Rajalakshmi Engineering College

Name: mahati sathish
Email: 240701302@rajalakshmi.edu.in
Roll no: 240701302
Phone: 9176485888
Branch: REC
Department: I CSE FC
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_CY

Attempt : 1
Total Mark : 30
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Suppose you are building a calculator application that allows users to enter mathematical expressions in infix notation. One of the key features of your calculator is the ability to convert the entered expression to postfix notation using a Stack data structure.

Write a function to convert infix notation to postfix notation using a Stack.

Input Format

The input consists of a string, an infix expression that includes only digits(0-9), and operators(+, -, *, /).

Output Format

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

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1+2*3/4-5
Output: 123*4/+5-

Answer

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

struct Stack

{


    char *arr;
    int top;
    int capacity;

};

struct Stack* createStack(int capacity)

{


    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->arr = (char*)malloc(stack->capacity * sizeof(char));
    return stack;

}

int isEmpty(struct Stack* stack)
```

```
{
    return stack->top == -1;
}
void push(struct Stack* stack, char c)
{

    stack->arr[++stack->top] = c;
}
char pop(struct Stack* stack)
{

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

    return stack->arr[stack->top];
}
int precedence(char c)
{

    if (c == '+' || c == '-') return 1;
    if (c == '*' || c == '/') return 2;
    return 0;
```

```
}
void infixToPostfix(char* expression)
{
    struct Stack* stack = createStack(30);
    int i = 0;
    while (expression[i] != '\0')
{
        if (isdigit(expression[i]))
{
            printf("%c", expression[i]);
} else if (expression[i] == '(')
{
            push(stack, expression[i]);
} else if (expression[i] == ')')
{
            while (!isEmpty(stack) && peek(stack) != '(')
{
                printf("%c", pop(stack));
```

```
}
        pop(stack);

} else

{

        while (!isEmpty(stack) && precedence(peek(stack)) >=
precedence(expression[i]))

{

            printf("%c", pop(stack));

}
        push(stack, expression[i]);

}
    i++;

}
  while (!isEmpty(stack))

{

    printf("%c", pop(stack));

}

}

int main()
```

```
{

    char expression[31];
    scanf("%s", expression);
    infixToPostfix(expression);
    return 0;

}
```

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

2. Problem Statement

You are required to implement a stack data structure using a singly linked list that follows the Last In, First Out (LIFO) principle.

The stack should support the following operations: push, pop, display, and peek.

Input Format

The input consists of four space-separated integers N, representing the elements to be pushed onto the stack.

Output Format

The first line of output displays all four elements in a single line separated by a space.

The second line of output is left blank to indicate the pop operation without displaying anything.

The third line of output displays the space separated stack elements in the same line after the pop operation.

The fourth line of output displays the top element of the stack using the peek operation.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 11 22 33 44
Output: 44 33 22 11

33 22 11
33

Answer

-

Status : Skipped ***Marks : 0/10***

3. Problem Statement

In an educational setting, Professor Smith tasks Computer Science students with designing an algorithm to evaluate postfix expressions efficiently, fostering problem-solving skills and understanding of stack-based computations.

The program prompts users to input a postfix expression, evaluates it, and displays the result, aiding students in honing their coding abilities.

Input Format

The input consists of the postfix mathematical expression.

The expression will contain real numbers and mathematical operators (+, -, *, /), without any space.

Output Format

The output prints the result of evaluating the given postfix expression.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 82/
Output: 4

Answer

-

Status : Skipped ***Marks : 0/10***