single C program that implements the Stack ADT, converts an infix expression to postfix, and evaluates the postfix expression using the stack.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>

#define MAX 100 // Maximum size of the stack

// Stack structure definition
typedef struct Stack {
    int arr[MAX];
    int top;
} Stack;

// Function to initialize the stack
void initStack(Stack *s) {
    s->top = -1;
}

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

// Function to check if the stack is full
int isFull(Stack *s) {
    return s->top == MAX - 1;
}

// Function to push an element onto the stack
void push(Stack *s, int value) {
    if (isFull(s)) {
        printf("Stack Overflow!\n");
        return;
    }
    s->arr[++(s->top)] = value;
}

// Function to pop an element from the stack
int pop(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack Underflow!\n");
        return -1;
    }
    return s->arr[(s->top)--];
}

// Function to peek the top element of the stack
int peek(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack is empty!\n");
```

```
        return -1;
    }
    return s->arr[s->top];
}

// Function to get the precedence of operators
int precedence(char c) {
    if (c == '+' || c == '-') return 1;
    if (c == '*' || c == '/') return 2;
    if (c == '^') return 3;
    return 0;
}

// Function to perform arithmetic operations
int applyOperator(int a, int b, char operator) {
    switch (operator) {
        case '+': return a + b;
        case '-': return a - b;
        case '*': return a * b;
        case '/': return a / b;
        case '^': return (int)pow(a, b);
        default: return 0;
    }
}

// Function to convert infix expression to postfix expression
void infixToPostfix(char *infix, char *postfix) {
    Stack s;
    initStack(&s);
    int k = 0;  // Index for postfix array

    for (int i = 0; infix[i] != '\0'; i++) {
        char c = infix[i];

        // If the character is an operand, add it to the postfix expression
        if (isalnum(c)) {
            postfix[k++] = c;
        }
        // If the character is '(', push it onto the stack
        else if (c == '(') {
            push(&s, c);
        }
        // If the character is ')', pop until '(' is encountered
        else if (c == ')') {
            while (!isEmpty(&s) && peek(&s) != '(') {
                postfix[k++] = pop(&s);
            }
            pop(&s); // Pop the '('
        }
        // If the character is an operator, pop operators with higher or equal
precedence
        else if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^') {
            while (!isEmpty(&s) && precedence(peek(&s)) >= precedence(c)) {
```

```
                postfix[k++] = pop(&s);
            }
            push(&s, c);
        }
    }

    // Pop any remaining operators from the stack
    while (!isEmpty(&s)) {
        postfix[k++] = pop(&s);
    }

    postfix[k] = '\0'; // Null-terminate the postfix expression
}

// Function to evaluate the postfix expression
int evaluatePostfix(char *postfix) {
    Stack s;
    initStack(&s);

    for (int i = 0; postfix[i] != '\0'; i++) {
        char c = postfix[i];

        // If the character is an operand, push it onto the stack
        if (isdigit(c)) {
            push(&s, c - '0');  // Convert char to integer and push
        }
        // If the character is an operator, pop two operands and apply the operator
        else if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^') {
            int b = pop(&s);
            int a = pop(&s);
            int result = applyOperator(a, b, c);
            push(&s, result);
        }
    }

    // The final result will be the only element remaining in the stack
    return pop(&s);
}

int main() {
    char infix[MAX], postfix[MAX];

    // Read the infix expression from the user
    printf("Enter infix expression: ");
    fgets(infix, sizeof(infix), stdin);

    // Convert the infix expression to postfix
    infixToPostfix(infix, postfix);

    // Print the postfix expression
    printf("Postfix expression: %s\n", postfix);

    // Evaluate the postfix expression
```

```
    int result = evaluatePostfix(postfix);

    // Print the result of the evaluation
    printf("Result: %d\n", result);

    return 0;
}
```

Example:
Enter infix expression: 3+(2*5)-(9/3)

Output:
Postfix expression: 325*+93/-
Result: 10