Rajalakshmi Engineering College

Name: keerthana v
Email: 241501082@rajalakshmi.edu.in
Roll no: 241501082
Phone: 7358113357
Branch: REC
Department: I AI & ML FA
Batch: 2028
Degree: B.E - AI & ML

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;
}
int isOperand(char ch)
{


    return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');

}

int Prec(char ch)

{


    switch (ch)
```

```
{

        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;


}
    return -1; // Invalid operator
}

void infixToPostfix(char* exp)

{


    struct Stack* stack = createStack(strlen(exp));
    if (!stack) return;

    char result[100]; // To store the postfix expression
    int k = 0; // Index for result

    for (int i = 0; exp[i]; i++)

{


        // If the scanned character is an operand, add it to output
        if (isOperand(exp[i]))

{


            result[k++] = exp[i];
```

```
}
    // If the scanned character is '(', push it to the stack
    else if (exp[i] == '(')

{


    push(stack, exp[i]);


}
    // If the scanned character is ')', pop and output from the stack
    // until an '(' is encountered
    else if (exp[i] == ')')

{


    while (!isEmpty(stack) && peek(stack) != '(')

{


        result[k++] = pop(stack);


}
    pop(stack); // Remove '(' from stack


}
    // An operator is encountered
    else

{


    while (!isEmpty(stack) && Prec(peek(stack)) >= Prec(exp[i]))

{
```

```
            result[k++] = pop(stack);

        }
            push(stack, exp[i]);

    }

    }

    // Pop all the operators from the stack
    while (!isEmpty(stack))
    {

        result[k++] = pop(stack);

    }

    result[k] = '\0'; // Null-terminate the result string
    printf("%s\n", result); // Print the postfix expression
}
int main() {
    char exp[100];
    scanf("%s", exp);

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

Status : Correct *Marks : 10/10*