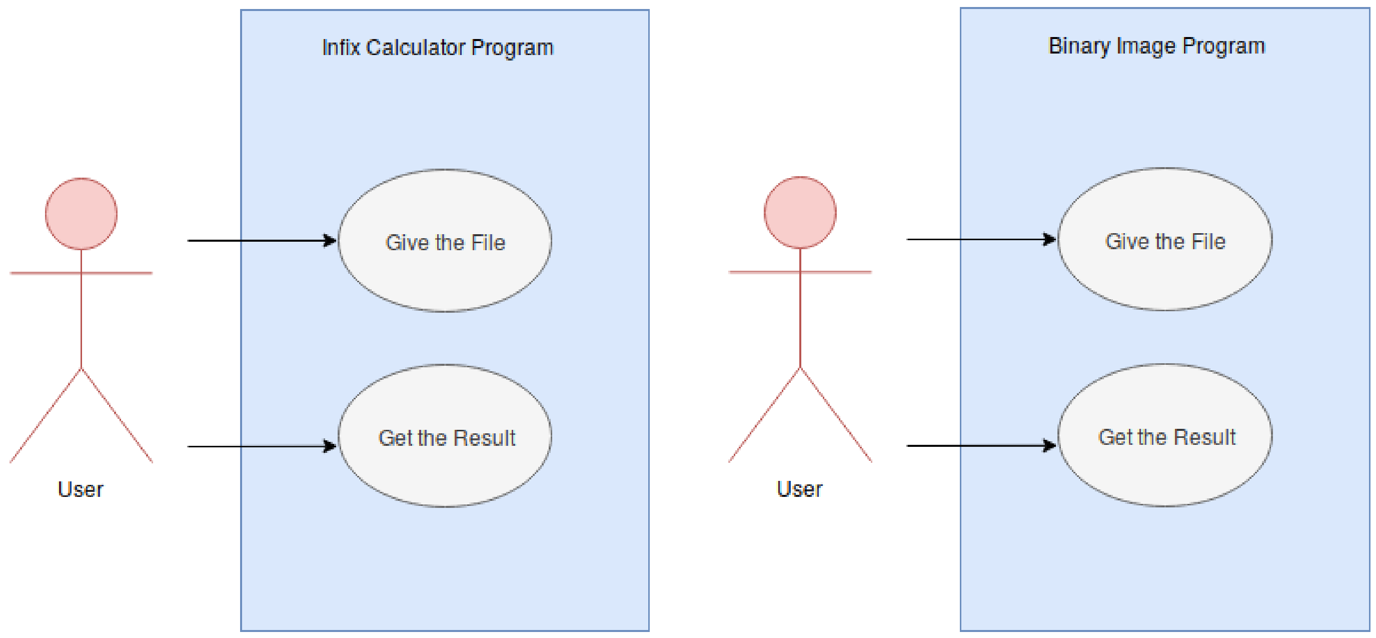


2.1.1 Class Diagram for Part II (A Calculator)



2.2 Use Case Diagrams

Use case diagrams are pretty much the same, since the only interaction between user and the program is giving input as file and taking the ouput as result:



2.3 Problem Solution Approach

2.3.1 Problem Solution Approach for Part I (Counting Components)

As I was mentioned during the “system requirement” section, I decided to go on with the “array” structure for to store the bits. But, I did not only store them as bits, instead I decided to write a concrete “Matrix” class for the represent the bits regarding to their colors and positions. Because, after pushing a “white” bit to stack, I needed to know it’s position once I popped it, I can make use of it’s position in searching the neighbourhood. And I needed the “color” field for to change it later on to “black” bit. Because, I needed to make sure that once I removed I won’t be checking that one again. Stack was a brilliant idea for this problem. Because, the stack solution was simulating the recursion solution while searching the neighbourhood of bits. Using of stack made the implementation very clean and solid. Time complexity of the algorithm is O(n). In the avarage senario, I check some of the bits twice, but this is only making the “T(n)” is “4n” which is very acceptable since it is still O(n). For the worst case senario, “T(n)” is not greater than “5n” which again is sane.

In space wise, I am only using stack(n at worst case) and array(n) which was not a big deal.

2.3.2 Problem Solution Approach for Part II (A Calculator)

I have designed the “Variable” class for to deal with the variables better. As soon as I get them from file, I put them into my “Variable Array”. In some way, I was actually simulating the “HashMap” structure’s behavior which has been really helpful. I defined two different stack, one stack for the operations and one stack for the numbers. Whenever I saw a closing brackets, until I saw a opening brackets, I have popped the numbers and the operations from both stack and process the operation due to operands. I was iterating the list only once and pushing/popping the characters only once, which makes the algorithm “T(n)=n”. Stack idea was very useful, in terms of dealing with the precedence of the operations.

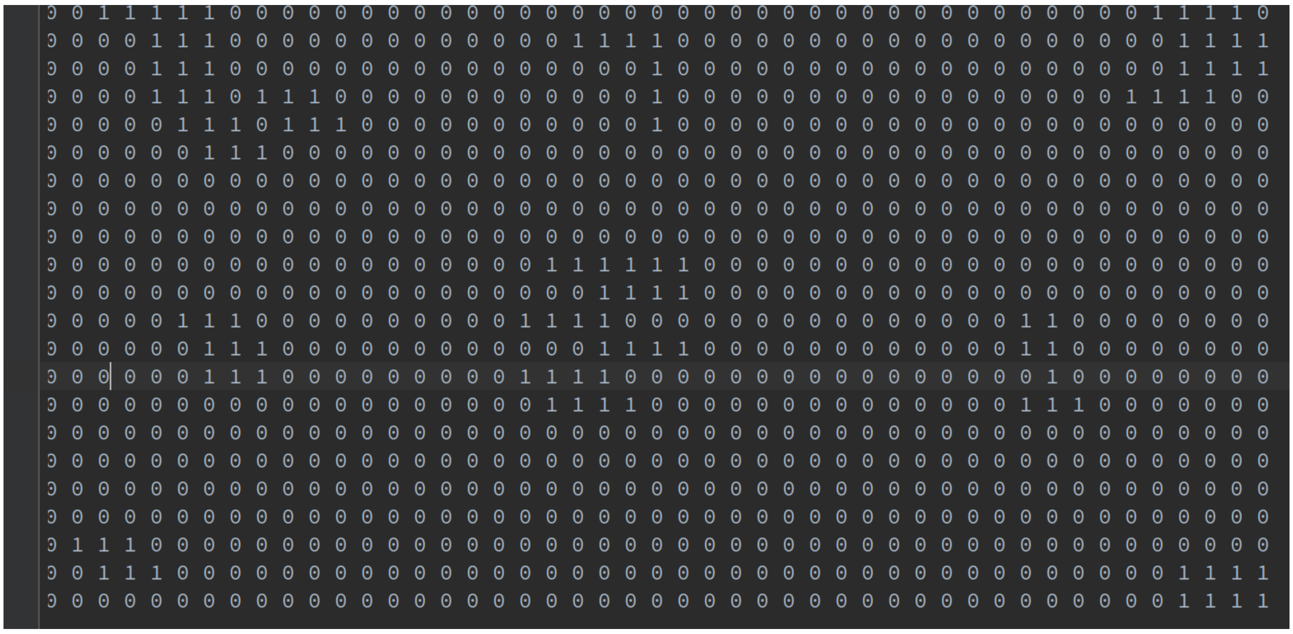
3 RESULT

3.1 Test Cases

I have wrote 4 test cases for each problem which was rough cases.

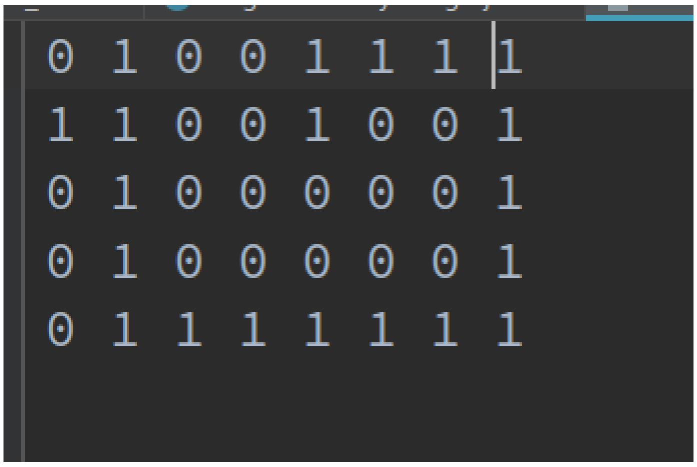
3.1.1 Test Cases for Part I (Counting Components)

Case 1: (White Component = 9):



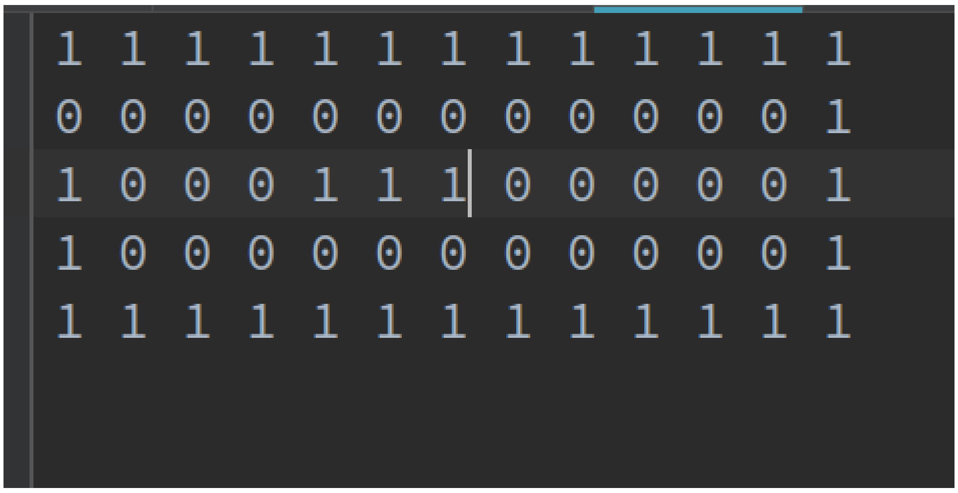
```
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 0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

Case 2: (White Component = 1):



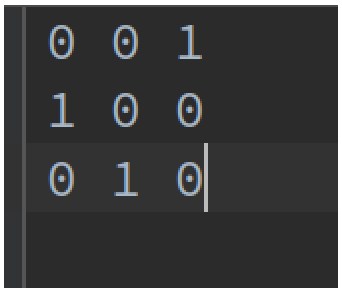
```
0 1 0 0 1 1 1 1
1 1 0 0 1 0 0 1
0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1
0 1 1 1 1 1 1 1
```

Case 3: (White Component = 2):



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

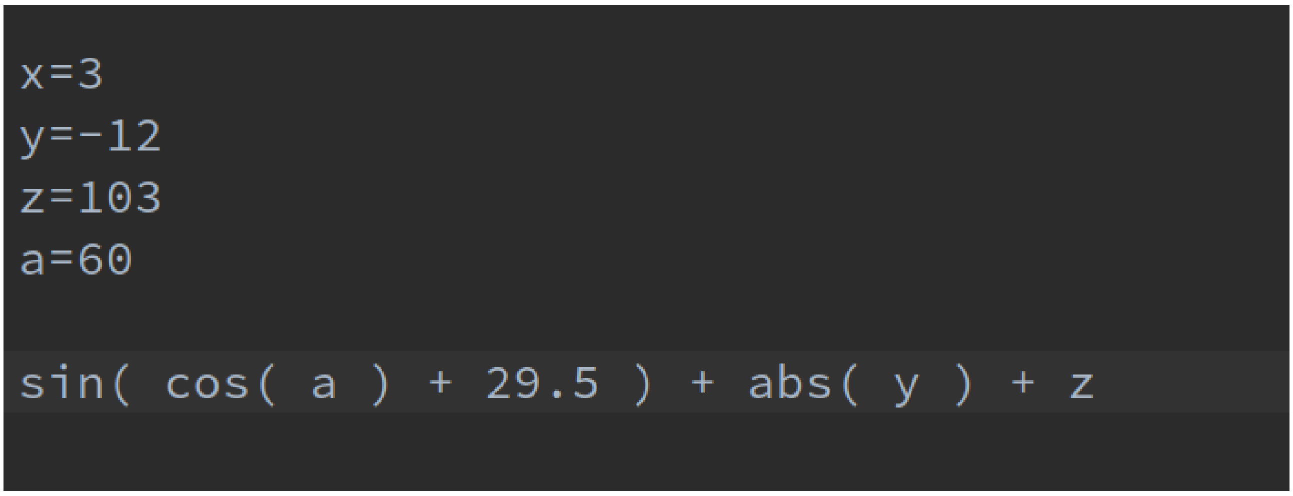
Case 4: (White Component = 3):



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

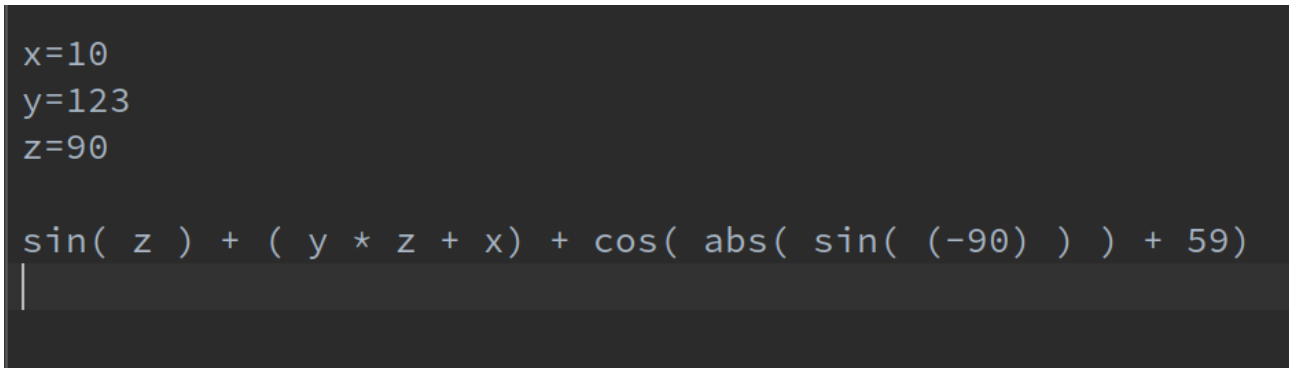
3.1.1 Test Cases for Part II (A Calculator)

Case 1: (Result = 115.499)



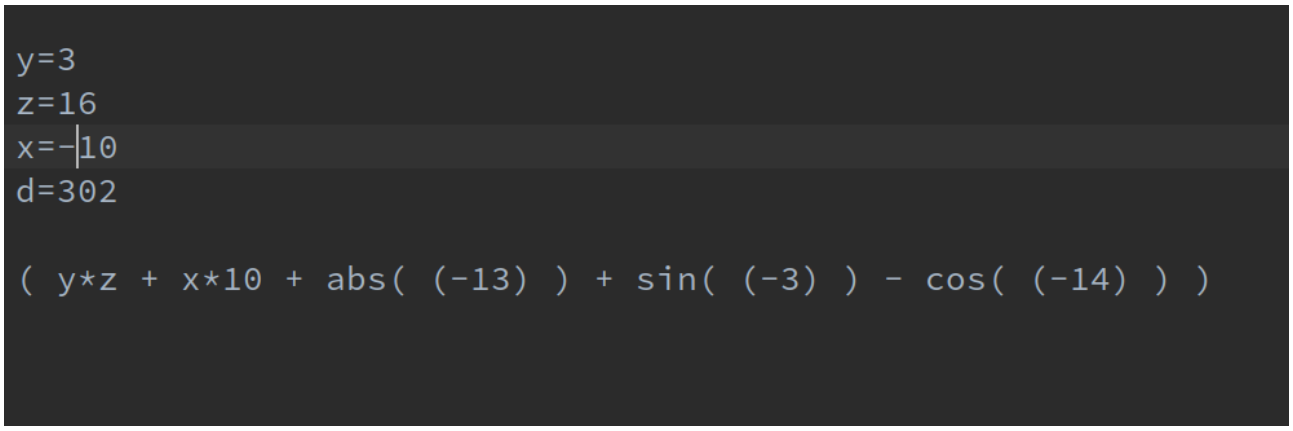
```
x=3
y=-12
z=103
a=60

sin( cos( a ) + 29.5 ) + abs( y ) + z
```

Case 2: (Result = 11081.5)



```
x=10
y=123
z=90

sin( z ) + ( y * z + x) + cos( abs( sin( (-90) ) ) + 59)
```

Case 3: (Result = -40.02)



```
y=3
z=16
x=-10
d=302

( y*z + x*10 + abs( (-13) ) + sin( (-3) ) - cos( (-14) ) )
```

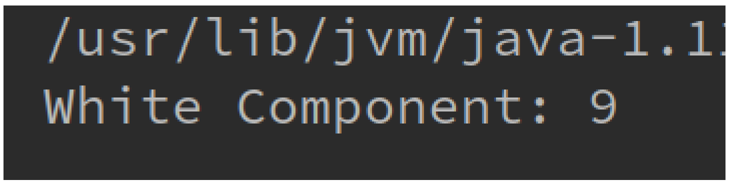
Case 4: (Result = 0)

```
()
```

3.2 Running Results

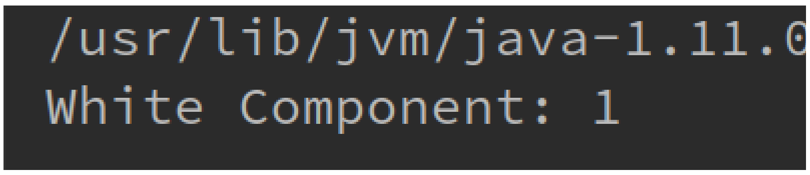
3.2.1 Test Cases for Part I (Counting Components)

Case 1:



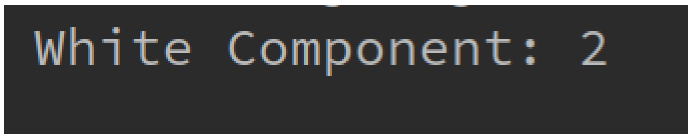
```
/usr/lib/jvm/java-1.1
White Component: 9
```

Case 2:



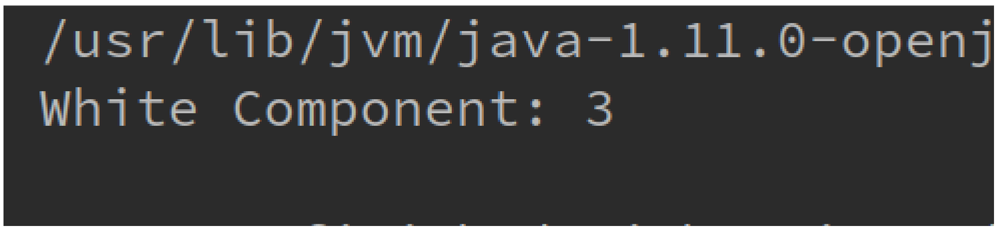
```
/usr/lib/jvm/java-1.11.0
White Component: 1
```

Case 3:



```
White Component: 2
```

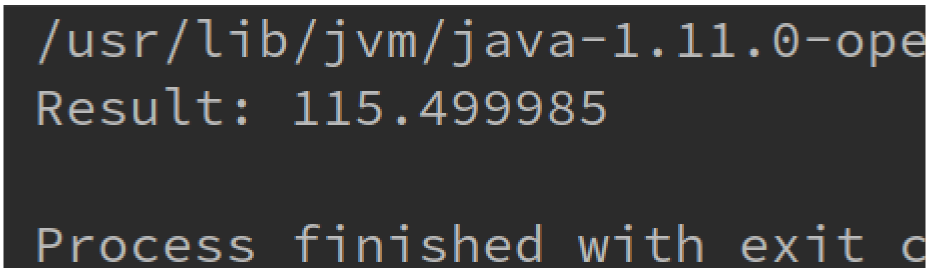
Case 4:



```
/usr/lib/jvm/java-1.11.0-openj
White Component: 3
```

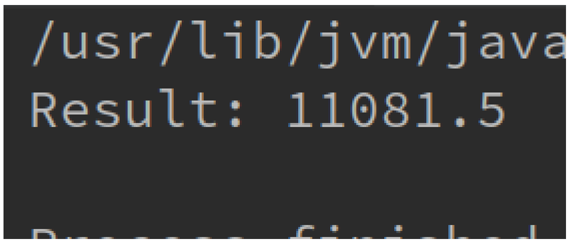
3.2.2 Test Cases for Part II (A Calculator)

Output for Case 1:



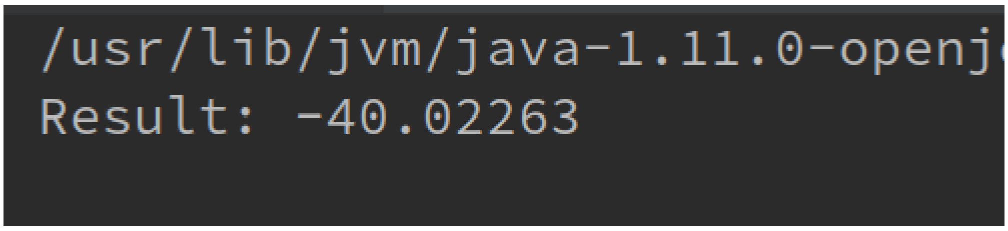
```
/usr/lib/jvm/java-1.11.0-ope
Result: 115.499985

Process finished with exit c
```

Output for Case 2:



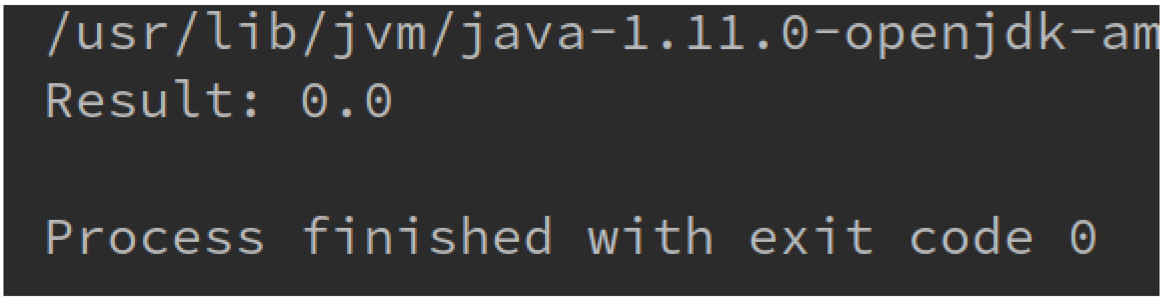
```
/usr/lib/jvm/java
Result: 11081.5

Process finished
```

Output for Case 3:



```
/usr/lib/jvm/java-1.11.0-openj
Result: -40.02263
```

Output for Case 4:



```
/usr/lib/jvm/java-1.11.0-openjdk-am
Result: 0.0

Process finished with exit code 0
```