**DSA Assignment 6**

[**https://github.com/PrathamAsrani/DSA\_C/blob/master/assignment\_6.c**](https://github.com/PrathamAsrani/DSA_C/blob/master/assignment_6.c)

Suppose there are two stack both of which intersect at some point and become a single stack. The head or start pointers of both the stack are known, but the intersecting node is not known. Also, the number of nodes in each of the lists before they intersect is unknown and may be different in each list. List1 may have n nodes before it reaches the intersection point, and List2 might have m nodes before it reaches the intersection point where m and n may be m = n,m < n or m > n. Give an algorithm for finding the merging point.

Ans 1 :

#include <stdio.h>

#include <stdlib.h>

struct stack

{

    int \*data;

    int size, top;

};

int isFull(struct stack \*s);

int isEmpty(struct stack \*s);

struct stack \*create(int size);

void push(struct stack \*s, int data);

int pop(struct stack \*s);

void search(struct stack \*s1, struct stack \*s2);

int main()

{

    int m, n;

    printf("Enter size of stack 1: ");

    scanf("%d", &m);

    printf("Enter size of stack 2: ");

    scanf("%d", &n);

    struct stack \*s1 = create(m);

    struct stack \*s2 = create(n);

    int ele1, ele2;

    printf("Enter the elements for stack 1 : \n");

    for (int i = 0; i < m; i++)

    {

        scanf("%d", &ele1);

        push(s1, ele1);

    }

    printf("Enter the elements for stack 2 : \n");

    for (int i = 0; i < n; i++)

    {

        scanf("%d", &ele2);

        push(s2, ele2);

    }

    search(s1, s2);

    return 0;

}

int isFull(struct stack \*s)

{

    if (s->top == s->size - 1)

    {

        return 1;

    }

    else

    {

        return 0;

    }

}

int isEmpty(struct stack \*s)

{

    if (s->top == -1)

    {

        return 1;

    }

    else

    {

        return 0;

    }

}

struct stack \*create(int size)

{

    struct stack \*ptr = (struct stack \*)malloc(sizeof(struct stack));

    ptr->size = size;

    ptr->top = -1;

    ptr->data = (int \*)malloc(ptr->size \* sizeof(int));

    return ptr;

}

void push(struct stack \*s, int data)

{

    if (isFull(s))

    {

        printf("Stack is full\n");

    }

    else

    {

        s->top++;

        \*(s->data + s->top) = data;

    }

}

int pop(struct stack \*s)

{

    if (isEmpty(s))

    {

        printf("Stack is empty\n");

    }

    else

    {

        int top = \*(s->data + s->top);

        s->top--;

        return top;

    }

}

void search(struct stack \*s1, struct stack \*s2)

{

    int i = 0, j = 0;

    while (i < s1->size)

    {

        int key = s1->data[i];

        while (j < s2->size)

        {

            if (key == s2->data[j])

            {

                printf("Match found, same number is : %d\n", key);

                printf("The index in stack 1 is : %d\n", i+1);

                printf("The index in stack 2 is : %d\n", j+1);

            }

            j++;

        }

        j = 0;

        i++;

    }

}

Q. 2 design a single algo. to convert infix to postfix expressio and simultaneously evaluate the postfix expression. Use 2 stacks (operand and operator)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct Stack

{

    int size, top;

    int \*data;

};

/\* Infix to postfix \*/

int isFull(struct Stack \*s);

int isEmpty(struct Stack \*s);

char stacktop(struct Stack \*s);

void push(struct Stack \*s, char x);

char pop(struct Stack \*s);

int isOperator(char ch);

int precedence(char ch);

char \*infixToPostfix(char \*infix);

/\* Infix to postfix \*/

/\* Evaluation to postfix \*/

int calculation(char c, int num1, int num2);

int int\_converter(char c);

void push1(struct Stack \*s, int c);

int pop1(struct Stack \*s);

int evaluation\_of\_postfix(char \*postfix);

/\* Evaluation to postfix \*/

int main(){

    char \*infix = "a+d-e/s";

    printf("The input infix expression : %s\n", infix);

    char \*postfix;

    postfix = infixToPostfix(infix);

    printf("Postfix Expression is %s\n", postfix);

    char \*postfix\_exp = "6+8-9/1";

    int re = evaluation\_of\_postfix(&postfix\_exp);

    printf("The ans : %d\n",re);

}

int isFull(struct Stack \*s)

{

    if (s->top == s->size - 1)

    {

        return 1;

    }

    else

    {

        return 0;

    }

}

int isEmpty(struct Stack \*s)

{

    if (s->top == -1)

    {

        return 1;

    }

    else

    {

        return 0;

    }

}

char stacktop(struct Stack \*s)

{

    return s->data[s->top];

}

void push(struct Stack \*s, char x)

{

    if (isFull(s))

    {

        printf("Stack is full\n");

    }

    else

    {

        s->top++;

        s->data[s->top] = x;

    }

}

char pop(struct Stack \*s)

{

    if (isEmpty(s))

    {

        printf("Stack id empty\n");

    }

    else

    {

        char c = s->data[s->top];

        s->top--;

        return c;

    }

}

int isOperator(char ch)

{

    if (ch == '+' || ch == '-' || ch == '\*' || ch == '/')

    {

        return 1;

    }

    return 0;

}

int precedence(char ch)

{

    if (ch == '/' || ch == '\*')

    {

        return 2;

    }

    else if (ch == '+' || ch == '-')

    {

        return 1;

    }

    return 0;

}

char \*infixToPostfix(char \*infix)

{

    struct Stack \*p = (struct Stack \*)malloc(sizeof(struct Stack));

    p->size = strlen(infix);

    p->top = -1;

    p->data = (int \*)malloc(p->size \* sizeof(int));

    char \*postfix = (char \*)malloc((p->size+1)\*sizeof(char));

    // if(!isOperator(infix))

    int i = 0, j = 0;

    while (infix[i] != '\0')

    {

        if (!isOperator(infix[i]))

        {

            postfix[j] = infix[i];

            i++;

            j++;

        }else{

            if(precedence(infix[i])>precedence(stacktop(p))){

                push(p, infix[i]);

                i++;

            }else{

                postfix[j] = pop(p);

                j++;

            }

        }

    }

    while (!isEmpty(p))

    {

        postfix[j] = pop(p);

        j++;

    }

    postfix[j] = '\0';

    return postfix;

}

int calculation(char c, int num1, int num2){

    int result;

    switch (c)

    {

    case '+':

        result = num1+num2;

        break;

    case '-':

        result = num1-num2;

        break;

    case '\*':

        result = num1\*num2;

        break;

    case '/':

        result = num1/num2;

        break;

    case '%':

        result = num1%num2;

        break;

    case '^':

        result = num1^num2;

        break;

    default:

        break;

    }

    return result;

}

int int\_converter(char c){

    int a = (int)c - 48;

    return a;

}

void push1(struct Stack \*s, int c){

    if(isFull(s)){

        printf("Stack is full\n");

    }else{

        s->top++;

        s->data[s->top] = c;

    }

}

int pop1(struct Stack \*s){

    if(isEmpty(s)){

        printf("Stack is empty\n");

    }else{

        int val = s->data[s->top];

        s->top--;

        return val;

    }

}

int evaluation\_of\_postfix(char \*postfix){

    struct Stack \*operator = (struct Stack \*)malloc(sizeof(struct Stack));

    operator->top = -1;

    operator->size  = strlen(postfix);

    operator->data = (int\*) malloc(operator->size \* sizeof(int));

    int i = 0;

    while(postfix[i] != '\0'){

        if(!isOperator(postfix[i])){

            push1(operator, (int)postfix[i]);

            i++;

        }

        else{

            int a = int\_converter(pop1(operator));

            int b = int\_converter(pop1(operator));

            int c = calculation(postfix[i], b, a);

            push1(operator, c);

            i++;

        }

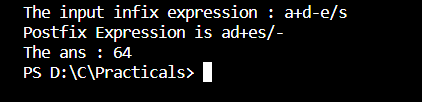
    }

    int result = (pop1(operator));

    return result;

}

Output :



Conclusion : Hence we successfully implemented Stack operations, Infix to postfix using stack, and postfix evaluation.