**PRACTICAL 06**

Q. Stack implementation :infix to postfix ,infix to prefix ,evaluation of postfix expression .

Program:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <math.h>

#define MAX 100

// Stack structure

typedef struct {

    char items[MAX];

    int top;

} Stack;

// Function prototypes

void initStack(Stack\* s);

void push(Stack\* s, char item);

char pop(Stack\* s);

char peek(Stack\* s);

int isFull(Stack\* s);

int isEmpty(Stack\* s);

int precedence(char c);

int isOperator(char c);

void infixToPostfix(char\* infix, char\* postfix);

void infixToPrefix(char\* infix, char\* prefix);

int evaluatePostfix(char\* postfix);

void reverse(char\* exp);

int main() {

    Stack s;

    initStack(&s);

    char infix[MAX], postfix[MAX], prefix[MAX];

    int choice;

    do {

        printf("\nMenu:\n");

        printf("1. Infix to Postfix\n");

        printf("2. Infix to Prefix\n");

        printf("3. Evaluate Postfix Expression\n");

        printf("4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter an infix expression: ");

                scanf("%s", infix);

                infixToPostfix(infix, postfix);

                printf("Postfix Expression: %s\n", postfix);

                break;

            case 2:

                printf("Enter an infix expression: ");

                scanf("%s", infix);

                infixToPrefix(infix, prefix);

                printf("Prefix Expression: %s\n", prefix);

                break;

            case 3:

                printf("Enter a postfix expression: ");

                scanf("%s", postfix);

                int result = evaluatePostfix(postfix);

                printf("Result: %d\n", result);

                break;

            case 4:

                printf("Exiting...\n");

                break;

            default:

                printf("Invalid choice! Please try again.\n");

        }

    } while (choice != 4);

    return 0;

}

// Stack functions

void initStack(Stack\* s) {

    s->top = -1;

}

void push(Stack\* s, char item) {

    if (!isFull(s)) {

        s->items[++s->top] = item;

    }

}

char pop(Stack\* s) {

    if (!isEmpty(s)) {

        return s->items[s->top--];

    }

    return '\0'; // Return null character if stack is empty

}

char peek(Stack\* s) {

    if (!isEmpty(s)) {

        return s->items[s->top];

    }

    return '\0'; // Return null character if stack is empty

}

int isFull(Stack\* s) {

    return s->top == MAX - 1;

}

int isEmpty(Stack\* s) {

    return s->top == -1;

}

int precedence(char c) {

    switch (c) {

        case '+':

        case '-':

            return 1;

        case '\*':

        case '/':

            return 2;

        case '^':

            return 3;

        default:

            return 0;

    }

}

int isOperator(char c) {

    return c == '+' || c == '-' || c == '\*' || c == '/' || c == '^';

}

void infixToPostfix(char\* infix, char\* postfix) {

    Stack s;

    initStack(&s);

    int j = 0;

    for (int i = 0; infix[i]; i++) {

        if (isalnum(infix[i])) {

            postfix[j++] = infix[i];

        } else if (infix[i] == '(') {

            push(&s, infix[i]);

        } else if (infix[i] == ')') {

            while (!isEmpty(&s) && peek(&s) != '(') {

                postfix[j++] = pop(&s);

            }

            pop(&s); // Remove '('

        } else if (isOperator(infix[i])) {

            while (!isEmpty(&s) && precedence(peek(&s)) >= precedence(infix[i])) {

                postfix[j++] = pop(&s);

            }

            push(&s, infix[i]);

        }

    }

    while (!isEmpty(&s)) {

        postfix[j++] = pop(&s);

    }

    postfix[j] = '\0';

}

void reverse(char\* exp) {

    int n = strlen(exp);

    for (int i = 0; i < n / 2; i++) {

        char temp = exp[i];

        exp[i] = exp[n - i - 1];

        exp[n - i - 1] = temp;

    }

}

void infixToPrefix(char\* infix, char\* prefix) {

    reverse(infix);

    Stack s;

    initStack(&s);

    int j = 0;

    for (int i = 0; infix[i]; i++) {

        if (isalnum(infix[i])) {

            prefix[j++] = infix[i];

        } else if (infix[i] == ')') {

            push(&s, infix[i]);

        } else if (infix[i] == '(') {

            while (!isEmpty(&s) && peek(&s) != ')') {

                prefix[j++] = pop(&s);

            }

            pop(&s); // Remove ')'

        } else if (isOperator(infix[i])) {

            while (!isEmpty(&s) && precedence(peek(&s)) > precedence(infix[i])) {

                prefix[j++] = pop(&s);

            }

            push(&s, infix[i]);

        }

    }

    while (!isEmpty(&s)) {

        prefix[j++] = pop(&s);

    }

    prefix[j] = '\0';

    reverse(prefix);

}

int evaluatePostfix(char\* postfix) {

    Stack s;

    initStack(&s);

    int i;

    for (i = 0; postfix[i]; i++) {

        if (isdigit(postfix[i])) {

            push(&s, postfix[i] - '0');  // Convert char to int

        } else {

            int op2 = pop(&s);

            int op1 = pop(&s);

            switch (postfix[i]) {

                case '+':

                    push(&s, op1 + op2);

                    break;

                case '-':

                    push(&s, op1 - op2);

                    break;

                case '\*':

                    push(&s, op1 \* op2);

                    break;

                case '/':

                    push(&s, op1 / op2);

                    break;

                case '^':

                    push(&s, (int)pow(op1, op2));

                    break;

            }

        }

    }

    return pop(&s);

}

Output: