

DATA STRUCTURES & ALGORITHMS

#07

Stack & queue

|  |
| --- |
| Student Name: |
| Roll Number: Section: |
| Work submitted on: |

|  |  |  |  |
| --- | --- | --- | --- |
| **Maximum Marks** | **Performance** | **Viva** | **Total** |
| **Marks Obtained** |  |  |  |
| **Remarks (if any)** |  | | |
|  | | | |
| **Experiment evaluated by** | | | |
| Instructor Name: | | | |
| Signature: | | | |

|  |
| --- |
| Stack and Queue Related Tasks |

**Task 1**

Write a program in Java to reverse a sentence using a stack. Use a stack to push all the words of the sentence, and then pop them out to form the reversed sentence.

Outcome :

Input: "The quick brown fox jumps over the lazy dog."

Output: "dog. lazy the over jumps fox brown quick The"

Explanation: The program uses a stack to push all the words of the input sentence, and then pops them out in reverse order to form the output sentence.

**Task 2:**

Write a program in Java to evaluate a postfix expression using a stack. Parse the expression and push the operands onto the stack, then pop them out and perform the operations.

In a postfix expression,

• an operator is written after its operands.

• the infix expression 2+3 is 23+ in postfix notation.

• For postfix expressions, operations are performed in the order in which they are written (left to right).

• No parentheses are necessary. ‘

• the infix expression 2+3\*4 is 234\*+ in postfix notation

• the infix expression 3\*4+2\*5 translates to 34\*25\*+ in postfix notation.

• the infix expression 3\*(4+2)\*5 translates to 342+\*5\*

Outcome:

Input: "5 3 8 \* + 4 -"

Output: 29

Explanation: The postfix expression "5 3 8 \* + 4 -" can be evaluated using a stack in the following way:

Push the operands onto the stack in the order they appear: 5, 3, 8.

When encountering the multiplication operator, pop the top two elements from the stack (8 and 3), multiply them, and push the result (24) onto the stack.

When encountering the addition operator, pop the top two elements from the stack (24 and 5), add them, and push the result (29) onto the stack.

When encountering the subtraction operator, pop the top two elements from the stack (29 and 4), subtract them, and push the result (25) onto the stack.

The final result is the top element of the stack (25).

Therefore, the output of the program would be 29.

**Task 3:**

Write a program in Java to check if a string is a palindrome using a stack. Push the first half of the string onto the stack, and then pop elements and compare them with the second half of the string.

Input: "racecar"

Output: true

Explanation: To check if the string "racecar" is a palindrome using a stack, we can do the following:

Push the first half of the string ("rac") onto the stack.

If the length of the string is odd, skip the middle character ("e").

Pop each element from the stack and compare it with the corresponding element in the second half of the string ("car").

If all elements match, the string is a palindrome.

In this case, the first half of the string ("rac") is pushed onto the stack. When we start popping elements from the stack, we get "c", "a", and "r", which match the second half of the string ("car") in reverse order. Therefore, the output of the program would be true, indicating that "racecar" is a palindrome.

**Task 4:**

Write a program in Java to convert an infix expression to a postfix expression using a stack. Parse the infix expression and push operators onto the stack according to precedence rules, and then pop them out to form the postfix expression.

**Input: "3 + 4 \* 2 / ( 1 - 5 ) ^ 2 ^ 3"**

**Output: "3 4 2 \* 1 5 - 2 3 ^ ^ / +"**

* Explanation: To convert the infix expression "3 + 4 \* 2 / ( 1 - 5 ) ^ 2 ^ 3" to postfix using a stack, we can do the following:
* Create an empty stack and an empty output string.
* Iterate through each token (number or operator) in the infix expression from left to right.
* If the token is a number, add it to the output string.
* If the token is an operator, compare its precedence with the top operator on the stack.
* If the token has higher precedence, push it onto the stack.
* If the token has lower or equal precedence, pop operators from the stack and add them to the output string until the top operator has lower precedence than the token, then push the token onto the stack.
* If the token is a left parenthesis, push it onto the stack.
* If the token is a right parenthesis, pop operators from the stack and add them to the output string until a left parenthesis is encountered (which is then discarded).
* After all tokens have been processed, pop any remaining operators from the stack and add them to the output string.

**Task 5:**

Write a program in Java to implement a stack that can find the minimum element in constant time. Use two stacks, one to hold the elements and one to hold the minimum values.

Input: Push 3, 5, 2, 7, 1 onto the stack

Output: Min = 1