Rajalakshmi Engineering College

Name: Jayasree D
Email: 241801100@rajalakshmi.edu.in
Roll no:
Phone: 9025821157
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;
}
def precedence(op):
    """Return precedence of operators."""
    if op in {'+', '-'}:
        return 1
    if op in {'*', '/'}:
        return 2
    if op == '^':
        return 3
    return 0

def infix_to_postfix(expression):
    """Convert infix expression to postfix notation."""
    stack = []
    output = ''

    for char in expression:
        if char.isalnum():  # If operand, add to output
            output += char
        elif char == '(':  # If '(', push to stack
            stack.append(char)
        elif char == ')':  # If ')', pop from stack to output until '(' is found
            while stack and stack[-1] != '(':
                output += stack.pop()
            stack.pop()  # Remove '(' from stack
        else:  # Operator case
            while stack and precedence(stack[-1]) >= precedence(char):
                output += stack.pop()
            stack.append(char)

    # Pop remaining operators from stack to output
    while stack:
        output += stack.pop()

    return output

# Taking input for a single test case
expression = input().strip()  # Ensuring we read only one input at a time
print(infix_to_postfix(expression))

int main() {
    char exp[100];
    scanf("%s", exp);

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

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