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

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

// Function to return associativity of operators
char associativity(char c) {
    if (c == '^')
        return 'R';
    return 'L'; // Default to left-associative
}

// The main function to convert infix expression to postfix expression
void infixToPostfix(char s[]) {
    char result[1000];
    int resultIndex = 0;
    int len = strlen(s);
    char stack[1000];
    int stackIndex = -1;

    for (int i = 0; i < len; i++) {
        char c = s[i];

        // If the scanned character is an operand, add it to the output
string.
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0'
&& c <= '9')) {
            result[resultIndex++] = c;
        }
        // If the scanned character is an â€˜(â€˜, push it to the stack.
        else if (c == '(') {
            stack[++stackIndex] = c;
        }
        // If the scanned character is an â€˜)â€™, pop and add to the
output string from the stack
        // until an â€˜(â€˜ is encountered.
        else if (c == ')') {
            while (stackIndex >= 0 && stack[stackIndex] != '(') {
                result[resultIndex++] = stack[stackIndex--];
            }
            stackIndex--; // Pop '('
        }
        // If an operator is scanned
```

```
        else {
            while (stackIndex >= 0 && (prec(s[i]) <
prec(stack[stackIndex]) ||
                                       prec(s[i]) ==
prec(stack[stackIndex]) &&
                                           associativity(s[i]) == 'L')) {
                result[resultIndex++] = stack[stackIndex--];
            }
            stack[++stackIndex] = c;
        }
    }

    // Pop all the remaining elements from the stack
    while (stackIndex >= 0) {
        result[resultIndex++] = stack[stackIndex--];
    }

    result[resultIndex] = '\0';
    printf("%s\n", result);
}

// Driver code
int main() {
    char exp[] = "a+b*(c^d-e)^(f+g*h)-i";

    // Function call
    infixToPostfix(exp);

    return 0;
}
```