Stack - VISHNU SWAROOP P S - 2347265

```c
 //Application of Stack (convert an infix expression to the postfix form)
//Input must be given as an expression using scanf function
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

// Stack structure for operators
struct Stack {
    int top;
    unsigned capacity;
    char* array;
};

// Function to initialize a stack
struct Stack* createStack(unsigned capacity) {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->array = (char*)malloc(stack->capacity * sizeof(char));
    return stack;
}

// Function to check if the stack is empty
int isEmpty(struct Stack* stack) {
    return stack->top == -1;
}

// Function to push an element onto the stack
void push(struct Stack* stack, char op) {
    stack->array[++stack->top] = op;
}

// Function to pop an element from the stack
char pop(struct Stack* stack) {
    if (!isEmpty(stack))
        return stack->array[stack->top--];
    return '$'; // '$' denotes an empty stack
```

```
}

// Function to get the precedence of an operator
int getPrecedence(char op) {
    if (op == '+' || op == '-')
        return 1;
    else if (op == '*' || op == '/')
        return 2;
    return 0;
}

// Function to convert infix expression to postfix
void infixToPostfix(char* infix, char* postfix) {
    struct Stack* stack = createStack(strlen(infix));
    int i, j;

    for (i = 0, j = -1; infix[i]; ++i) {
        // If the scanned character is an operand, add it to the postfix
expression
        if (isalnum(infix[i])) {
            postfix[++j] = infix[i];
        }
        // If the scanned character is an '(', push it onto the stack
        else if (infix[i] == '(') {
            push(stack, infix[i]);
        }
        // If the scanned character is an ')', pop and append from the
stack until an '(' is encountered
        else if (infix[i] == ')') {
            while (!isEmpty(stack) && stack->array[stack->top] != '(') {
                postfix[++j] = pop(stack);
            }
            if (!isEmpty(stack) && stack->array[stack->top] != '(')
                return; // invalid expression
            else
                pop(stack); // pop the '('
        }
        // Operator encountered
        else {
```

```
            while (!isEmpty(stack) && getPrecedence(infix[i]) <=
getPrecedence(stack->array[stack->top])) {
                postfix[++j] = pop(stack);
            }
            push(stack, infix[i]);
        }
    }

    // Pop all the remaining operators from the stack
    while (!isEmpty(stack)) {
        postfix[++j] = pop(stack);
    }

    // Null-terminate the postfix expression
    postfix[++j] = '\0';
}

int main() {
    char infix[100];
    char postfix[100];

    // Get the infix expression from the user
    printf("Enter the infix expression: ");
    scanf("%s", infix);

    // Convert infix to postfix
    infixToPostfix(infix, postfix);

    // Display the postfix expression
    printf("Postfix expression: %s\n", postfix);

    return 0;
}
```

Output :

```
PS D:\Progrmming\DSA\labworks\Lab 3> .\Stack.exe
Enter the infix expression: a+b+(c-d)*a
Postfix expression: ab+cd-a*+
PS D:\Progrmming\DSA\labworks\Lab 3> .\Stack.exe
Enter the infix expression: a+b-c
Postfix expression: ab+c-
PS D:\Progrmming\DSA\labworks\Lab 3>
```