Assignment 5

Aim - To implement Stack ADT using an array and evaluate a postfix expression.

Theory
A stack is used which follows LIFO principle. (Last In First Out) meaning the last element inserted is the first one to be removed.

The basic operations are push (insert an element to the top), pop (remove an element from the top).

In post fix the operators are written after their operands like 2+3 is 23+ in post fix form. Postfix eliminates the need for paranthesses and operator presedence rules, making evaluation simpler.

To evaluate a postfix expression, first create an empty stack and intialise top to zero.

Scan the expression from left to right, processing each character at a time. If the character is a digit, it should be added to the stack. If it is an operator, +, -, x, / then the top 2 elements must be removed and the operator applied on them. Then push the answer to the

stack.
This process should be repeated until the entire expression is evaluated.
The final element in the stack is the final result.
If the number of operands and operators does not satisfy the condition then the expression is invalid.

Time complexity of this program is O(n) where n is the number of characters in the string given as all push pop and operations have time complexity O(1).

Conclusion
This program evaluates postfix using stack ADT. It clearly demonstrates the practical application of stack operations and the LIFO principle in solving arithmetic expressions efficiently.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
# define SIZE 25
struct Stack {
   int top;
   int arr[SIZE];
};
void push(struct Stack *s, int ss);
int pop(struct Stack *s);
void main()
{
   struct Stack s;
   s.top = 0;
   char aa[SIZE];
   printf("Enter postfix: ");
   scanf("%s", aa);
   int length = strlen(aa);
   int dc = 0;
   for(int i = 0; i<length; i++){
      if (isdigit(aa[i])){
      dc++;
   }
   }
   if (length-dc != dc-1){
      printf("ERROR");
      return;
   }
```

```
    for(int i = 0; i<length;i++){
        if (isdigit(aa[i])){
            push(&s, aa[i] - '0');
        } else {
            if (s.top < 1){
                printf("ERROR");
                return;
            }
            int t1 = pop(&s);
            int t2 = pop(&s);
            if (aa[i] == '+'){
                push(&s, (t1+t2));
            } else if (aa[i] == '-') {
                push(&s, (t2-t1));
            } else if (aa[i] == '*') {
                push(&s, (t2*t1));
            } else if (aa[i] == '/') {
                push(&s, (t2/t1));
            }
        }
    }
    printf("Result = %d", pop(&s));
}
void push(struct Stack *s, int ss)
{
    // Checks if the stack is full
    if (s->top < SIZE)
    {
```

```
      s->arr[s->top] = ss;
      s->top++;
   }
   else
   {
      printf("STACK IS FULL\n");
   }
}
int pop(struct Stack *s)
{
   if (s->top > 0)
   {
      s->top--;
      return s->arr[s->top];
   }
   else
   {
      printf("NO ELEMENTS IN STACK\n");
      return -1;
   }
}
```