# Data Structures and Algorithms

# INFO 6205

# Homework 3

# Due: October 3, 2020

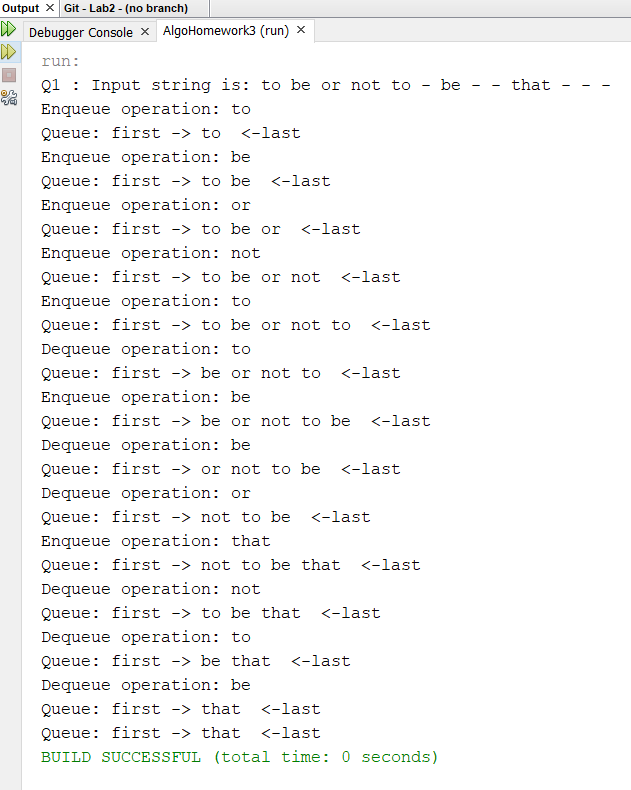
Put all your java, compiled class files and documentation files into a zip file named Homework3.zip and submit it via the drop box on Canvas before the END of due date.

Put your name on all .java files. There will be a short quiz on this homework.

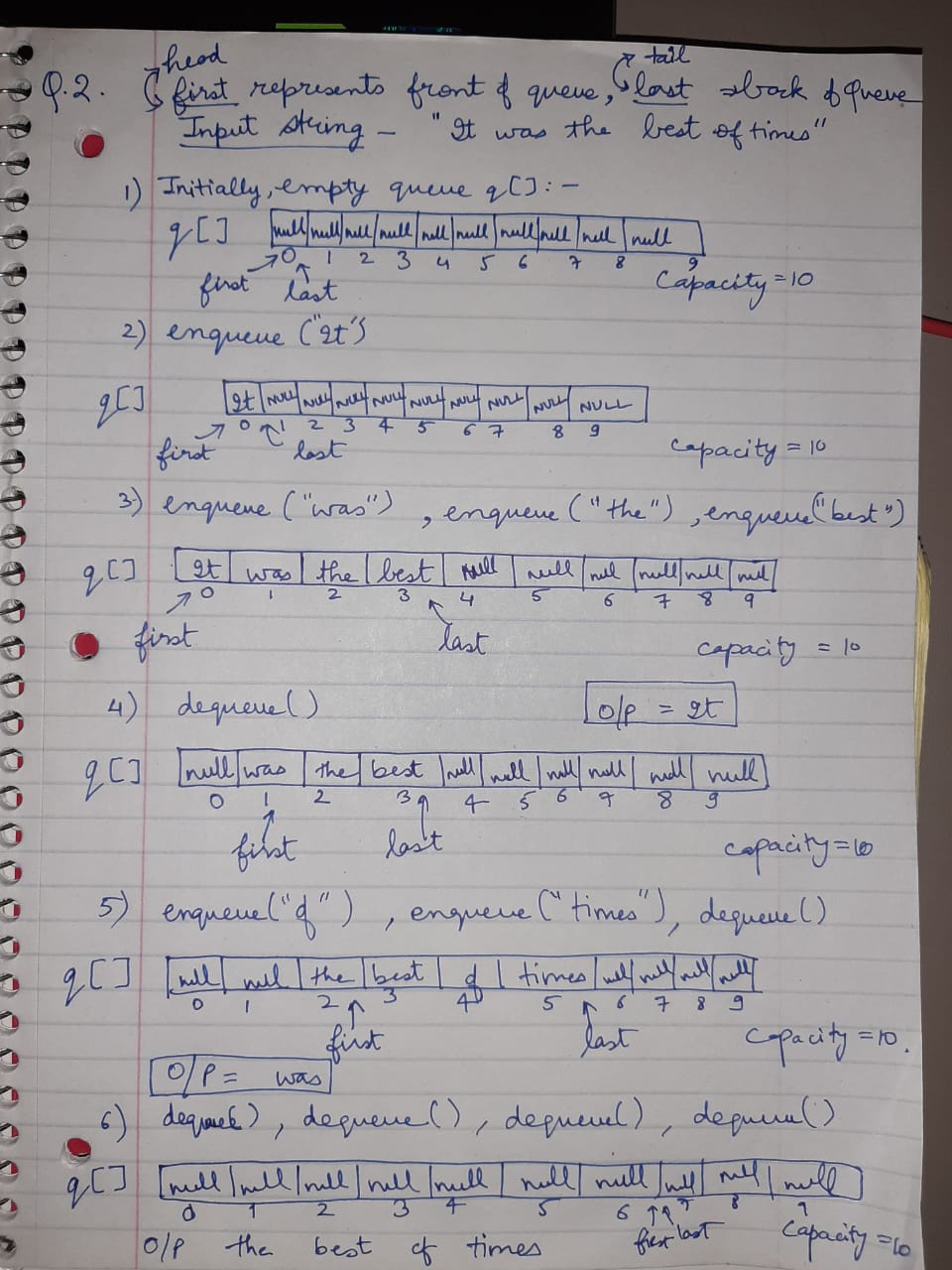
1. For the LinkedList implementation of Queue example “to be or not to be” I discussed in class, write a TestLinkedListQueue class to test enqueue, dequeue,, isEmpty and other operations as needed.

**Answer- Code in QueueLinklist.java and TestLinkedListQueue.java**

Output->



2. Describe the Array Implementation of Queue with “It was the best of times” example. You need to provide a sample data and walk through the enqueue and dequeue, and other operations as necessary and manage the head and tail pointers.



In diagram, first depicts head pointer and last depicts tail pointer.

**3. Consider the following QueueOfStrings code to manage queue. The input to this method is String “The temperature - - degrees today and it - - - tomorrow”.**

A) Show step-by-step of queue execution

B) What is the output

public static void main(String[] args)  
 {  
 QueueOfStrings q = new QueueOfStrings();  
 while (!StdIn.isEmpty())

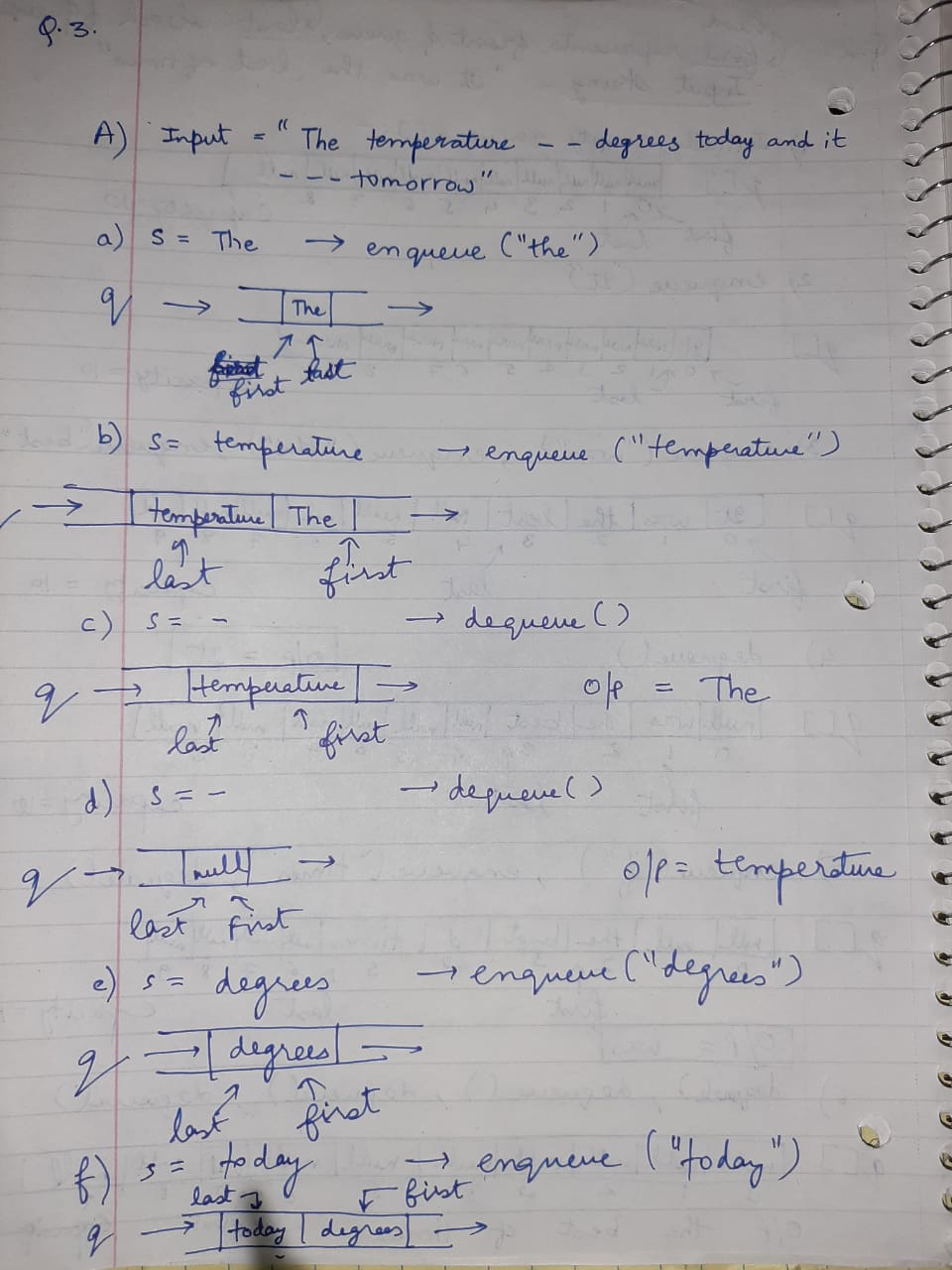
{

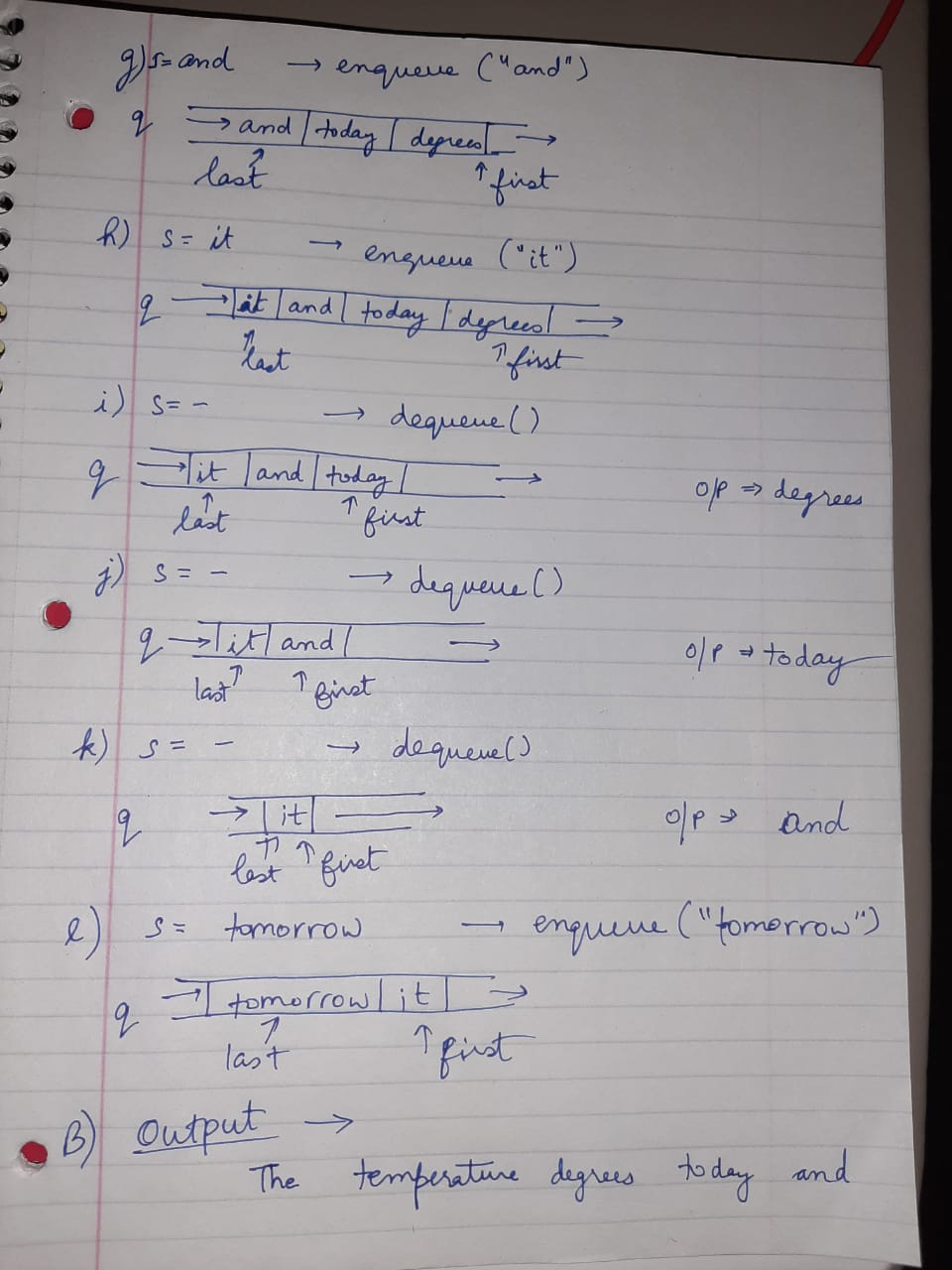
String s = StdIn.readString();

if (s.equals("-")) StdOut.print(q.dequeue());

else q.enqueue(s);

}   
}

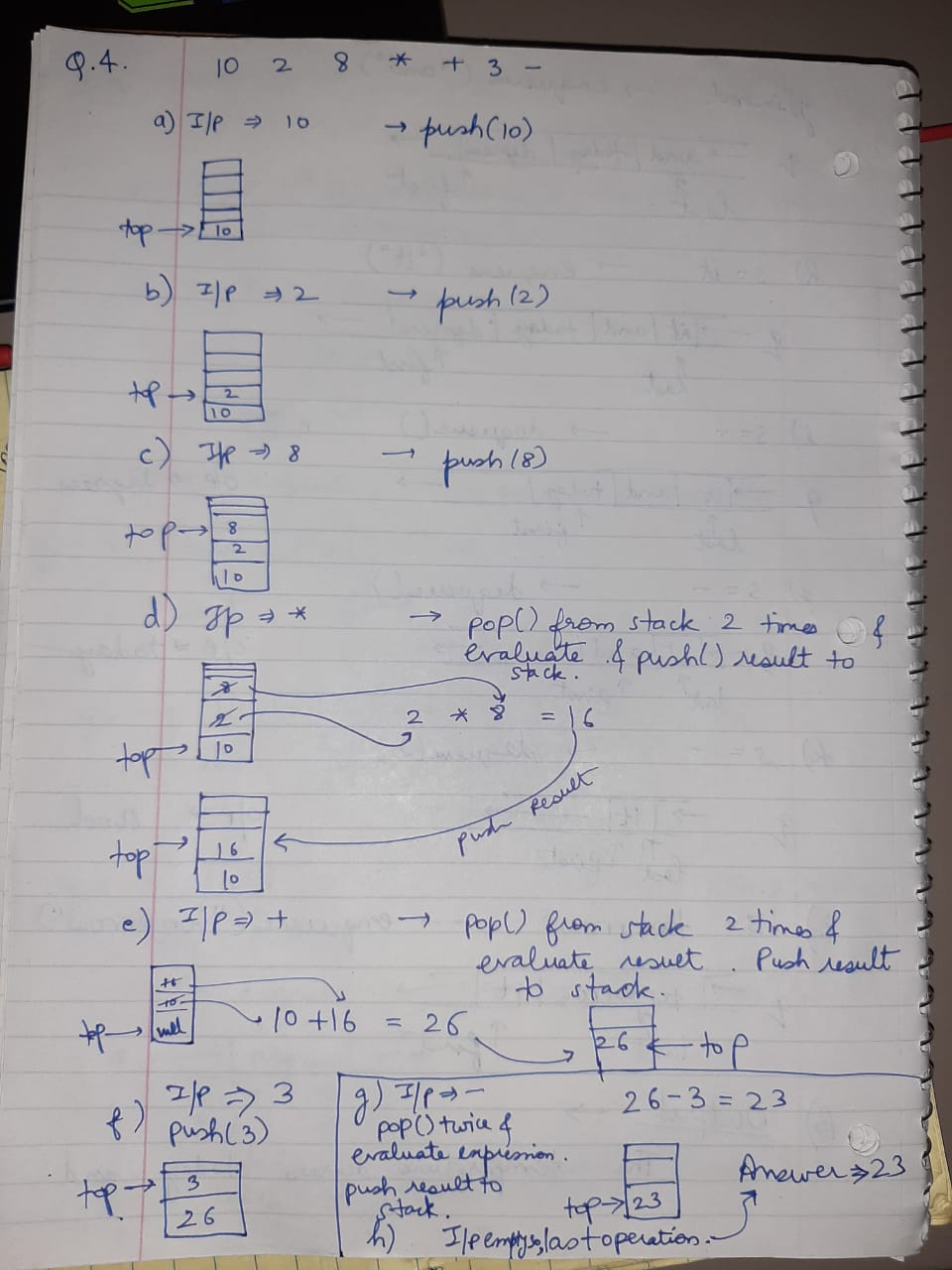




**4. Consider this Algorithm to “Evaluate Postfix Expression**”: **10 2 8 \* + 3 -**

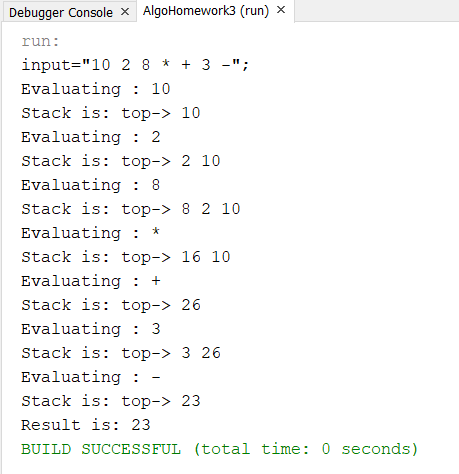
Algorithm: Maintain a stack and scan the postfix expression from left to right – When we get a number, output it – When we get an operator, pop the top element in the stack until there is no operator having higher priority then this operator, and then push (operator) into the stack – When the expression is ended, pop all the operators remain in the stack:

1. Show Stack step-by-step

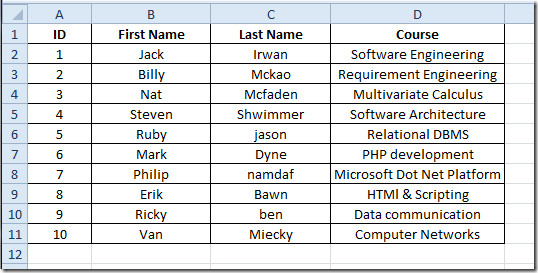


1. Write Java code to compute postfix expression—

**Answer- Code in files- StackGenericLLAditi.java, Q4PostfixExpression.java**

Output- 

5. Consider the following data:



Build **Queue** with LinkedList implementation and Array implementation:

a) Create file “input.txt” with this data

b) Read input.data into an an ArrayList.

c) Create Queue with LinkedList implementation

d) Write Node data structure of your input data

e) Queue must support all operations of queue

f) Write a Test program to test your linked implementation of Queue:

—enqueue all elements into queue

—dequeue 4 elements from queue

—enqueue all elements into queue

—dequeue all elements from queue

—dequeue 1 element

—enqueue all elements into queue

—enqueue this element into the queue:

11 John Johnson Java Programming

—Print queue with the goal:

i) reverse order ii) original order as was first read into array list

g) Compile and Run your program

h) what is Queue LinkedList time-complexity?

i) Repeat (a)—(g) with Queue fixed Array Implementation

j) what is Queue Fixed Array time-complexity?

k) What are the consequences of oversizing or undersizing fixed array size?

**Answers-**

Code files-

Input- input.txt

Queue link list implementation-- QueueGenericLLAditi.java, Q5QueueLLAditi.java,User.java

Queue Fix array implementation—Q5QueueArrayAditi.java, QueueGenericArrayAditi.java,User.java

h) For operations enqueue(), dequeue(), size(), isEmpty(), the time complexity is O(1) for Link list implementation as well as fixed array implementation for all operations. Printing queue takes O(N) time where N is number of elements in queue.

k) Oversizing : Space complexity increases as array is assigned a size greater than required.

Undersizing: Time complexity increases as array size is to be increased frequently.

Output for Queue Linklist-

run:

Enqueue all elements...

Queue: 1 Jack Irwan Software Engineering 2 Billy Mckao Requirement Engineering 3 Nat Mcfaden Multivariate Calculus 4 Steven Shwimmer Software Architecture 5 Ruby jason Relational DBMS 6 Mark Dyne PHP development 7 Philip namdaf Microsoft Dot Net Platform 8 Erik Bawn HTML & Scripting 9 Ricky ben Data communication 10 Van Miecky Computer Networks

Dequeue 4 elements...

1 Jack Irwan Software Engineering

2 Billy Mckao Requirement Engineering

3 Nat Mcfaden Multivariate Calculus

4 Steven Shwimmer Software Architecture

Enqueue all elements...

Queue: 5 Ruby jason Relational DBMS 6 Mark Dyne PHP development 7 Philip namdaf Microsoft Dot Net Platform 8 Erik Bawn HTML & Scripting 9 Ricky ben Data communication 10 Van Miecky Computer Networks 1 Jack Irwan Software Engineering 2 Billy Mckao Requirement Engineering 3 Nat Mcfaden Multivariate Calculus 4 Steven Shwimmer Software Architecture 5 Ruby jason Relational DBMS 6 Mark Dyne PHP development 7 Philip namdaf Microsoft Dot Net Platform 8 Erik Bawn HTML & Scripting 9 Ricky ben Data communication 10 Van Miecky Computer Networks

Dequeue all elements...

5 Ruby jason Relational DBMS

6 Mark Dyne PHP development

7 Philip namdaf Microsoft Dot Net Platform

8 Erik Bawn HTML & Scripting

9 Ricky ben Data communication

10 Van Miecky Computer Networks

1 Jack Irwan Software Engineering

2 Billy Mckao Requirement Engineering

3 Nat Mcfaden Multivariate Calculus

4 Steven Shwimmer Software Architecture

5 Ruby jason Relational DBMS

6 Mark Dyne PHP development

7 Philip namdaf Microsoft Dot Net Platform

8 Erik Bawn HTML & Scripting

9 Ricky ben Data communication

10 Van Miecky Computer Networks

Dequeue 1 element...

Underflow: Empty Queue.

null

Enqueue all elements...

Queue: 1 Jack Irwan Software Engineering 2 Billy Mckao Requirement Engineering 3 Nat Mcfaden Multivariate Calculus 4 Steven Shwimmer Software Architecture 5 Ruby jason Relational DBMS 6 Mark Dyne PHP development 7 Philip namdaf Microsoft Dot Net Platform 8 Erik Bawn HTML & Scripting 9 Ricky ben Data communication 10 Van Miecky Computer Networks

Enqueue 11 John Johnson Java Programming...

Now queue is...

Queue: 1 Jack Irwan Software Engineering 2 Billy Mckao Requirement Engineering 3 Nat Mcfaden Multivariate Calculus 4 Steven Shwimmer Software Architecture 5 Ruby jason Relational DBMS 6 Mark Dyne PHP development 7 Philip namdaf Microsoft Dot Net Platform 8 Erik Bawn HTML & Scripting 9 Ricky ben Data communication 10 Van Miecky Computer Networks 11 John Johnson Java Programming

Print queue with the goal- Reverse order

Use q1 and q2, on new element dequeue all from q1 to q2 and enqueue new element to q1 and dequeue q2 and enqueue all to q1

Priting final queue-

10 Van Miecky Computer Networks

9 Ricky ben Data communication

8 Erik Bawn HTML & Scripting

7 Philip namdaf Microsoft Dot Net Platform

6 Mark Dyne PHP development

5 Ruby jason Relational DBMS

4 Steven Shwimmer Software Architecture

3 Nat Mcfaden Multivariate Calculus

2 Billy Mckao Requirement Engineering

1 Jack Irwan Software Engineering

Print queue with the goal- Original order

Enqueu all elements...then dequeue all to get original order

Dequeuing all elements in the original order...

1 Jack Irwan Software Engineering

2 Billy Mckao Requirement Engineering

3 Nat Mcfaden Multivariate Calculus

4 Steven Shwimmer Software Architecture

5 Ruby jason Relational DBMS

6 Mark Dyne PHP development

7 Philip namdaf Microsoft Dot Net Platform

8 Erik Bawn HTML & Scripting

9 Ricky ben Data communication

10 Van Miecky Computer Networks

BUILD SUCCESSFUL (total time: 0 seconds)

Output for Queue Array fix-

run:

Enqueue all elements...

Queue: first->1 Jack Irwan Software Engineering 2 Billy Mckao Requirement Engineering 3 Nat Mcfaden Multivariate Calculus 4 Steven Shwimmer Software Architecture 5 Ruby jason Relational DBMS 6 Mark Dyne PHP development 7 Philip namdaf Microsoft Dot Net Platform 8 Erik Bawn HTML & Scripting 9 Ricky ben Data communication 10 Van Miecky Computer Networks <-last

Dequeue 4 elements...

1 Jack Irwan Software Engineering

2 Billy Mckao Requirement Engineering

3 Nat Mcfaden Multivariate Calculus

4 Steven Shwimmer Software Architecture

Enqueue all elements...

Overflow

Overflow

Overflow

Overflow

Overflow

Overflow

Queue: first->5 Ruby jason Relational DBMS 6 Mark Dyne PHP development 7 Philip namdaf Microsoft Dot Net Platform 8 Erik Bawn HTML & Scripting 9 Ricky ben Data communication 10 Van Miecky Computer Networks 1 Jack Irwan Software Engineering 2 Billy Mckao Requirement Engineering 3 Nat Mcfaden Multivariate Calculus 4 Steven Shwimmer Software Architecture <-last

Dequeue all elements...

5 Ruby jason Relational DBMS

6 Mark Dyne PHP development

7 Philip namdaf Microsoft Dot Net Platform

8 Erik Bawn HTML & Scripting

9 Ricky ben Data communication

10 Van Miecky Computer Networks

1 Jack Irwan Software Engineering

2 Billy Mckao Requirement Engineering

3 Nat Mcfaden Multivariate Calculus

4 Steven Shwimmer Software Architecture

Dequeue 1 element...

Underflow!

null

Enqueue all elements...

Queue: first->1 Jack Irwan Software Engineering 2 Billy Mckao Requirement Engineering 3 Nat Mcfaden Multivariate Calculus 4 Steven Shwimmer Software Architecture 5 Ruby jason Relational DBMS 6 Mark Dyne PHP development 7 Philip namdaf Microsoft Dot Net Platform 8 Erik Bawn HTML & Scripting 9 Ricky ben Data communication 10 Van Miecky Computer Networks <-last

Enqueue 11 John Johnson Java Programming...

Overflow

Now queue is...

Queue: first->1 Jack Irwan Software Engineering 2 Billy Mckao Requirement Engineering 3 Nat Mcfaden Multivariate Calculus 4 Steven Shwimmer Software Architecture 5 Ruby jason Relational DBMS 6 Mark Dyne PHP development 7 Philip namdaf Microsoft Dot Net Platform 8 Erik Bawn HTML & Scripting 9 Ricky ben Data communication 10 Van Miecky Computer Networks <-last

Print queue with the goal- Reverse order

Use q1 and q2, on new element dequeue all from q1 to q2 and enqueue new element to q1 and dequeue q2 and enqueue all to q1

Dequeuing all elements in the Reverse order...

10 Van Miecky Computer Networks

9 Ricky ben Data communication

8 Erik Bawn HTML & Scripting

7 Philip namdaf Microsoft Dot Net Platform

6 Mark Dyne PHP development

5 Ruby jason Relational DBMS

4 Steven Shwimmer Software Architecture

3 Nat Mcfaden Multivariate Calculus

2 Billy Mckao Requirement Engineering

1 Jack Irwan Software Engineering

Print queue with the goal- Original order

Enqueu all elements...then dequeue all to get original order

Dequeuing all elements in the original order...

1 Jack Irwan Software Engineering

2 Billy Mckao Requirement Engineering

3 Nat Mcfaden Multivariate Calculus

4 Steven Shwimmer Software Architecture

5 Ruby jason Relational DBMS

6 Mark Dyne PHP development

7 Philip namdaf Microsoft Dot Net Platform

8 Erik Bawn HTML & Scripting

9 Ricky ben Data communication

10 Van Miecky Computer Networks

BUILD SUCCESSFUL (total time: 0 seconds)

**6. Consider following Algorithm to “Evaluate Infix Expressions” with Two arrays:**

**Test data:**

**(A + B) \* C + D / (E + F \* G) - H**

**(300 + 23) \* (43 - 21) / (84 + 7)**

**(4 + 8) \* (6 - 5)/((3 - 2) \* (2 + 2))**

**A) Step through algorithm to develop a Stack Table for for each Infix expression**

**B) Write Java code to test each Infix Expression**

**C) Compile and Run**

**Algorithm:**

Iterate through given expression, one character at a time

1. If the character is an operand, push it to the operand stack.

2. If the character is an operator,

1. If the operator stack is empty then push it to the operator stack.

2. Else If the operator stack is not empty,

• If the character’s precedence is greater than or equal to the precedence of the stack top of the operator stack, then push the character to the operator stack.

• If the character’s precedence is less than the precedence of the stack top of the operator stack then do Process (as explained above) until character’s precedence is less or stack is not empty.

3. If the character is “(“, then push it onto the operator stack.

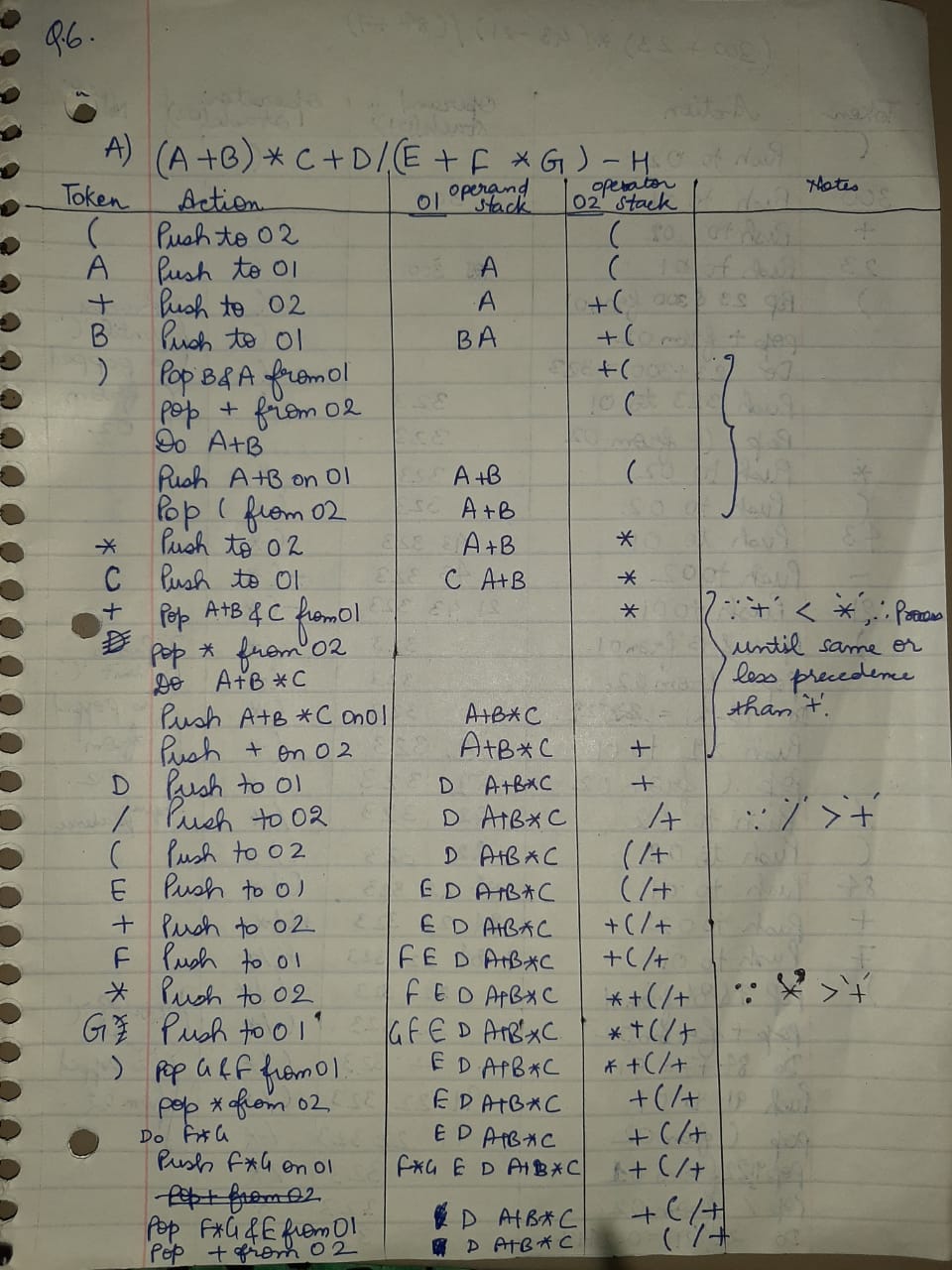
4. If the character is “)”, then do *Process* (as explained above) until the corresponding “(” is encountered in operator stack. Now just pop out the “(“.

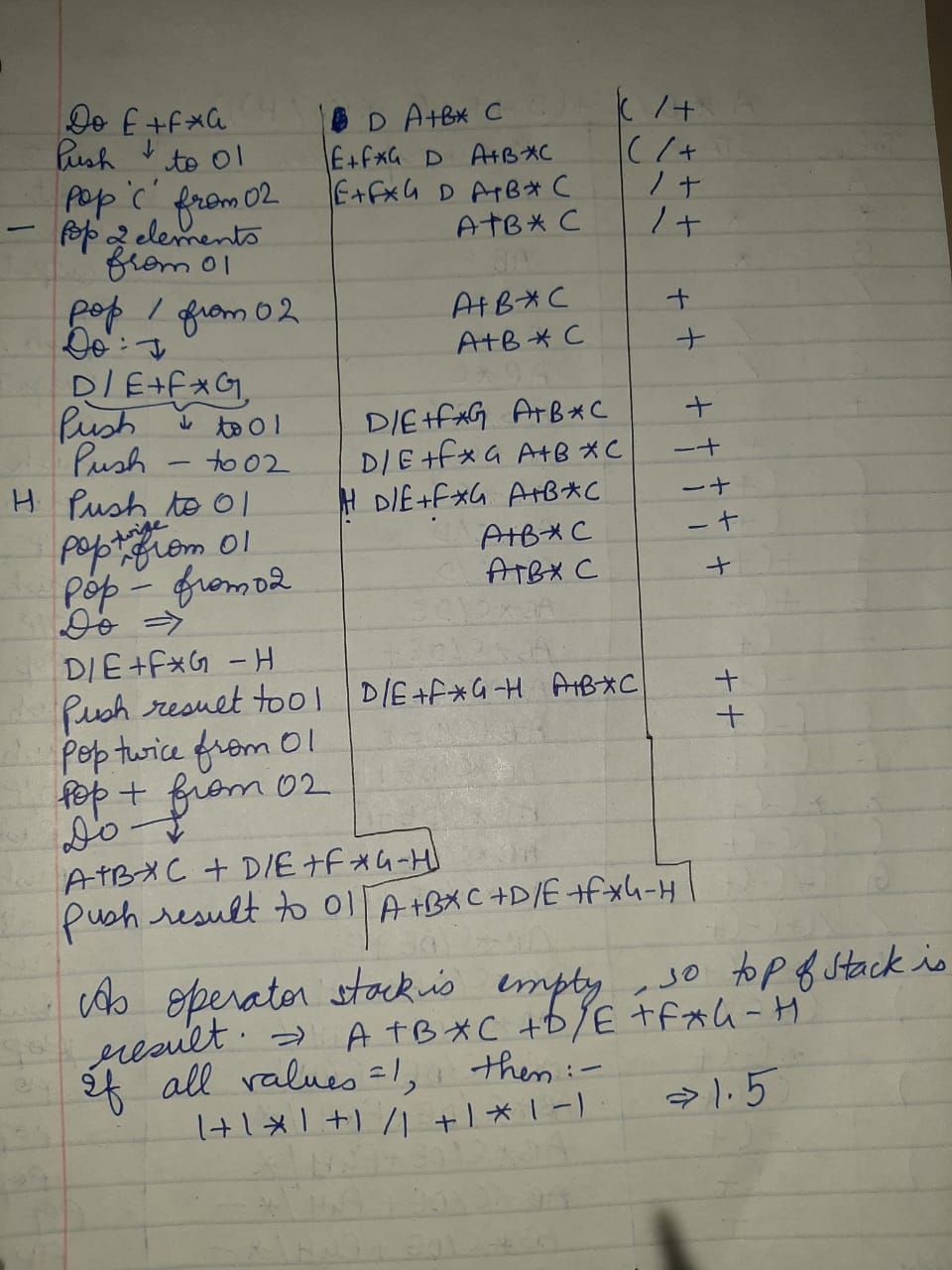
Once the expression iteration is completed and the operator stack is not empty,

do *Process* until the operator stack is empty.  The values left in the operand stack

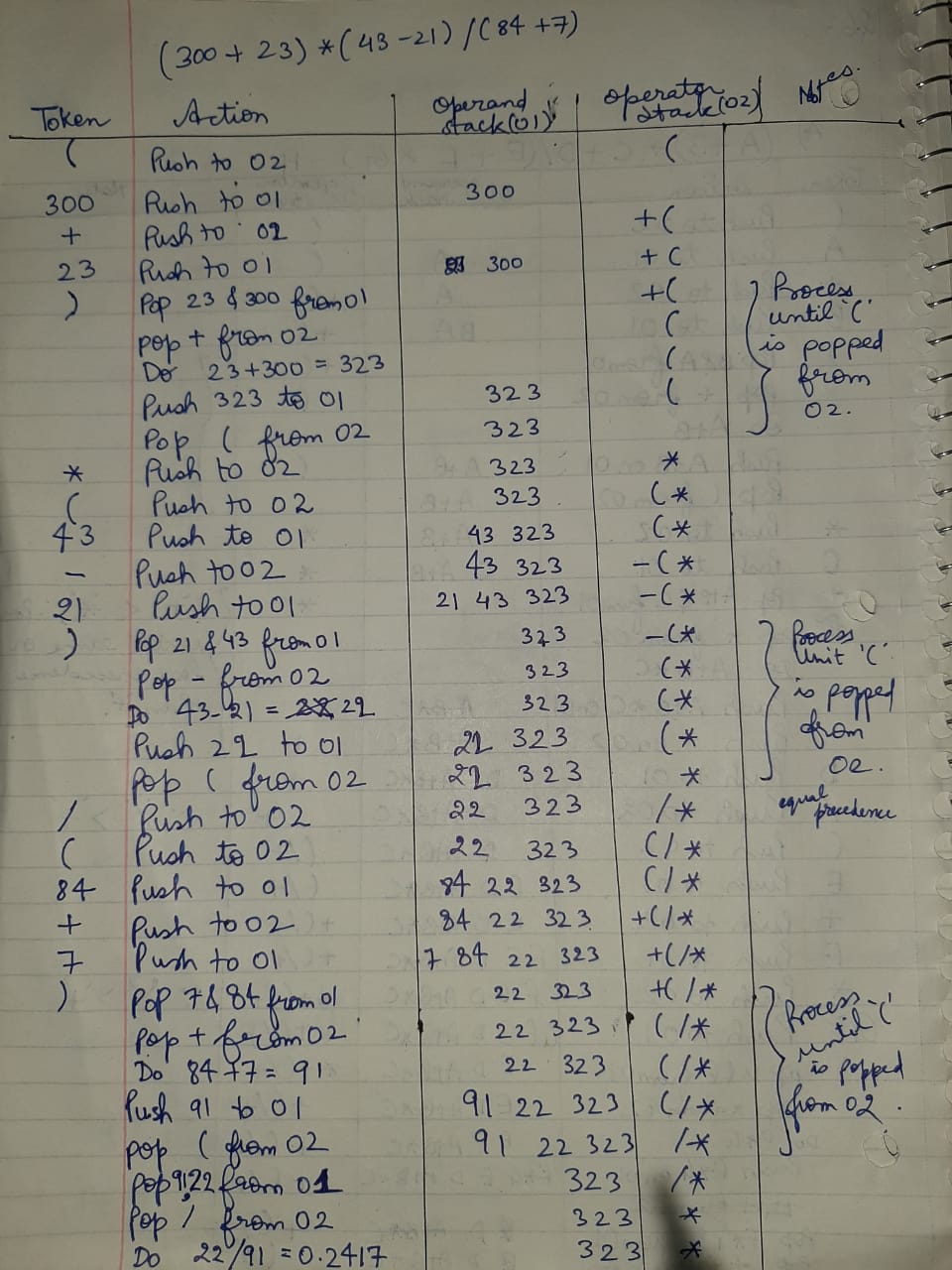
is our final result.

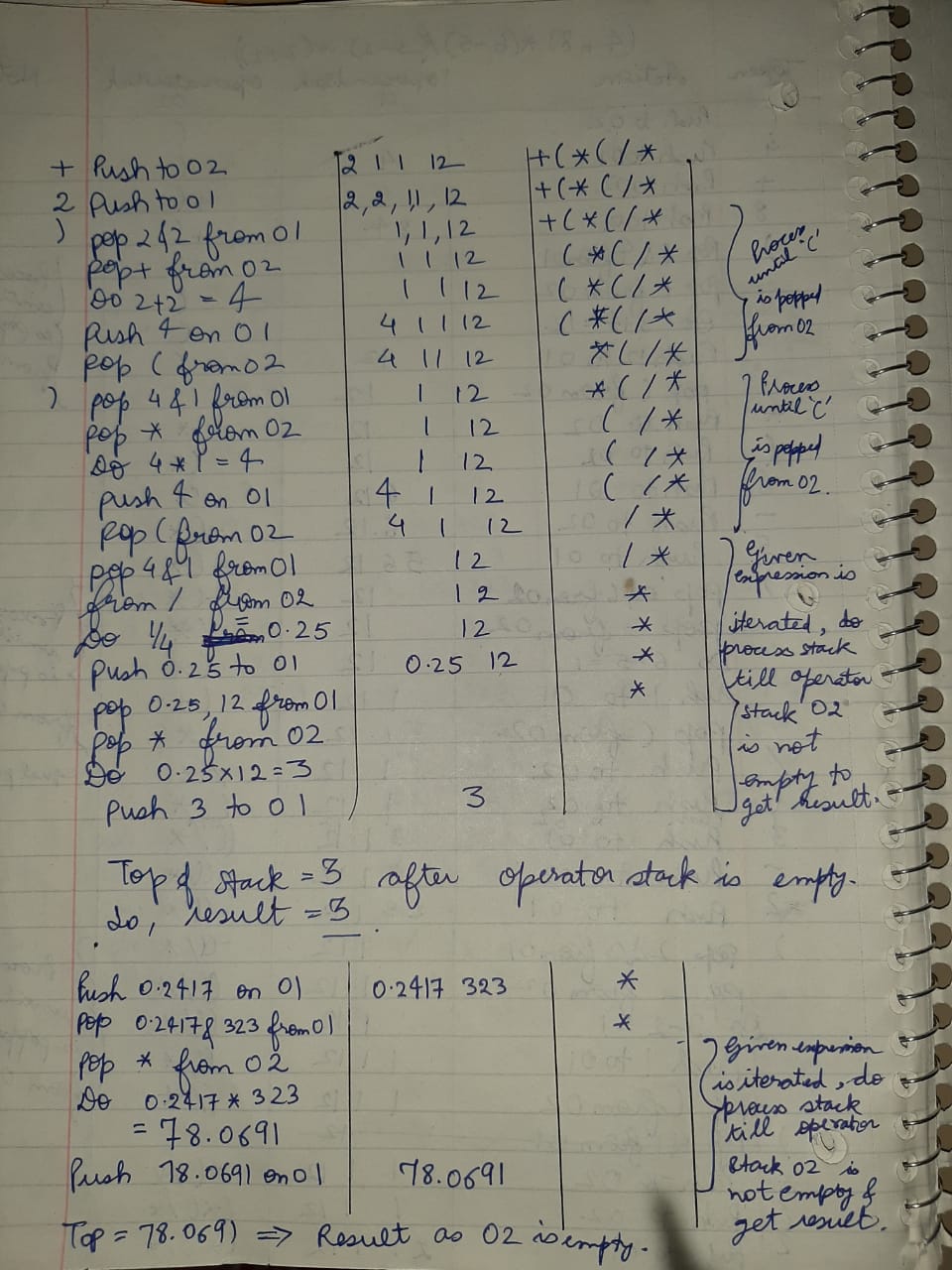
A)(A + B) \* C + D / (E + F \* G) - H



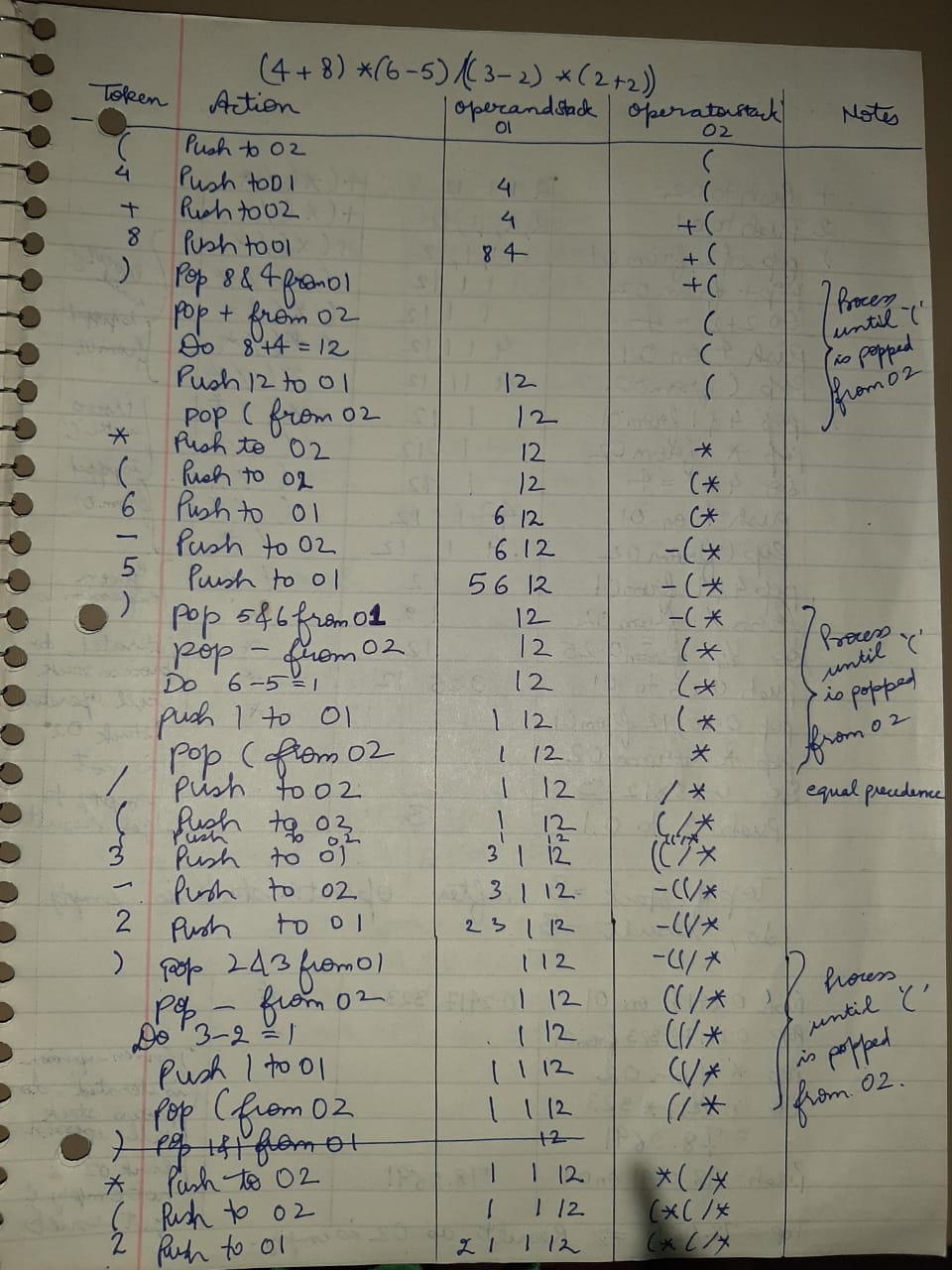


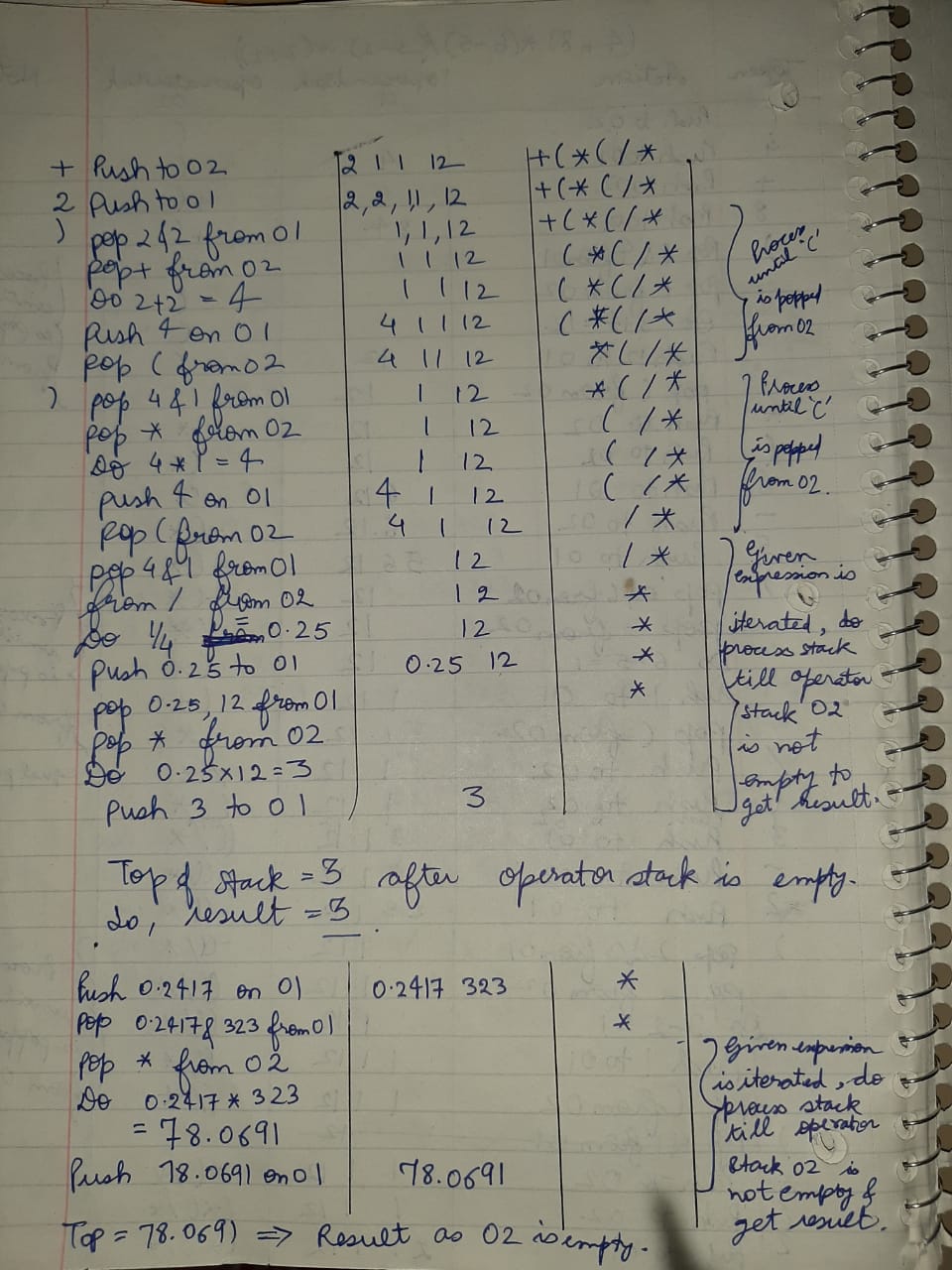
B) (300 + 23) \* (43 - 21) / (84 + 7)





1. (4 + 8) \* (6 - 5)/((3 - 2) \* (2 + 2))





Java code in file-- **Answers--- Java code in - Q6InfixEvaluateAditi.java**

Output—

run:

s is (

push (

Operator stack- [(]

Operand Stack- []

s is 300

push val-300

Operator stack- [(]

Operand Stack- [300.0]

s is +

push (

push +

Operator stack- [(, +]

Operand Stack- [300.0]

s is 23

push val-23

Operator stack- [(, +]

Operand Stack- [300.0, 23.0]

s is )

push 323.0

Operator stack- []

Operand Stack- [323.0]

s is \*

push ops

Operator stack- [\*]

Operand Stack- [323.0]

s is (

push (

Operator stack- [\*, (]

Operand Stack- [323.0]

s is 43

push val-43

Operator stack- [\*, (]

Operand Stack- [323.0, 43.0]

s is -

push (

push -

Operator stack- [\*, (, -]

Operand Stack- [323.0, 43.0]

s is 21

push val-21

Operator stack- [\*, (, -]

Operand Stack- [323.0, 43.0, 21.0]

s is )

push 22.0

Operator stack- [\*]

Operand Stack- [323.0, 22.0]

s is /

push \*

push /

Operator stack- [\*, /]

Operand Stack- [323.0, 22.0]

s is (

push (

Operator stack- [\*, /, (]

Operand Stack- [323.0, 22.0]

s is 84

push val-84

Operator stack- [\*, /, (]

Operand Stack- [323.0, 22.0, 84.0]

s is +

push (

push +

Operator stack- [\*, /, (, +]

Operand Stack- [323.0, 22.0, 84.0]

s is 7

push val-7

Operator stack- [\*, /, (, +]

Operand Stack- [323.0, 22.0, 84.0, 7.0]

s is )

push 91.0

Operator stack- [\*, /]

Operand Stack- [323.0, 22.0, 91.0]

push 0.24175824175824176

push 78.08791208791209

For input ( 300 + 23 ) \* ( 43 - 21 ) / ( 84 + 7 )output -> 78.08791208791209

s is (

push (

Operator stack- [(]

Operand Stack- []

s is 4

push val-4

Operator stack- [(]

Operand Stack- [4.0]

s is +

push (

push +

Operator stack- [(, +]

Operand Stack- [4.0]

s is 8

push val-8

Operator stack- [(, +]

Operand Stack- [4.0, 8.0]

s is )

push 12.0

Operator stack- []

Operand Stack- [12.0]

s is \*

push ops

Operator stack- [\*]

Operand Stack- [12.0]

s is (

push (

Operator stack- [\*, (]

Operand Stack- [12.0]

s is 6

push val-6

Operator stack- [\*, (]

Operand Stack- [12.0, 6.0]

s is -

push (

push -

Operator stack- [\*, (, -]

Operand Stack- [12.0, 6.0]

s is 5

push val-5

Operator stack- [\*, (, -]

Operand Stack- [12.0, 6.0, 5.0]

s is )

push 1.0

Operator stack- [\*]

Operand Stack- [12.0, 1.0]

s is /

push \*

push /

Operator stack- [\*, /]

Operand Stack- [12.0, 1.0]

s is (

push (

Operator stack- [\*, /, (]

Operand Stack- [12.0, 1.0]

s is (

push (

Operator stack- [\*, /, (, (]

Operand Stack- [12.0, 1.0]

s is 3

push val-3

Operator stack- [\*, /, (, (]

Operand Stack- [12.0, 1.0, 3.0]

s is -

push (

push -

Operator stack- [\*, /, (, (, -]

Operand Stack- [12.0, 1.0, 3.0]

s is 2

push val-2

Operator stack- [\*, /, (, (, -]

Operand Stack- [12.0, 1.0, 3.0, 2.0]

s is )

push 1.0

Operator stack- [\*, /, (]

Operand Stack- [12.0, 1.0, 1.0]

s is \*

push (

push \*

Operator stack- [\*, /, (, \*]

Operand Stack- [12.0, 1.0, 1.0]

s is (

push (

Operator stack- [\*, /, (, \*, (]

Operand Stack- [12.0, 1.0, 1.0]

s is 2

push val-2

Operator stack- [\*, /, (, \*, (]

Operand Stack- [12.0, 1.0, 1.0, 2.0]

s is +

push (

push +

Operator stack- [\*, /, (, \*, (, +]

Operand Stack- [12.0, 1.0, 1.0, 2.0]

s is 2

push val-2

Operator stack- [\*, /, (, \*, (, +]

Operand Stack- [12.0, 1.0, 1.0, 2.0, 2.0]

s is )

push 4.0

Operator stack- [\*, /, (, \*]

Operand Stack- [12.0, 1.0, 1.0, 4.0]

s is )

push 4.0

Operator stack- [\*, /]

Operand Stack- [12.0, 1.0, 4.0]

push 0.25

push 3.0

For input ( 4 + 8 ) \* ( 6 - 5 ) / ( ( 3 - 2 ) \* ( 2 + 2 ) )output -> 3.0

s is (

push (

Operator stack- [(]

Operand Stack- []

s is A

push val-A

Operator stack- [(]

Operand Stack- [A]

s is +

push (

push +

Operator stack- [(, +]

Operand Stack- [A]

s is B

push val-B

Operator stack- [(, +]

Operand Stack- [A, B]

s is )

push A + B

Operator stack- []

Operand Stack- [A + B]

s is \*

push ops

Operator stack- [\*]

Operand Stack- [A + B]

s is C

push val-C

Operator stack- [\*]

Operand Stack- [A + B, C]

s is +

push A + B \* C

push +

Operator stack- [+]

Operand Stack- [A + B \* C]

s is D

push val-D

Operator stack- [+]

Operand Stack- [A + B \* C, D]

s is /

push +

push /

Operator stack- [+, /]

Operand Stack- [A + B \* C, D]

s is (

push (

Operator stack- [+, /, (]

Operand Stack- [A + B \* C, D]

s is E

push val-E

Operator stack- [+, /, (]

Operand Stack- [A + B \* C, D, E]

s is +

push (

push +

Operator stack- [+, /, (, +]

Operand Stack- [A + B \* C, D, E]

s is F

push val-F

Operator stack- [+, /, (, +]

Operand Stack- [A + B \* C, D, E, F]

s is \*

push +

push \*

Operator stack- [+, /, (, +, \*]

Operand Stack- [A + B \* C, D, E, F]

s is G

push val-G

Operator stack- [+, /, (, +, \*]

Operand Stack- [A + B \* C, D, E, F, G]

s is )

push F \* G

push E + F \* G

Operator stack- [+, /]

Operand Stack- [A + B \* C, D, E + F \* G]

s is -

push D / E + F \* G

push -

Operator stack- [+, -]

Operand Stack- [A + B \* C, D / E + F \* G]

s is H

push val-H

Operator stack- [+, -]

Operand Stack- [A + B \* C, D / E + F \* G, H]

push D / E + F \* G - H

push A + B \* C + D / E + F \* G - H

For input ( A + B ) \* C + D / ( E + F \* G ) - Houtput -> A + B \* C + D / E + F \* G - H

BUILD SUCCESSFUL (total time: 0 seconds)

**7. Consider the following Algorithm to convert Infix expression to Postfix.**

**A) Infix expression example: A \* B / C + (D + E - (F \* (G / H)))**

**B) Apply Algorithm to Infix example, show step-by-step**

**C) Write Java code for the algorithm to convert Infix to Postfix expression**

A \* B / C + (D + E - (F \* (G / H)))

Algorithm:

while there are more symbols to read

read the next symbol

case:

operand --> output it.

’(’ --> push it on the stack.

’)’ --> pop operators from the stack to output

until a ’(’ is popped; do not output either of

the parentheses.

operator --> pop higher- or equal-precedence operators

from the stack to the output; stop before

popping a lower-precedence operator or

a ’(’. Push the operator on the stack.

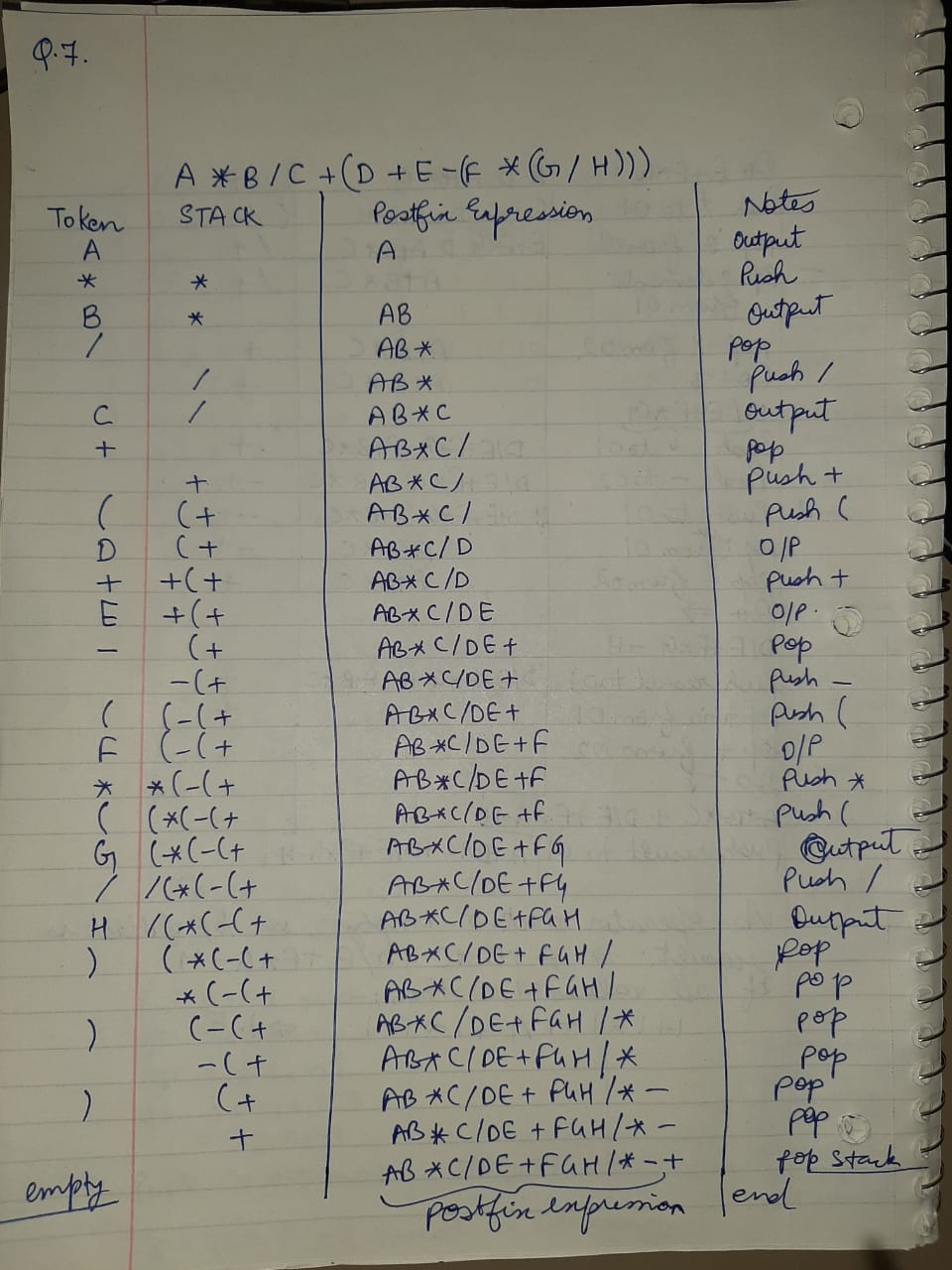
end case

end while

pop the remaining operators from the stack to the output

Answer-🡪

Java code in file-- Q7InfixtoPostfixAditi.java



Output-- run:

word -A

Stack is-> []

List is-> [A]

word -

word -\*

Stack is-> [\*]

List is-> [A]

word -

word -B

Stack is-> [\*]

List is-> [A, B]

word -

word -/

Stack is-> [/]

List is-> [A, B, \*]

word -

word -C

Stack is-> [/]

List is-> [A, B, \*, C]

word -

word -+

Stack is-> [+]

List is-> [A, B, \*, C, /]

word -

word -(

Stack is-> [+, (]

List is-> [A, B, \*, C, /]

word -

word -D

Stack is-> [+, (]

List is-> [A, B, \*, C, /, D]

word -

word -+

Stack is-> [+, (, +]

List is-> [A, B, \*, C, /, D]

word -

word -E

Stack is-> [+, (, +]

List is-> [A, B, \*, C, /, D, E]

word -

word --

Stack is-> [+, (, -]

List is-> [A, B, \*, C, /, D, E, +]

word -

word -(

Stack is-> [+, (, -, (]

List is-> [A, B, \*, C, /, D, E, +]

word -

word -F

Stack is-> [+, (, -, (]

List is-> [A, B, \*, C, /, D, E, +, F]

word -

word -\*

Stack is-> [+, (, -, (, \*]

List is-> [A, B, \*, C, /, D, E, +, F]

word -

word -(

Stack is-> [+, (, -, (, \*, (]

List is-> [A, B, \*, C, /, D, E, +, F]

word -

word -G

Stack is-> [+, (, -, (, \*, (]

List is-> [A, B, \*, C, /, D, E, +, F, G]

word -

word -/

Stack is-> [+, (, -, (, \*, (, /]

List is-> [A, B, \*, C, /, D, E, +, F, G]

word -

word -H

Stack is-> [+, (, -, (, \*, (, /]

List is-> [A, B, \*, C, /, D, E, +, F, G, H]

word -

word -)

Stack is-> [+, (, -, (, \*]

List is-> [A, B, \*, C, /, D, E, +, F, G, H, /]

word -

word -)

Stack is-> [+, (, -]

List is-> [A, B, \*, C, /, D, E, +, F, G, H, /, \*]

word -

word -)

Stack is-> [+]

List is-> [A, B, \*, C, /, D, E, +, F, G, H, /, \*, -]

answer-->

AB\*C/DE+FGH/\*-+BUILD SUCCESSFUL (total time: 0 seconds)