```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define SIZE 100

char stack[SIZE];
int top = -1;

// Function to push to stack
void push(char ch) {
    stack[++top] = ch;
}

// Function to pop from stack
char pop() {
    return stack[top--];
}

// Function to return precedence of operators
int precedence(char op) {
    switch(op) {
        case '^': return 3;
        case '*':
        case '/': return 2;
        case '+':
        case '-': return 1;
        default : return 0;
    }
}

// Function to check if character is operator
int isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^';
}

// Function to convert infix to postfix
void infixToPostfix(char infix[], char postfix[]) {
    int i, k = 0;
    char ch;

    for(i = 0; i < strlen(infix); i++) {
        ch = infix[i];

        if(isalnum(ch)) {
            postfix[k++] = ch;
        }
        else if(ch == '(') {
```

```
            push(ch);
        }
        else if(ch == ')') {
            while(top != -1 && stack[top] != '(') {
                postfix[k++] = pop();
            }
            pop(); // Remove '('
        }
        else if(isOperator(ch)) {
            while(top != -1 && precedence(stack[top]) >= precedence(ch)) {
                postfix[k++] = pop();
            }
            push(ch);
        }
    }

    // Pop remaining operators
    while(top != -1) {
        postfix[k++] = pop();
    }

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

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

    printf("Enter infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

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

    return 0;
}
```