

```

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>


#define MAX 100


// Stack for characters (operators)
char stack[MAX];
int top = -1;


// Stack for integers (evaluation)
int evalStack[MAX];
int evalTop = -1;


// Function prototypes
void push(char c);
char pop();
char peek();
int precedence(char op);
int isOperator(char c);
void infixToPostfix(char* infix, char* postfix);
int evaluatePostfix(char* postfix);


// Character stack operations
void push(char c) {
    if (top < MAX - 1)
        stack[++top] = c;
}

char pop() {

```

```
    if (top >= 0)
        return stack[top--];
    return '\0';
}
```

```
char peek() {
    if (top >= 0)
        return stack[top];
    return '\0';
}
```

// Integer stack operations for evaluation

```
void pushEval(int val) {
    if (evalTop < MAX - 1)
        evalStack[++evalTop] = val;
}
```

```
int popEval() {
    if (evalTop >= 0)
        return evalStack[evalTop--];
    return 0;
}
```

// Get precedence of operators

```
int precedence(char op) {
    if (op == '^') return 3;
    if (op == '*' || op == '/') return 2;
    if (op == '+' || op == '-') return 1;
    return 0;
}
```

```
// Check if a character is an operator

int isOperator(char c) {

    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');

}
```

```
// Convert infix expression to postfix

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

    int i = 0, j = 0;

    char token, x;

    while ((token = infix[i++]) != '\0') {

        if (isspace(token))

            continue;

        if (isalnum(token)) {

            postfix[j++] = token;

        } else if (token == '(') {

            push(token);

        } else if (token == ')') {

            while ((x = pop()) != '(')

                postfix[j++] = x;

            postfix[j++] = x;

        } else if (isOperator(token)) {

            while (precedence(peek()) >= precedence(token))

                postfix[j++] = pop();

            push(token);

        }

    }

}
```

```
while (top != -1)

    postfix[j++] = pop();
```

```
postfix[j] = '\0';
```

```
}
```

```
// Evaluate postfix expression
```

```
int evaluatePostfix(char* postfix) {
```

```
    int i = 0;
```

```
    char token;
```

```
    int op1, op2;
```

```
    while ((token = postfix[i++]) != '\0') {
```

```
        if (isdigit(token)) {
```

```
            pushEval(token - '0'); // Convert char to int
```

```
        } else if (isOperator(token)) {
```

```
            op2 = popEval();
```

```
            op1 = popEval();
```

```
            switch (token) {
```

```
                case '+': pushEval(op1 + op2); break;
```

```
                case '-': pushEval(op1 - op2); break;
```

```
                case '*': pushEval(op1 * op2); break;
```

```
                case '/': pushEval(op1 / op2); break;
```

```
                case '^': {
```

```
                    int result = 1;
```

```
                    for (int j = 0; j < op2; j++) result *= op1;
```

```
                    pushEval(result);
```

```
                    break;
```

```
                }
```

```
            }
```

```
        }
```

```
    }
```

```
    return popEval();
```

```
}
```

```
// Main function

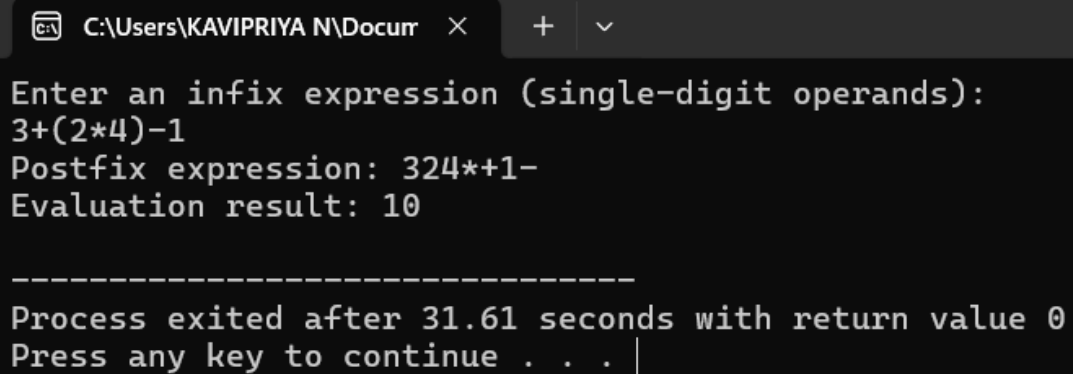
int main() {
    char infix[MAX], postfix[MAX];

    printf("Enter an infix expression (single-digit operands):\n");
    fgets(infix, MAX, stdin);

    infixToPostfix(infix, postfix);
    printf("Postfix expression: %s\n", postfix);

    int result = evaluatePostfix(postfix);
    printf("Evaluation result: %d\n", result);

    return 0;
}
```



```
C:\Users\KAVIPRIYA N\Docum
Enter an infix expression (single-digit operands):
3+(2*4)-1
Postfix expression: 324*+1-
Evaluation result: 10

-----
Process exited after 31.61 seconds with return value 0
Press any key to continue . . . |
```