K. J. Somaiya College of Engineering, Mumbai-77
(A constituent College of SomaiyaVidyavihar University)

Batch: C1	**Roll. No.:16010122257**
Experiment:10	
Grade: AA / AB / BB / BC / CC / CD /DD	

Title: Using virtual labs to understand the data structures

Objective: Use of virtual labs to understand the concepts and theory with examples and verify the same with practice questions.

Expected Outcome of Experiment:

CO	Outcome
CO1	Explain the different data structures used in problem solving
CO2	Apply linear and non-linear data structure in application development
CO3	Demonstrate sorting and searching methods.

Websites/books referred:

1. *https://www.geeksforgeeks.org/convert-infix-expression-to-postfix-expression/*

2. *https://donsheehy.github.io/datastructures/fullbook.pdf*

3.Our course's Google Classroom slides

Abstract: the virtual lab experiments help in understanding how various data structures work. They also emphasize on some important applications of various data structures and enable students to get familiarized with how certain applications can benefit from the choice of data structures.

Assigned data structure: *(Teacher would assign one of the following to one student)*

1. Stack - https://ds1-iiith.vlabs.ac.in/exp/stacks-queues/stacks/stackdemo.html
2. Infix and postfix - https://ds1-iiith.vlabs.ac.in/exp/infix-postfix/evaluation-of-postfix-expressions/postfix_eval.html
3. Queue - https://ds1-iiith.vlabs.ac.in/exp/stacks-queues/stacks/stackdemo.html
4. Bubble sort - https://ds1-iiith.vlabs.ac.in/exp/bubble-sort/bubble-sort/bsexercise.html
5. Graph DFS - https://ds1-iiith.vlabs.ac.in/exp/depth-first-search/index.html
6. Graph BFS - https://ds1-iiith.vlabs.ac.in/exp/breadth-first-search/index.html
7. Binary search tree - https://ds1-iiith.vlabs.ac.in/exp/binary-search-trees/bst-insert/bstInsert.html
8. Hash tables - https://ds1-iiith.vlabs.ac.in/exp/hash-tables/quadratic-probing/qp_practice.html
9. Linked list - https://ds1-iiith.vlabs.ac.in/exp/linked-list/singly-linked-list/sllpractice.html

Aim / learning objective of the assigned expt:

In this experiment we will be able to learn the following topics :

Formal Definitions of Infix and Postfix expressions.

Basic concepts behind Infix to Postfix conversion.

Conversion methods from Infix to Postfix.

Evaluation method of Postfix Expressions.

Concept and algorithm of the application/activity followed:

Infix: operand operator operand

– E.g a+b

• Postfix: operand operand operator

– E.g. a b +

• Operator Precedence

– ^ - exponential operator

– *, /

– +, -

Infix to postfix process without parenthesis

• Create an empty stack called opstack for keeping operators. Create an empty list for output.

• Scan the input string from left to right.

– If the input is an operand, append it to the end of the output list.

– If the token is an operator, *, /, +, or -, push it on the opstack. However, first remove any operators already on the opstack that have higher or equal precedence and append them to the output list.

• When the input expression has been completely processed, check the opstack. Any operators still on the stack can be removed and appended to the end of the output list.

Infix to postfix process with parenthesis

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.

1. Scan X from left to right and repeat Step 2 to 5 for each element of X until the Stack is empty.
2. If an operand is encountered, add it to Y
3. If a left parenthesis is encountered, push it onto Stack.
4. If an operator is encountered ,then:

– Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator until an opening parenthesis is encountered.

– Add operator to Stack.

5. If a right parenthesis is encountered ,then:

– Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.

– Remove the left Parenthesis.

6. END.

3. Evaluation of postfix expression

• Create a stack for storing operands

• Scan the input expression from left to right

– If the element is operand, push it onto the

stack

– If the element is operator, pop two

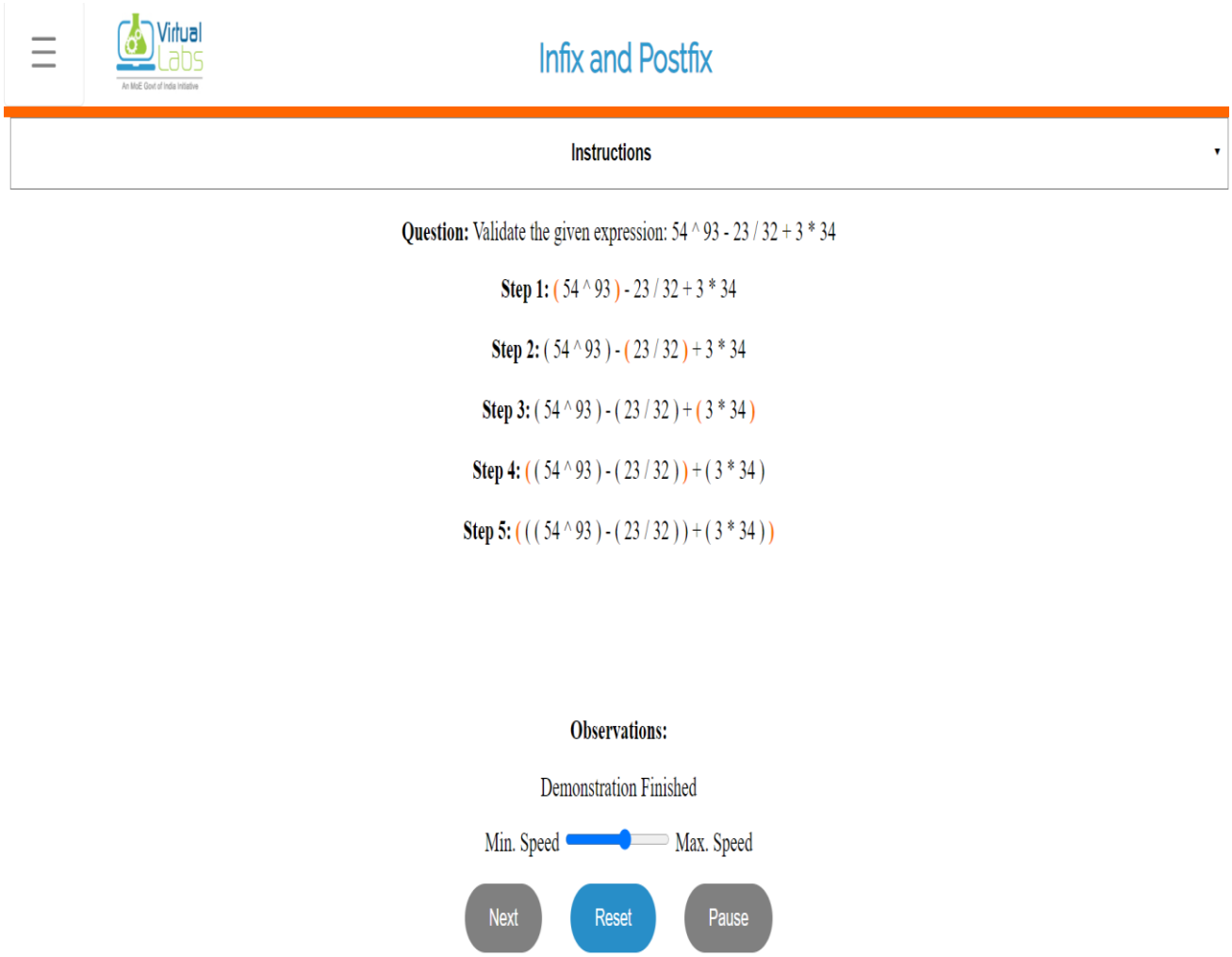
operands, evaluate and push the result

onto the stack

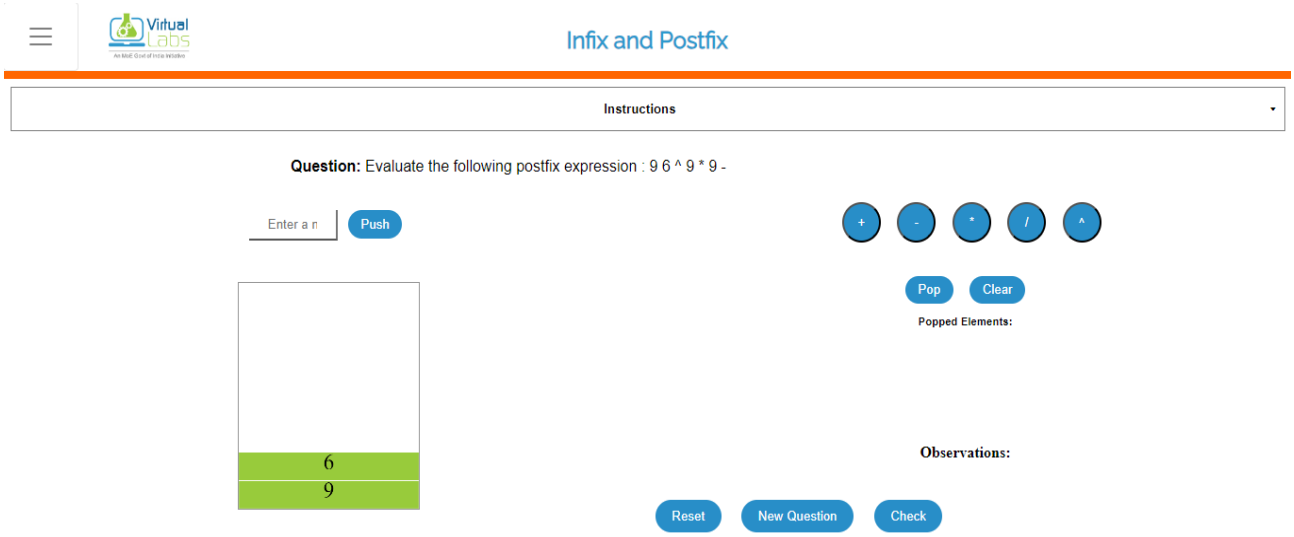
• If the expression is over, the stack

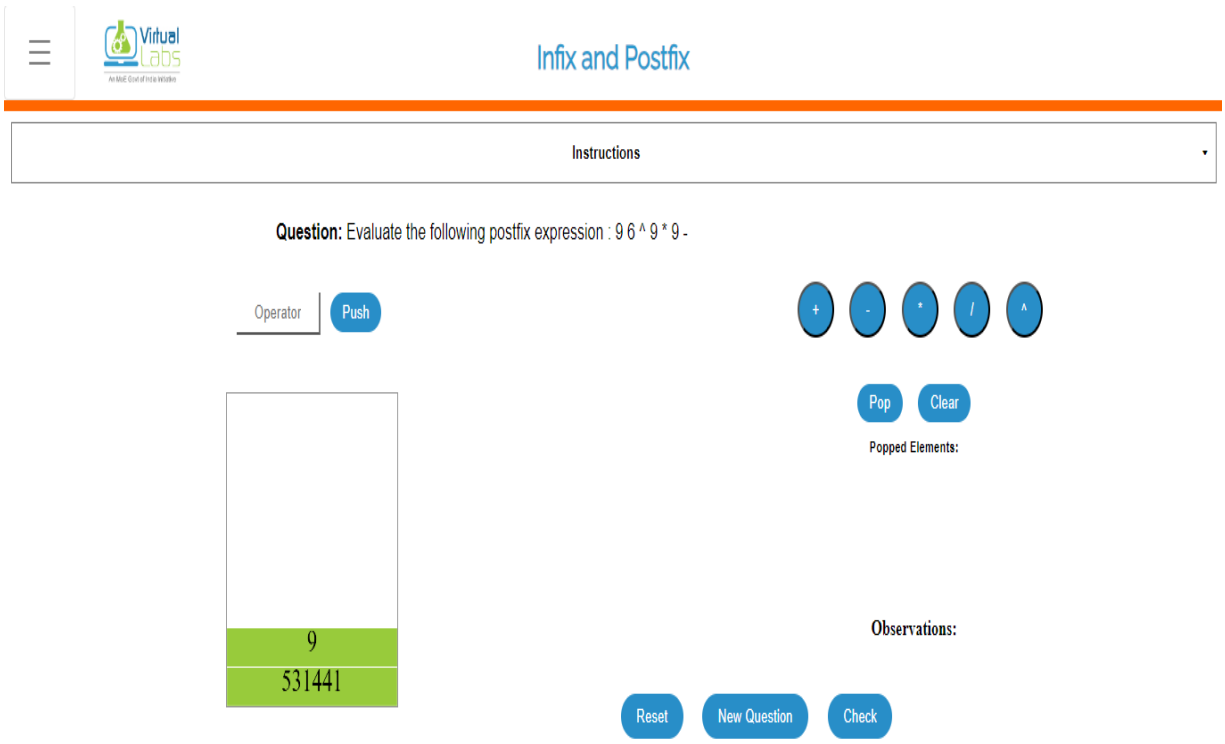
contains the final answer

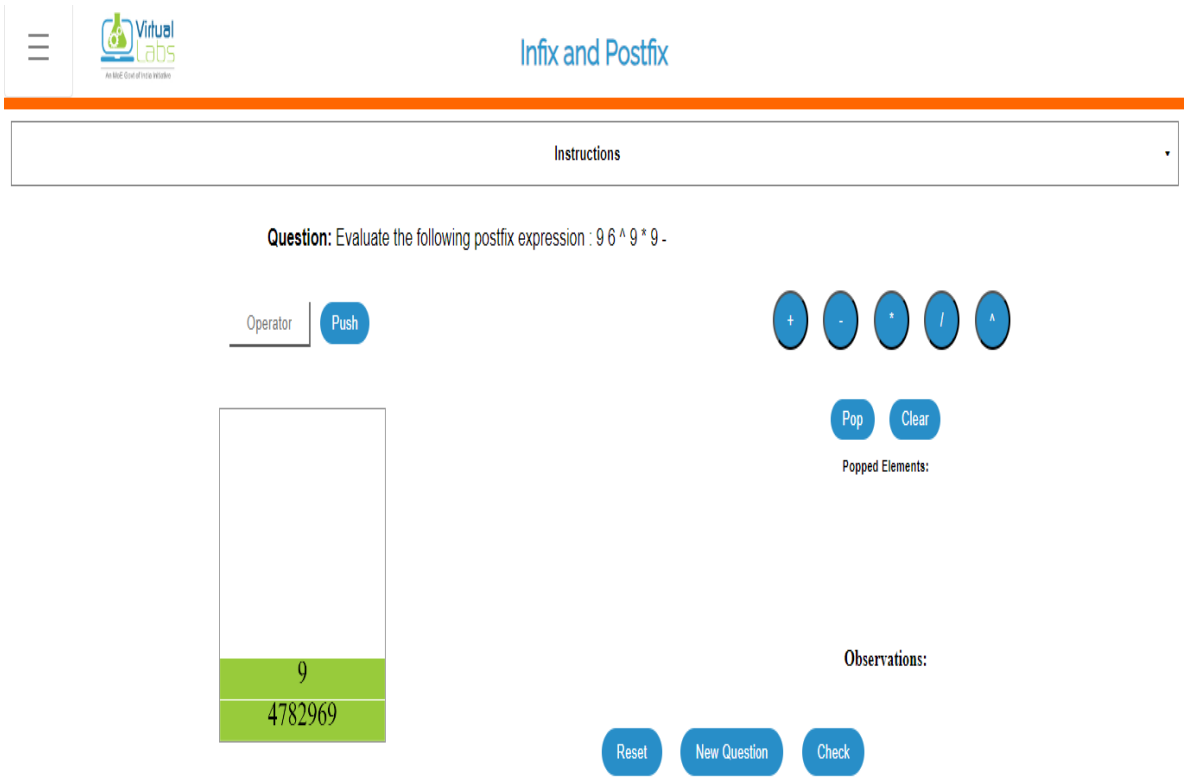
Demo execution screenshots:

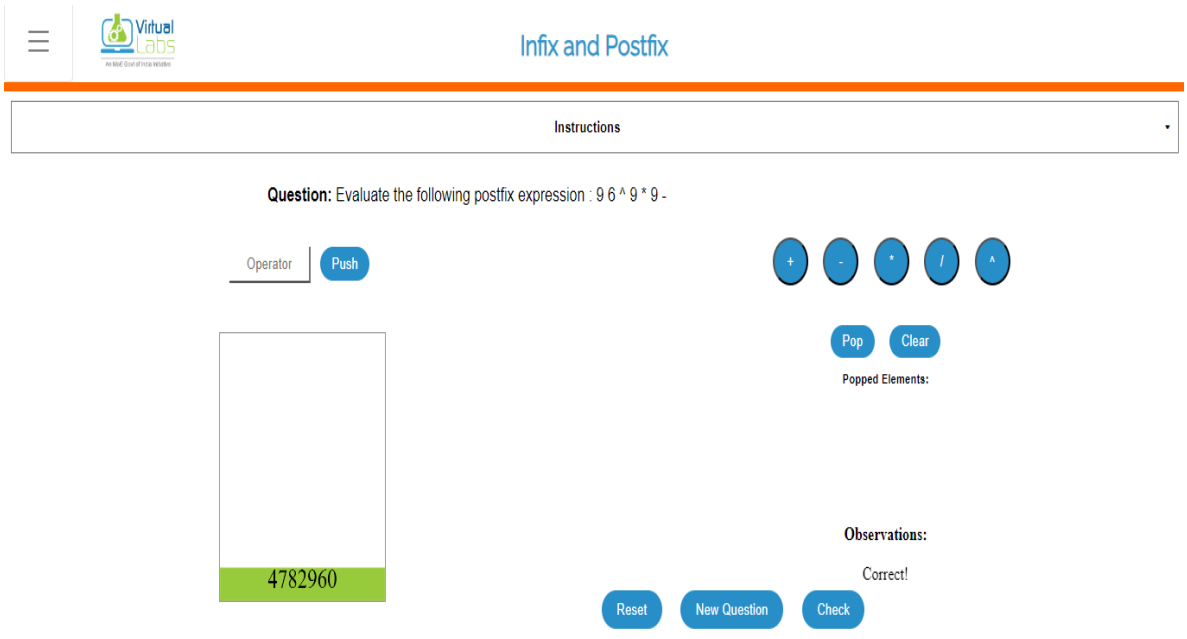


Practice problem screenshots:

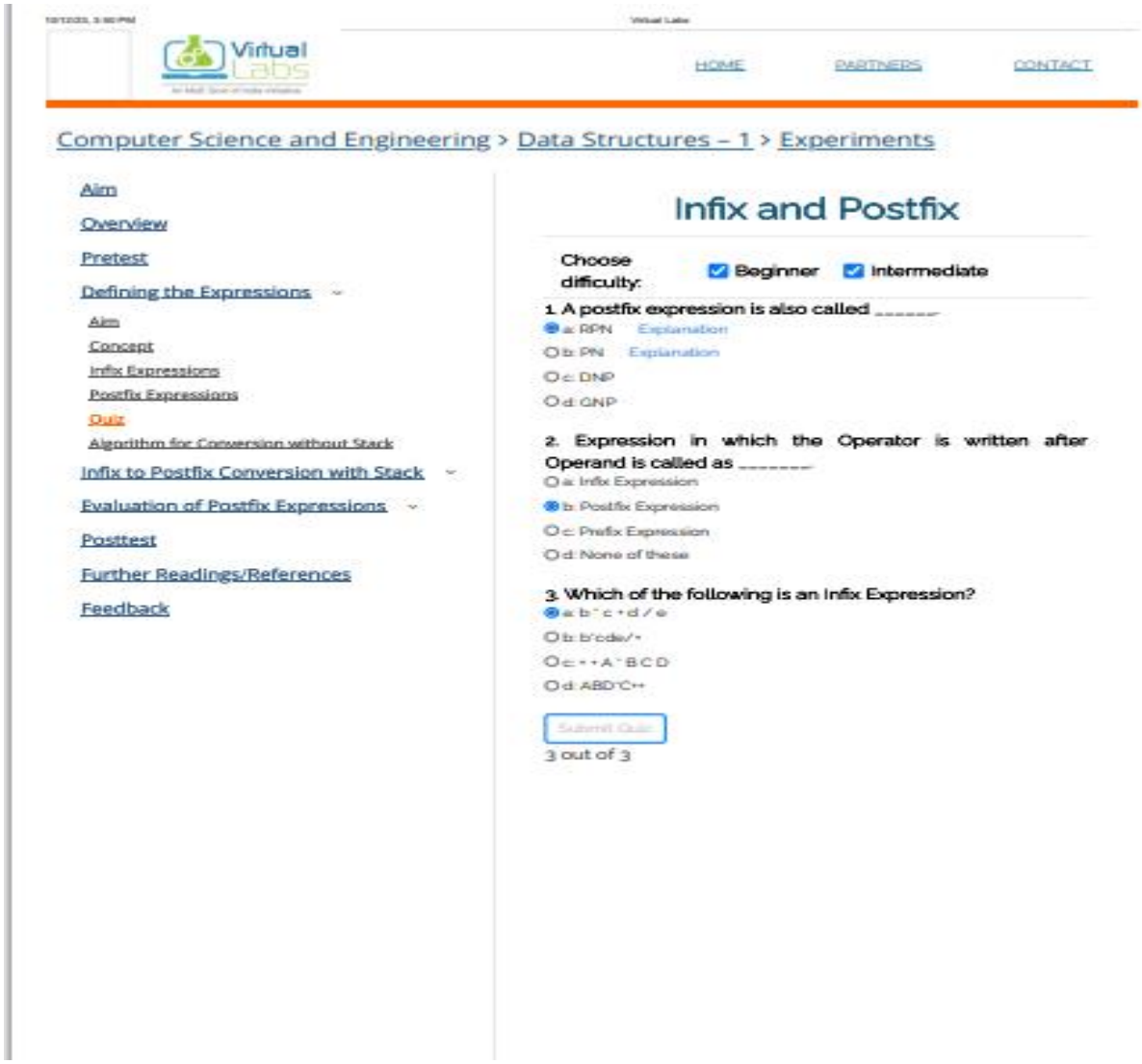


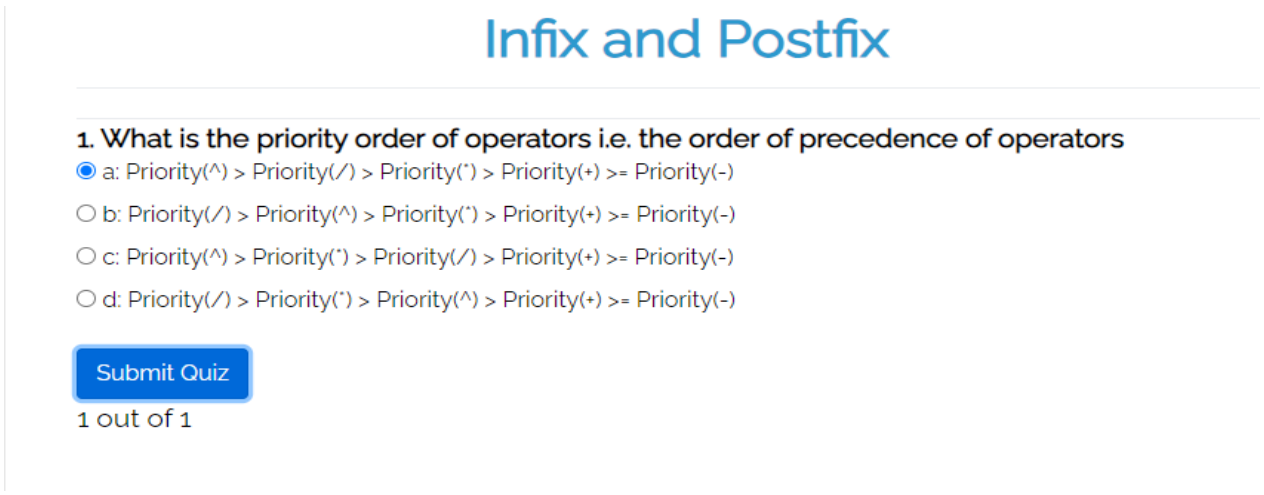


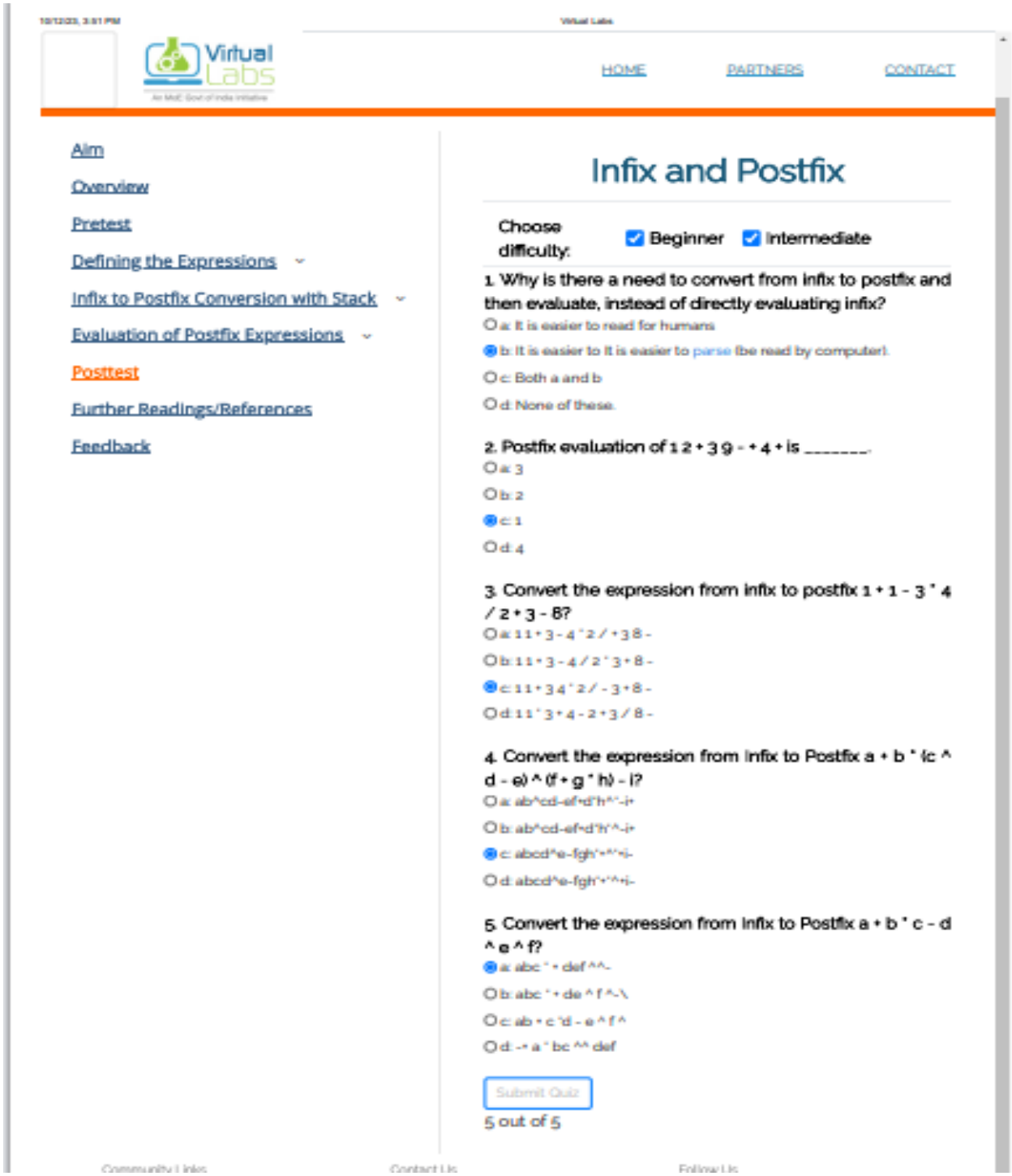

Infix and Postfix
Instructions
Question: Evaluate the following postfix expression : 9 6 ^ 9 * 9 -
Operator
Push
+ - * / ^
Pop
Clear
Popped Elements:
9
4782969
Observations:
Reset
New Question
Check


Infix and Postfix
Instructions
Question: Evaluate the following postfix expression : 9 6 ^ 9 * 9 -
Operator
Push
+ - * / ^
Pop
Clear
Popped Elements:
4782960
Observations:
Correct!
Reset
New Question
Check

Quiz screenshots:


Virtual Labs
HOME
PARTNERS
CONTACT
Computer Science and Engineering > Data Structures – 1 > Experiments
Aim
Overview
Pretest
Defining the Expressions
Aim
Concept
Infix Expressions
Postfix Expressions
Quiz
Algorithm for Conversion without Stack
Infix to Postfix Conversion with Stack
Evaluation of Postfix Expressions
Posttest
Further Readings/References
Feedback
Infix and Postfix
Choose difficulty:
Beginner
Intermediate
1 A postfix expression is also called _____
a: RPN Explanation
b: PN Explanation
c: DNP
d: QNP
2. Expression in which the Operator is written after Operand is called as _____
a: Infix Expression
b: Postfix Expression
c: Prefix Expression
d: None of these
3. Which of the following is an Infix Expression?
a: b * c + d / e
b: b'cde/+
c: + + A * B C D
d: ABD*C++
Submit Quiz
3 out of 3


Infix and Postfix
1. What is the priority order of operators i.e. the order of precedence of operators
a: Priority(^) > Priority(/) > Priority(*) > Priority(+) >= Priority(-)
b: Priority(/) > Priority(^) > Priority(*) > Priority(+) >= Priority(-)
c: Priority(^) > Priority(*) > Priority(/) > Priority(+) >= Priority(-)
d: Priority(/) > Priority(*) > Priority(^) > Priority(+) >= Priority(-)
Submit Quiz
1 out of 1


Virtual Labs
HOME
PARTNERS
CONTACT
Aim
Overview
Pretest
Defining the Expressions
Infix to Postfix Conversion with Stack
Evaluation of Postfix Expressions
Posttest
Further Readings/References
Feedback
Infix and Postfix
Choose difficulty:
Beginner
Intermediate
1. Why is there a need to convert from infix to postfix and then evaluate, instead of directly evaluating infix?
a: It is easier to read for humans
b: It is easier to It is easier to parse (be read by computer).
c: Both a and b
d: None of these.
2. Postfix evaluation of 1 2 + 3 9 - * 4 + is _____.
a: 3
b: 2
c: 1
d: 4
3. Convert the expression from infix to postfix 1 + 1 - 3 * 4 / 2 + 3 - 8?
a: 1 1 + 3 - 4 * 2 / + 3 8 -
b: 1 1 + 3 - 4 / 2 * 3 + 8 -
c: 1 1 + 3 4 * 2 / - 3 + 8 -
d: 1 1 * 3 + 4 - 2 + 3 / 8 -
4. Convert the expression from infix to Postfix a + b * (c ^ d - e) ^ (f + g * h) - i?
a: ab^cd-ef+d*h^^-i+
b: ab^cd-ef+d*h^^-i+
c: abcd^e-fgh*+^^-i-
d: abcd^e-fgh*+^^-i-
5. Convert the expression from Infix to Postfix a + b * c - d ^ e ^ f?
a: abc * + def ^^-
b: abc * + de ^ f ^-\
c: ab + c *d - e ^ f ^
d: -+ a * bc ^^ def
Submit Quiz
5 out of 5
Community Links
Contact Us
Follow Us

Conclusion and your take away after performing the virtual lab experiment: -

In conclusion,our virtual lab experiment to convert infix expressions to postfix notation and subsequently evaluate them has provided valuable insights into the fundamentals of computer science and the design of algorithms.

Through this hands-on experience,we've delved into the intricacies of parsing mathematical expressions,manipulating data structures,and implementing efficient algorithms.(Take-away)