Experiments\InfixToPostfix.c

```c
#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 infix[]) {
    char postfix[25];
    int postfixIndex = 0;
    int len = strlen(infix);
    char stack[25];
    int stackIndex = -1;

    for (int i = 0; i < len; i++) {
        char c = infix[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')) {
            postfix[postfixIndex++] = c;
        }
        // If the scanned character is an ‘(‘, push it to the stack.
        else if (c == '(') {
            stack[++stackIndex] = c;
        }
        // If the scanned character is an ‘)’, pop and add to the output string from the stack
        // until an ‘(‘ is encountered.
        else if (c == ')') {
            while (stackIndex >= 0 && stack[stackIndex] != '(') {
                postfix[postfixIndex++] = stack[stackIndex--];
            }
            stackIndex--; // Pop '('
        }
        // If an operator is scanned
        else {
            while (stackIndex >= 0 && (prec(infix[i]) < prec(stack[stackIndex]) ||
                               prec(infix[i]) == prec(stack[stackIndex]) &&
                               associativity(infix[i]) == 'L')) {

                postfix[postfixIndex++] = stack[stackIndex--];
            }
            stack[++stackIndex] = c;
        }
    }

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

    postfix[postfixIndex] = '\0';
    printf("Postfix expression: %s\n", postfix);
}

// Driver code
int main() {
    char exp[25];
    //char exp[] = "a+b*(c^d-e)^(f+g*h)-i";

    printf("Enter an expression: ");
    fgets(exp, sizeof(exp), stdin);

    // Function call
    infixToPostfix(exp);

    return 0;
}





/*
Output:

PS C:\Users\gagan\Desktop\Data_Structure_And_Algorithm\Experiments> cd
"c:\Users\gagan\Desktop\Data_Structure_And_Algorithm\Experiments\" ; if ($?) { gcc
InfixToPostfix.c -o InfixToPostfix } ; if ($?) { .\InfixToPostfix }
Enter an expression: a+b*(c^d-e)^(f+g*h)-i
Postfix expression: abcd^e-fgh*+^*+i-

```

```
*/
```