Practical-2

Aim : Write a program to deal with real-world situations where Stack data structure is widely used.

<u>Evaluation of expression:</u>

Stacks are used to evaluate expressions, especially in languages that use postfix or prefix notation. Operators and operands are pushed onto the stack, and operations are performed based on the LIFO principle.

Algorithm:-

1. **Initialize an Empty Stack**:
   - Start by creating an empty stack. This stack will be used to store numbers (operands) as you process the expression.
2. **Iterate Through Each Character**:
   - Go through each character in the expression string one by one.
3. **Check If the Character is a Digit**:
   - If the character is a digit (e.g., '0', '1', '2', ..., '9'):
     - **Convert the Character to an Integer**: Use a method (like **Character.getNumericValue**) to convert the character to its integer value.
     - **Push the Integer onto the Stack**: Add this integer to the top of the stack.
4. **Handle Operators**:
   - If the character is an operator (like '+', '-', '*', or '/'):
     - **Pop Two Operands from the Stack**: Remove the top two numbers from the stack. Let's call them **operand1** and **operand2**. Note that **operand1** is the most recently added number, and **operand2** is the one below it.
     - **Perform the Operation**: Depending on the operator, perform the corresponding arithmetic operation:
       - For **+**, add **operand2** and **operand1**.
       - For **-**, subtract **operand2** from **operand1**.
       - For *****, multiply **operand2** by **operand1**.

        - For **/**, divide **operand2** by **operand1** (make sure to handle division by zero if you were to implement error handling).
    - **Push the Result Back onto the Stack**: After performing the operation, push the result back onto the stack.

5. **Final Result**:
    - Once you have processed all characters in the expression:
        - The stack should contain only one element, which is the final result of the expression.
        - **Pop This Element**: Remove this final result from the stack and return it as the output of the evaluation.

Example:-

- The postfix expression "53 2 1 - * 4 +" is evaluated as follows:
    1. Push 5, 3, 2, and 1 onto the stack.
    2. Subtract 2 - 1 to get 1, and push it back onto the stack.
    3. Multiply 3 * 1 to get 3, and push it back onto the stack.
    4. Push 4 onto the stack.
    5. Add 3 + 4 to get 7, and push it back onto the stack.

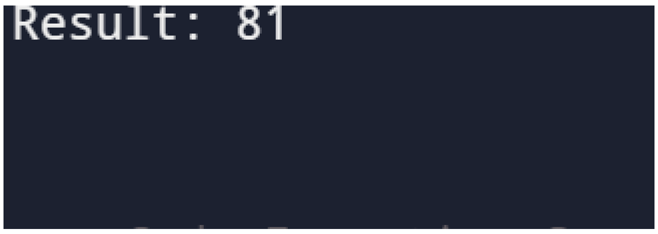
The final result of the expression is 7.

Program:-

```
#include <stdio.h>  // Include standard input/output library for printf and scanf
#include <stdlib.h> // Include standard library for memory allocation (malloc) and exit
#include <ctype.h>  // Include ctype library for character handling functions like isdigit

#define MAX_SIZE 100 // Define maximum size for the stack

// Stack structure to hold the stack data
typedef struct {
   int top;            // Index of the top element in the stack
```

```
    int items[MAX_SIZE];  // Array to store stack elements
} Stack;

// Function to create a stack
Stack* createStack() {
    Stack* stack = (Stack*)malloc(sizeof(Stack)); // Allocate memory for a new Stack
// pointer variable named stack of type Stack*.
    stack->top = -1; // Initialize the stack as empty by setting top to -1
    return stack;    // Return the pointer to the newly created stack
}

// Function to check if the stack is empty
int isEmpty(Stack* stack) {
    return stack->top == -1; // Return 1 (true) if top is -1, indicating the stack is empty
}

// Function to push an element onto the stack
void push(Stack* stack, int value) {
    if (stack->top < MAX_SIZE - 1) { // Check if there is space in the stack
        stack->items[++stack->top] = value; // Increment top and add value to the stack
    } else {
        printf("Stack overflow\n"); // Print error message if stack is full
    }
}

// Function to pop an element from the stack
int pop(Stack* stack) {
    if (!isEmpty(stack)) { // Check if the stack is not empty
        return stack->items[stack->top--]; // Return the top item and decrement top
    } else {
        printf("Stack underflow\n"); // Print error message if stack is empty
        exit(EXIT_FAILURE); // Exit the program if trying to pop from an empty stack
    }
}

// Function to evaluate the postfix expression
int evaluateExpression(const char* expression) {
    Stack* stack = createStack(); // Create a stack for evaluating the expression

    // Loop through each character in the expression until the null terminator
    for (int i = 0; expression[i] != '\0'; i++) {
        char ch = expression[i]; // Get the current character
        if (isdigit(ch)) { // Check if the character is a digit
            push(stack, ch - '0'); // Convert character to integer and push onto stack
        } else {
 // If the character is an operator
            int operand2 = pop(stack); // Pop the top two operands from the stack
            int operand1 = pop(stack);
            switch (ch) { // Perform the operation based on the operator
                case '+': push(stack, operand1 + operand2); break; // Addition
                case '-': push(stack, operand1 - operand2); break; // Subtraction
                case '*': push(stack, operand1 * operand2); break; // Multiplication
                case '/': // Division
                    if (operand2 != 0) { // Check for division by zero
                        push(stack, operand1 / operand2); // Push the result onto the stack
                    } else {
                        printf("Division by zero error\n"); // Print error if division by zero
                        exit(EXIT_FAILURE); // Exit the program
                }
                break;
            }
        }
    }

    int result = pop(stack); // Pop the final result from the stack
    free(stack); // Free the allocated memory for the stack
    return result; // Return the final result of the postfix expression
}

int main() {
    const char* expression = "36+9*"; // Example postfix expression
    printf("Result: %d\n", evaluateExpression(expression)); // Evaluate and print the result
    return 0; // Return success
}
```

Output:-



```
Result: 81
```