Rajalakshmi Engineering College

Name: KISHORE RAJ.A.S.
Email: 241801127@rajalakshmi.edu.in
Roll no: 241801127
Phone: 7397295789
Branch: REC
Department: I AI & DS FB
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

You are a software developer tasked with building a module for a scientific calculator application. The primary function of this module is to convert infix mathematical expressions, which are easier for users to read and write, into postfix notation (also known as Reverse Polish Notation). Postfix notation is more straightforward for the application to evaluate because it removes the need for parentheses and operator precedence rules.

The scientific calculator needs to handle various mathematical expressions with different operators and ensure the conversion is correct. Your task is to implement this infix-to-postfix conversion algorithm using a stack-based approach.

Example

Input:

a+b

Output:

ab+

Explanation:

The postfix representation of (a+b) is ab+.

Input Format

The input is a string, representing the infix expression.

Output Format

The output displays the postfix representation of the given infix expression.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: a+(b*e)

Output: abe*+

Answer

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

struct Stack {
    int top;
    unsigned capacity;
    char* array;
};

struct Stack* createStack(unsigned capacity) {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));

    if (!stack)
```

```
        return NULL;

    stack->top = -1;
    stack->capacity = capacity;
    stack->array = (char*)malloc(stack->capacity * sizeof(char));

    return stack;
}

int isEmpty(struct Stack* stack) {
    return stack->top == -1;
}

char peek(struct Stack* stack) {
    return stack->array[stack->top];
}

char pop(struct Stack* stack) {
    if (!isEmpty(stack))
        return stack->array[stack->top--];
    return '$';
}

void push(struct Stack* stack, char op) {
    stack->array[++stack->top] = op;
}

// You are using Gcc
int precedence(char op) {
    if (op == '^') return 3;
    else if (op == '*' || op == '/') return 2;
    else if (op == '+' || op == '-') return 1;
    return 0;
}

// Function to check if the character is an operand (variable or number)
int isOperand(char ch) {
    return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');
}

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

```
}

// Function to convert infix expression to postfix
void infixToPostfix(char exp[]) {
    struct Stack* stack = createStack(10); // Initial capacity
    int k = 0;
    char ch;
    char* postfix = (char*)malloc(sizeof(char) * (strlen(exp) + 1)); // Allocate
memory for postfix expression

    // Iterate through each character of the infix expression
    for (int i = 0; exp[i] != '\0'; i++) {
        ch = exp[i];

        // If the character is an operand, add it to the postfix expression
        if (isOperand(ch)) {
            postfix[k++] = ch;
        }
        // If the character is '(', push it onto the stack
        else if (ch == '(') {
            push(stack, ch);
        }
        // If the character is ')', pop and add to postfix until '(' is found
        else if (ch == ')') {
            while (!isEmpty(stack) && peek(stack) != '(') {
                postfix[k++] = pop(stack);
            }
            pop(stack);  // Pop the '(' from the stack
        }
        // If the character is an operator
        else if (isOperator(ch)) {
            while (!isEmpty(stack) && precedence(peek(stack)) >= precedence(ch)) {
                if (ch == '^' && peek(stack) == '^') {
                    break;  // Right associativity for '^'
                }
                postfix[k++] = pop(stack);
            }
            push(stack, ch);  // Push the current operator onto the stack
        }
    }

    // Pop all the remaining operators from the stack
```

```
    while (!isEmpty(stack)) {
        postfix[k++] = pop(stack);
    }

    postfix[k] = '\0';  // Null-terminate the postfix string

    // Output the resulting postfix expression
    printf("%s\n", postfix);

    // Free allocated memory
    free(stack->array);  // Renamed 'arr' to 'array'
    free(stack);
    free(postfix);
}
int main() {
    char exp[100];
    scanf("%s", exp);

    infixToPostfix(exp);
    return 0;
}
```

Status : Correct **Marks : 10/10**