DSA lab

**Assignment 3**

**Name:** Madhav Jha

**Roll no.:** E3-48

**Branch:** CSE (AI & ML)

# ADT Stack Operation

Stack  
#include <stdio.h>

#include <stdlib.h>

void error(char e[]) {

    printf("\nERROR: %s!!\n", e);

}

typedef struct

{

    int top;

    int maxSize;

    double\* arr;

} Stack;

Stack\*

stackInitialize()

{

    Stack\* sp;

    sp = (Stack\*)malloc(sizeof(Stack));

    sp->top = -1;

    sp->maxSize = 1000;

    sp->arr = (double\*)calloc(sp->maxSize, sizeof(double));

    return sp;

}

void

push(Stack\* sp, double c)

{

    if (sp->top >= sp->maxSize - 1)

    {

        error("StackOverflow, Stack reformed");

        sp->maxSize \*= 2;

        sp->arr = (double\*)realloc(sp->arr, sp->maxSize \* sizeof(double));

    }

    sp->arr[++sp->top] = c;

}

double

peek(Stack\* sp)

{

    if (sp->top <= -1)

    {

        error("StackUnderflow, can't peek any element");

        return -1;

    }

    return sp->arr[sp->top];

}

double

pop(Stack\* sp)

{

    if (sp->top <= -1)

    {

        error("StackUnderflow");

        return 0;

    }

    return sp->arr[sp->top--];

}

int isEmpty(Stack\* sp) {

    return sp->top == -1;

}

## a.infix-postfix

#include <stdio.h>

#include <string.h>

#include "operatorUtil.c"

#include "stack.c"

char\* infixToPostfix(char s[]) {

    Stack\* arr;

    arr = stackInitialize();

    char\* res = (char\*)malloc(1000 \* sizeof(int));;

    int k = 0;

    for (int i = 0;i < strlen(s);i++) {

        if (s[i] == ' ') {

            continue;

        }

        if (s[i] >= '0' && s[i] <= '9') {

            res[k++] = s[i];

            continue;

        }

        // in case we have -2 + 3 or +2 +3 or -(3)

        if ((i - 1 < 0 || s[i - 1] == '(') && (s[i] == '+' || s[i] == '-')) {

            res[k++] = '0';

        }

        if (isOperator(s[i]) == 1) {

            // condition is if current operator order is <= previous operator then we pop else we push

            int prev = -1;

            if (isEmpty(arr) == 0 && isOperator((char)peek(arr))) {

                prev = operatorOrder((char)peek(arr));

            }

            int curr = operatorOrder(s[i]);

            while (curr <= prev && prev != -1 && isEmpty(arr) == 0) {

                if ((char)peek(arr) == '(') {

                    break;

                }

                char c = (char)pop(arr);

                res[k++] = c;

                if (isEmpty(arr) == 0 && (char)peek(arr) != '(') {

                    prev = operatorOrder((char)peek(arr));

                }

                else {

                    break;

                }

            }

            push(arr, (char)s[i]);

        }

        else if ((char)s[i] == '(') {

            push(arr, '(');

        }

        else if ((char)s[i] == ')') {

            while ((char)peek(arr) != '(' || isEmpty(arr)) {

                char c = (char)pop(arr);

                res[k++] = c;

            }

            pop(arr);

        }

        else {

            error("Invalid character");

            break;

        }

    }

    while (isEmpty(arr) != 1) {

        res[k++] = (char)pop(arr);

    }

    res[k++] = '\0';

    return res;

}

int main() {

    char str[] = "2+4\*4/2+3";

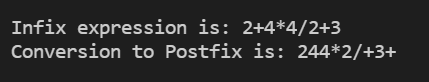
    printf("\nInfix expression is: %s", str);

    char\* res = infixToPostfix(str);

    printf("\nConversion to Postfix is: %s", res);

}

## Output:



## b.infix to prefix

#include <stdio.h>

#include <string.h>

#include "operatorUtil.c"

#include "stack.c"

char\* infixToPrefix(char s[]) {

    Stack\* arr;

    arr = stackInitialize();

    char\* res = (char\*)malloc(1000 \* sizeof(int));;

    int k = 0;

    strrev(s);

    for (int i = 0;i < strlen(s);i++) {

        if (s[i] == ')') {

            s[i] = '(';

        }

        else if (s[i] == '(') {

            s[i] = ')';

        }

        if (s[i] == ' ') {

            continue;

        }

        if (s[i] >= '0' && s[i] <= '9') {

            res[k++] = s[i];

            continue;

        }

        // in case we have -2 + 3 or +2 +3 or -(3)

        if ((i - 1 < 0 || s[i - 1] == '(') && (s[i] == '+' || s[i] == '-')) {

            res[k++] = '0';

        }

        if (isOperator(s[i]) == 1) {

            // condition is if current operator order is <= previous operator then we pop else we push

            int prev = -1;

            if (isEmpty(arr) == 0 && isOperator((char)peek(arr))) {

                prev = operatorOrder((char)peek(arr));

            }

            int curr = operatorOrder(s[i]);

            if (curr <= prev && prev != -1) {

                int b = 0;

                if (s[i] == '^') {

                    b = curr <= prev;

                }

                else {

                    b = curr < prev;

                }

                while (b && prev != -1 && isEmpty(arr) != 1) {

                    if ((char)peek(arr) == '(') {

                        break;

                    }

                    char c = (char)pop(arr);

                    res[k++] = c;

                    if (isEmpty(arr) == 0 && (char)peek(arr) != '(') {

                        prev = operatorOrder((char)peek(arr));

                    }

                    else {

                        break;

                    }

                    if (s[i] == '^') {

                        b = curr <= prev;

                    }

                    else {

                        b = curr < prev;

                    }

                }

            }

            push(arr, (char)s[i]);

        }

        else if ((char)s[i] == '(') {

            push(arr, '(');

        }

        else if ((char)s[i] == ')') {

            while ((char)peek(arr) != '(') {

                char c = (char)pop(arr);

                res[k++] = c;

            }

            pop(arr);

        }

        else {

            error("Invalid character");

            break;

        }

    }

    while (isEmpty(arr) != 1) {

        res[k++] = (char)pop(arr);

    }

    res[k++] = '\0';

    return res;

}

int main() {

    char str[] = "2+4\*4/2+3";

    printf("\nInfix expression is: %s", str);

    char\* res = infixToPrefix(str);

    printf("\nConversion to Prefix is: %s", strrev(res));

}

## Output:

