```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <math.h>
#define MAX 100
char stack[MAX];
int top = -1;
void push(char x) {
    stack[++top] = x;
}
char pop() {
    return stack[top--];
}
int priority(char x) {
    if (x == '(') return 0;
    if (x == '+' || x == '-') return 1;
    if (x == '*' || x == '/') return 2;
    if (x == '^') return 3;
    return 0;
}
void infixToPostfix(char infix[], char postfix[]) {
    char x, token;
    int i = 0, k = 0;
    while ((token = infix[i++]) != '\0') {
        if (isalnum(token)) {
            postfix[k++] = token;
        }
```

```
        else if (token == '(') {
            push(token);
        }
        else if (token == ')') {
            while ((x = pop()) != '('
                postfix[k++] = x;
        }
        else {
            while (top != -1 && priority(stack[top]) >= priority(token))
                postfix[k++] = pop();
            push(token);
        }
    }
    while (top != -1)
        postfix[k++] = pop();
    postfix[k] = '\0';
}
int evalPostfix(char postfix[]) {
    int stack[MAX];
    int top = -1;
    int i, op1, op2, res;
    char token;
    for (i = 0; postfix[i] != '\0'; i++) {
        token = postfix[i];
        if (isdigit(token))
            stack[++top] = token - '0';
        else {
```

```
            op2 = stack[top--];
            op1 = stack[top--];
            switch (token) {
                case '+': res = op1 + op2; break;
                case '-': res = op1 - op2; break;
                case '*': res = op1 * op2; break;
                case '/': res = op1 / op2; break;
                case '^': res = pow(op1, op2); break;
            }
            stack[++top] = res;
        }
    }
    return stack[top];
}
int main() {
    char infix[MAX], postfix[MAX];
    printf("Enter an infix expression: ");
    scanf("%s", infix);
    infixToPostfix(infix, postfix);
    printf("Postfix Expression: %s\n", postfix);
    printf("Evaluation Result: %d\n", evalPostfix(postfix));
    return 0;
}
```

Output:


Output
Enter an infix expression: (3+4)*5
Postfix Expression: 34+5*
Evaluation Result: 35
=== Code Execution Successful ===