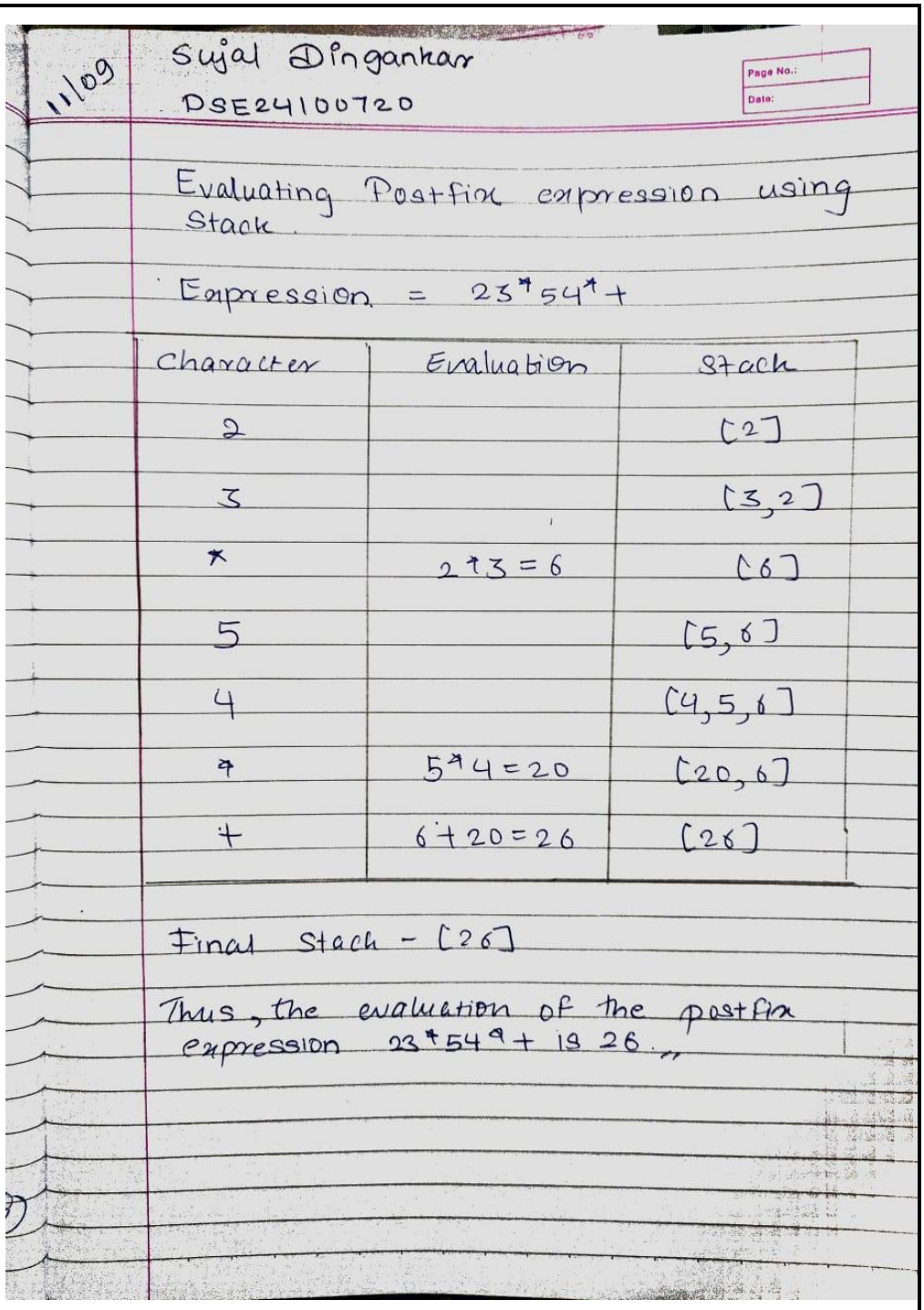
Name	Sujal Dingankar
UID no.	DSE24100720
Experiment No.	1

AIM:	Evaluation of a PostFix Expression
THEORY:	A **stack** is a linear data structure that follows the **LIFO (Last In, First Out)** principle. This means that the last element inserted into the stack will be the first one to be removed. A stack has a single access point, which is referred to as the **top** of the stack. All operations (insertion, deletion, or viewing the top element) are performed on the top. **Operations on Stack:** 1. **Push Operation**: Adds an element to the top of the stack. If the stack is full, no more elements can be added, leading to an overflow condition. 2. **Pop Operation:** Removes and returns the top element from the stack. If the stack is empty, this operation results in an underflow condition. 3. **Peek Operation:** Retrieves the top element of the stack without removing it. If the stack is empty, the operation indicates this. 4. **IsEmpty Operation:** Checks if the stack is empty, returning true if there are no elements in the stack, and false otherwise. 5. **IsFull Operation:** Determines whether the stack has reached its maximum capacity, returning true if the stack is full and false if it is not. **Applications on Stack:** 1. **Expression Evaluation:** Used to evaluate postfix and prefix expressions. 2. **Recursion Management:** Manages recursive function calls in programming. 3. **Backtracking:** Implements backtracking algorithms like maze solving. 4. **Function Call Management:** Handles function calls and returns in programming. 5. **Undo Mechanism:** Implements undo/redo functionality in software.

ALGORITHM:	1. **Initialize the stack**:
	  - Ask the user for the postfix expression.
	  - Determine the length of the postfix expression and initialize a stack of the same size.
	2. **Traverse each character of the postfix expression**:
	  - Loop through the string containing the postfix expression, processing one character at a time.
	3. **Check if the character is an operand**:
	  - If the character is a digit (between '0' and '9'), convert it to an integer by subtracting '0'.
	  - Push the integer onto the stack.
	4. **Check if the character is an operator**:
	  - If the character is an operator (+, -, *, /, ^), pop the top two operands from the stack.
	  - The first popped value is the second operand, and the second popped value is the first operand.
	5. **Perform the operation**:
	  - Depending on the operator:
	  ▪ For +, add the two operands.
	  ▪ For -, subtract the second operand from the first.
	  ▪ For *, multiply the two operands.
	  ▪ For /, divide the first operand by the second.
	  ▪ For ^, calculate the first operand raised to the power of the second.
	6. **Push the result**:
	  - Push the result of the operation back onto the stack.
	7. **Continue processing**:
	  - Continue processing the next character in the expression until all characters are processed.
	8. **Return the final result**:
	  - Once the entire postfix expression has been processed, the result of the expression will be at the top of the stack.
	  - Return and print this value as the final evaluated result.

PROBLEM SOLVING:



11/09

Sujal Dingankar

DSE24100720

Page No.:

Date:

Evaluating Postfix expression using Stack.

Expression = 23*54*+

Character	Evaluation	Stack
2		[2]
3		[3,2]
*	2*3=6	[6]
5		[5,6]
4		[4,5,6]
*	5*4=20	[20,6]
+	6+20=26	[26]

Final Stack - [26]

Thus, the evaluation of the postfix expression 23*54*+ is 26.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
struct stack{
    int *arr;
    int size;
    int top;
} st;

void init(int size){
    st.size = size;
    st.arr = (int*)malloc(size * sizeof(int *));
}

void push(int a){
    if(st.top == st.size - 1){
          printf("Stack is full");
    }else{
          st.arr[++st.top] = a;
          }
}

int pop(){
    if(st.top == -1){
          printf("Stack is empty");
          return 0;
    }else{
          st.top--;
    }
}

int peak(){
    if(st.top == -1){
          printf("Stack is empty");
          return 0;
    }else{
          return st.arr[st.top];
    }
}

int power(int a, int b){
    int i = 1;
```

```
    int ans = 1;
    if( b == 0){
          return 1;
    }else{
    while(i<=b){
          ans = a * ans;
          i++;
          }
    }
    return ans;
}

int evaluate(char* s){
    for(int i = 0; i < strlen(s); i++){
          if( s[i]-'0' >= 0 && s[i]-'0' <= 9){
                int c = s[i] - '0';
                push(c);

          }else{
                int n1 = peak();
                pop();
                int n2 = peak();
                pop();
                int ans;
                if(s[i] == '+'){
                      ans =  n2+n1;
                }else if(s[i] == '-'){
                      ans =  n2 - n1;
                }else if(s[i] == '*'){
                      ans =  n1 * n2;
                }else if(s[i] == '/'){
                      ans =  n2 / n1;
                }else if(s[i] == '^'){
                      ans = power(n2,n1);
                }
                push(ans);

          }

    }
    return st.arr[st.top];
}
```

```
int main(){

    char s[100];
    printf("Enter postfix expression:\n");
    scanf("%s", s);
    int n = strlen(s);
    init(n);
    printf("The evaluated postfix expression is: %d \n",
evaluate(s));
    return 0;
}
```

RESULT:

```
Enter postfix expression:
23*54*+
The evaluated postfix expression is: 26


...Program finished with exit code 0
Press ENTER to exit console.
```

CONCLUSION: From this experiment with stack-based postfix expression evaluation, I gained practical insight into how stacks operate on a Last In, First Out (LIFO) principle. I learned to effectively use stack operations—push, pop, and peek—to manage and process operands and operators, which is crucial for evaluating expressions. I will be able to solve problems using Stack Application.