Computer Engineering Department, S.V.N.I.T. Surat.

B Tech (CO) –IInd Year semester-III

Course: *Data Structures CO203*

**Tutorial-V**

Write an algorithm for the following using stack and implement in C.

1A.) Infix to Postfix Conversion

1B.) Infix to Prefix Conversion

Code:

*#include* <stdio.h>

*#include* <stdlib.h> *// exit func*

*#include* <ctype.h>  *// for isdigit*

*#include* <string.h>

*#define* SIZE 100

char stack[SIZE];

int top = -1;

*//Function Declarations*

void push(char ele);

char pop();

int is\_operator(char symbol);

int precedence(char symbol);

void Infix\_To\_PostFix(char infix\_exp[], char postfix\_exp[]);

void Infix\_to\_Prefix(char infix\_exp[], char prefix\_exp[]);

void Bracket(char \*exp);

int main()

{

    char infix[SIZE], postfix[SIZE], prefix[SIZE];

    int cnt = 0;

    printf("Enter the Number of Infix Expression to Convert to PostFix & Prefix Expression :\n");

    scanf("%d", &cnt);

    fflush(stdin);

*while* (cnt--)

    {

        printf("Enter Valid Infix Expression : ");

        gets(infix);

        Infix\_To\_PostFix(infix, postfix);

        printf("Postfix Expression : ");

        puts(postfix);

        Infix\_to\_Prefix(infix, prefix);

        printf("Prefix Expression  : ");

        puts(prefix);

        printf("\n");

        top = -1; *//Empty the Stack*

    }

*return* 0;

*// LIMITATION : SINGLE LETTER DIGITS AND VARIABLES*

}

*// Function Definations*

void push(char ele)

{

*if* (top >= SIZE - 1)

    {

        printf("\nStack Overflow!");

    }

*else*

    {

        top = top + 1;

        stack[top] = ele;

    }

}

char pop()

{

    char ele;

*if* (top < 0)

    {

*// underflow -> invalid expression*

*// OR '(' and ')' are not matched*

        printf("Stack Under Flow! : Invalid Infix Expression Entered!");

        getchar();

        exit(1);

    }

*else*

    {

        ele = stack[top];

        top = top - 1;

*return* (ele);

    }

}

int is\_operator(char symbol)

{

*if* (symbol == '^' || symbol == '\*' || symbol == '/' || symbol == '+' || symbol == '-' || symbol == '%')

    {

*return* 1;

    }

*else*

    {

*return* 0;

    }

}

int precedence(char symbol)

{

*// Exponent > (\* OR /) > (+ OR -)*

*if* (symbol == '^')

    {

*return* (3);

    }

*else* *if* (symbol == '\*' || symbol == '/')

    {

*return* (2);

    }

*else* *if* (symbol == '+' || symbol == '-') */\* lowest precedence \*/*

    {

*return* (1);

    }

*else*

    {

*return* (0);

    }

}

void Infix\_To\_PostFix(char infix\_exp[], char postfix\_exp[])

{

    int i, j;

    char ele;

    char x;

*//push '(' onto stack*

    push('(');

*// add ')' to infix expression*

    strcat(infix\_exp, ")");

    i = 0;

    j = 0;

*// Initialize with First Char*

    ele = infix\_exp[i];

*// run loop till end of infix expression*

*while* (ele != '\0')

    {

*if* (ele == '(')

        {

            push(ele);

        }

*else* *if* (isdigit(ele) || isalpha(ele))

        {

*// add operand symbol to postfix expr*

            postfix\_exp[j] = ele;

            j++;

        }

*else* *if* (is\_operator(ele) == 1)

        {

            x = pop();

*while* (is\_operator(x) == 1 && precedence(x) >= precedence(ele))

            {

*//add them to postfix expresion*

                postfix\_exp[j] = x;

                j++;

                x = pop();

*// pop all higher precendence operator*

            }

*// For -> While Loop popped element*

            push(x);

*//push current oprerator symbol onto stack*

            push(ele);

        }

*else* *if* (ele == ')')

        {

            x = pop();

*//keep popping until '(' encounterd*

*while* (x != '(')

            {

                postfix\_exp[j] = x;

                j++;

                x = pop();

            }

        }

*else*

        {

*// Neither of Above Symbols*

            printf("\nInvalid Infix Expression!\n");

            getchar();

            exit(1);

        }

        i++;

*// Next Symbol*

        ele = infix\_exp[i];

    }

*if* (top > 0)

    {

        printf("\nInvalid Infix Expression.\n");

        getchar();

        exit(1);

    }

*//add sentinel "\0" else puts() fucntion will print entire postfix[] array upto SIZE*

    postfix\_exp[j] = '\0';

}

void Bracket(char \*exp)

{

    int i = 0;

*while* (exp[i] != '\0')

    {

*if* (exp[i] == '(')

            exp[i] = ')';

*else* *if* (exp[i] == ')')

            exp[i] = '(';

        i++;

    }

}

void Infix\_to\_Prefix(char infix\_exp[], char prefix\_exp[])

{

    char tmp[SIZE];

    strcpy(tmp, infix\_exp);

*// Reverse Given Infix Expression*

    strrev(tmp);

*//change Bracket*

    Bracket(tmp);

*//get postfix of tmp*

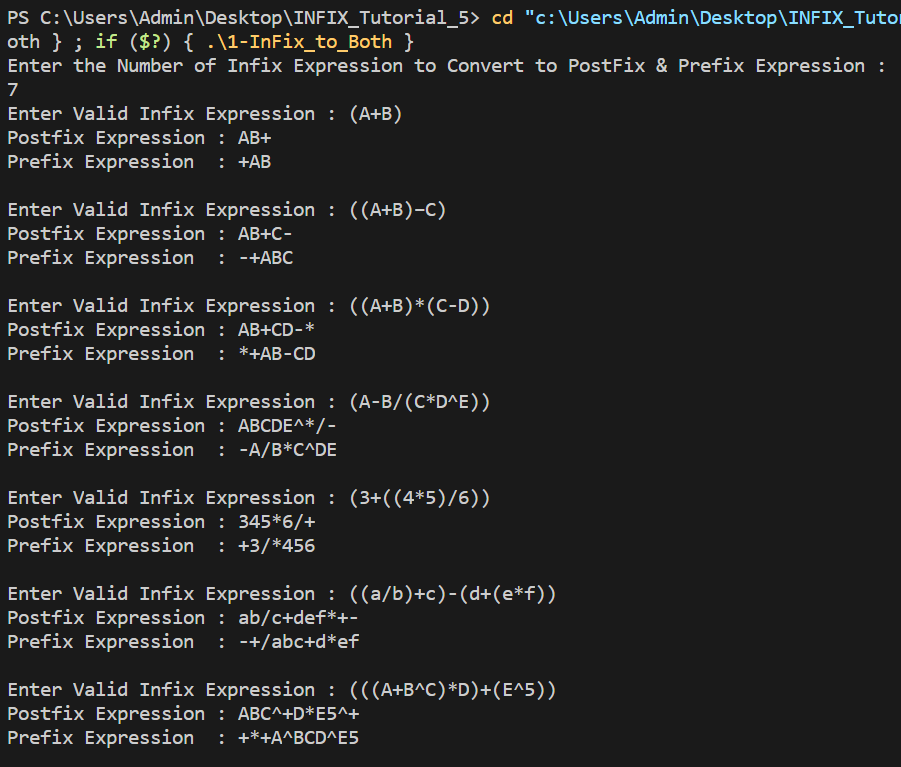
    Infix\_To\_PostFix(tmp, prefix\_exp);

*// reverse string again*

    strrev(prefix\_exp);

}

Sample Test Cases:



2A.) Postfix to Prefix Conversion

2B.) Postfix to Infix Conversion

Code:

*#include* <stdio.h>

*#include* <string.h>

*#include* <math.h>

*#include* <stdlib.h>

*#define* BLANK ' '

*#define* TAB '\t'

void PostFix\_To\_InFix(char postfix\_exp[], char infix\_exp[]);

*#define* MAX 100

char stack2[MAX];

int top2;

int is\_operator(char symbol);

void push2(char c);

char pop2();

void PostFix\_To\_PreFix(char postfix[], char prefix[]);

*#define* SIZE 100

int top;

char stack[SIZE][SIZE];

void push(char \*str);

char \*pop();

int isempty();

int white\_space(char symbol);

int main()

{

    char infix[SIZE], postfix[SIZE], prefix[SIZE];

    int cnt = 0;

    printf("Enter the Number of PostFix Expression to Convert to PreFix & Infix Expression :\n");

    scanf("%d", &cnt);

    fflush(stdin);

*while* (cnt--)

    {

        top = -1;

        printf("Enter Valid PostFix Expression : ");

        gets(postfix);

        PostFix\_To\_PreFix(postfix, prefix);

        printf("Prefix Expression : ");

        puts(prefix);

        top2 = -1;

        PostFix\_To\_InFix(postfix, infix);

        printf("Infix Expression  : ");

        puts(infix);

        printf("\n");

    }

*// LIMITATION : SINGLE LETTER DIGITS AND VARIABLES*

*return* 0;

}

void PostFix\_To\_PreFix(char postfix[], char prefix[])

{

    int i;

    char operand1[SIZE], operand2[SIZE];

    char symbol;

    char temp[2];

    char strin[SIZE];

*for* (i = 0; i < strlen(postfix); i++)

    {

        symbol = postfix[i];

        temp[0] = symbol;

        temp[1] = '\0';

*if* (!white\_space(symbol))

        {

*switch* (symbol)

            {

*case* '+':

*case* '-':

*case* '\*':

*case* '/':

*case* '%':

*case* '^':

*// string = operator + operand2 + operand1*

                strcpy(operand1, pop());

                strcpy(operand2, pop());

                strcpy(strin, temp);

                strcat(strin, operand2);

                strcat(strin, operand1);

*//push the string in Stack*

                push(strin);

*break*;

*// Operand should be pushed in Stack*

*default*:

                push(temp);

            }

        }

    }

    strcpy(prefix, stack[0]);

}

void push(char \*str)

{

*if* (top > SIZE)

    {

        printf("\nStack Overflow!\n");

        exit(1);

    }

*else*

    {

        top = top + 1;

        strcpy(stack[top], str);

    }

}

char \*pop()

{

*if* (top == -1)

    {

        printf("\nStack underflow || Enter Valid PostFix Expression\n");

        exit(2);

    }

*else*

*return* (stack[top--]);

}

int isempty()

{

*if* (top == -1)

*return* 1;

*else*

*return* 0;

}

int white\_space(char symbol)

{

*if* (symbol == BLANK || symbol == TAB || symbol == '\0')

*return* 1;

*else*

*return* 0;

}

*// ~~~~~~~~~~~~~~~~~~~~~~~~~~~POSTFIX TO INFIX~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*

void PostFix\_To\_InFix(char postfix\_exp[], char infix\_exp[])

{

    char str[MAX];

    int i, j = 0;

    strcpy(str, postfix\_exp);

    strrev(str);

*for* (i = 0; i < MAX; i++)

        stack2[i] = '\0';

    int n = strlen(str);

*for* (i = 0; i < n; i++)

    {

*if* (is\_operator(str[i]))

        {

            push2(str[i]);

        }

*else*

        {

            infix\_exp[j] = str[i];

            j++;

            infix\_exp[j] = pop2();

            j++;

        }

    }

    infix\_exp[j] = str[top--];

    strrev(infix\_exp);

}

void push2(char c)

{

    stack2[++top2] = c;

}

char pop2()

{

*return* stack2[top2--];

}

int is\_operator(char symbol)

{

*if* (symbol == '^' || symbol == '\*' || symbol == '/' || symbol == '+' || symbol == '-')

    {

*return* 1;

    }

*else*

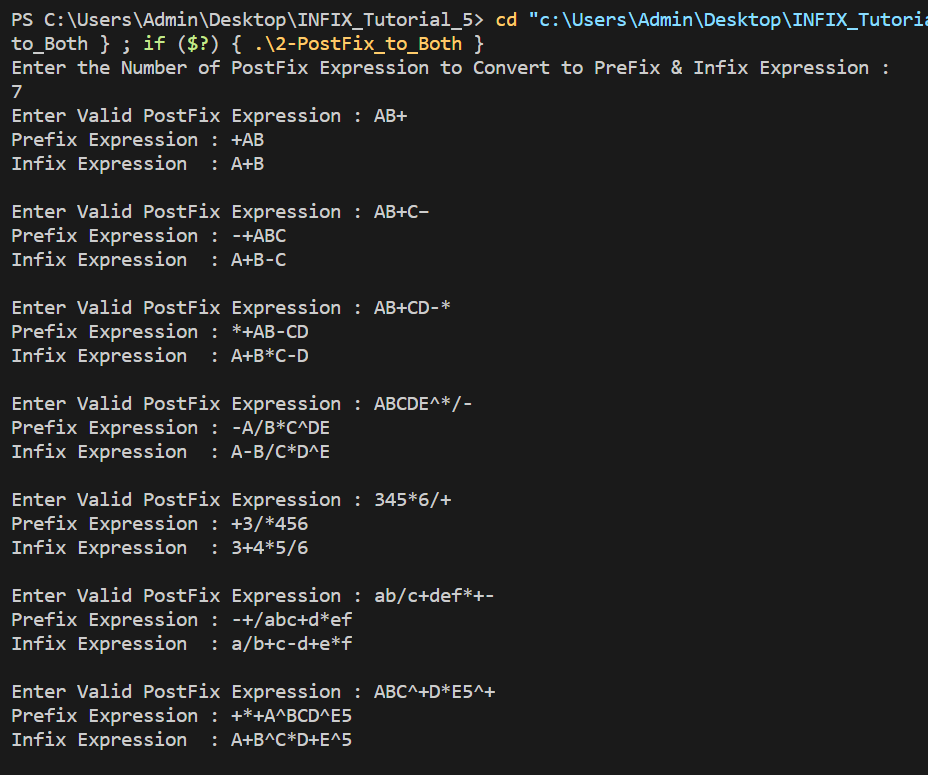
    {

*return* 0;

    }

}

Test Cases:



3A.) Prefix to Postfix Conversion

3B.) Prefix to Infix Conversion

Code:

*#include* <stdio.h>

*#include* <string.h>

*#include* <math.h>

*#include* <stdlib.h>

*#define* BLANK ' '

*#define* TAB '\t'

void PreFix\_To\_InFix(char prefix\_exp[], char infix\_exp[]);

*#define* MAX 100

char stack2[MAX];

int top2;

int is\_operator(char symbol);

void push2(char c);

char pop2();

void PreFix\_To\_PostFix(char prefix[], char postfix[]);

*#define* SIZE 100

char stack[SIZE][SIZE];

int top;

void push(char \*str);

char \*pop();

int isempty();

int white\_space(char symbol);

int main()

{

    char infix[SIZE], postfix[SIZE], prefix[SIZE];

    int cnt = 0;

    printf("Enter the Number of PreFix Expression to Convert to PostFix & Infix Expression :\n");

    scanf("%d", &cnt);

    fflush(stdin);

*while* (cnt--)

    {

        top = -1;

        printf("Enter Valid PreFix Expression : ");

        gets(prefix);

        PreFix\_To\_PostFix(prefix, postfix);

        printf("PostFix Expression : ");

        puts(postfix);

        top2 = -1;

        PreFix\_To\_InFix(prefix, infix);

        printf("Infix Expression   : ");

        puts(infix);

        printf("\n");

    }

}

void PreFix\_To\_PostFix(char prefix[], char postfix[])

{

    int i;

    char operand1[SIZE], operand2[SIZE];

    char symbol;

    char temp[2];

    char strin[SIZE];

*for* (i = strlen(prefix) - 1; i >= 0; i--)

    {

        symbol = prefix[i];

        temp[0] = symbol;

        temp[1] = '\0';

*if* (!white\_space(symbol))

        {

*switch* (symbol)

            {

*case* '+':

*case* '-':

*case* '\*':

*case* '/':

*case* '%':

*case* '^':

*// string = operand1 + operand2 + operator*

                strcpy(operand1, pop());

                strcpy(operand2, pop());

                strcpy(strin, operand1);

                strcat(strin, operand2);

                strcat(strin, temp);

*//push the string in Stack*

                push(strin);

*break*;

*// Operand should be pushed in Stack*

*default*:

                push(temp);

            }

        }

    }

*// printf("\nPostfix Expression :: ");*

*// puts(stack[0]);*

    strcpy(postfix, stack[0]);

}

void push(char \*str)

{

*if* (top > SIZE)

    {

        printf("\nStack Overflow!\n");

        exit(1);

    }

*else*

    {

        top = top + 1;

        strcpy(stack[top], str);

    }

}

char \*pop()

{

*if* (top == -1)

    {

        printf("\nStack UnderFlow!Invalid Prefix Expression!\n");

        exit(2);

    }

*else*

*return* (stack[top--]);

}

int isempty()

{

*if* (top == -1)

*return* 1;

*else*

*return* 0;

}

int white\_space(char symbol)

{

*if* (symbol == BLANK || symbol == TAB || symbol == '\0')

*return* 1;

*else*

*return* 0;

}

*// ~~~~~~~~~~~~~~~~~~~~~~~~~~~PREFIX TO INFIX~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*

void PreFix\_To\_InFix(char prefix\_exp[], char infix\_exp[])

{

    char str[MAX];

    strcpy(str, prefix\_exp);

    int i, j = 0;

    char a, b, op;

    int n = strlen(str);

*for* (i = 0; i < MAX; i++)

        stack2[i] = '\0';

*for* (i = 0; i < n; i++)

    {

*if* (is\_operator(str[i]))

        {

            push2(str[i]);

        }

*else*

        {

            op = pop2();

            a = str[i];

            infix\_exp[j] = a;

            j++;

            infix\_exp[j] = op;

            j++;

        }

    }

    infix\_exp[j] = str[top--];

}

int is\_operator(char symbol)

{

*if* (symbol == '^' || symbol == '\*' || symbol == '/' || symbol == '+' || symbol == '-')

    {

*return* 1;

    }

*else*

    {

*return* 0;

    }

}

void push2(char c)

{

    stack2[++top2] = c;

}

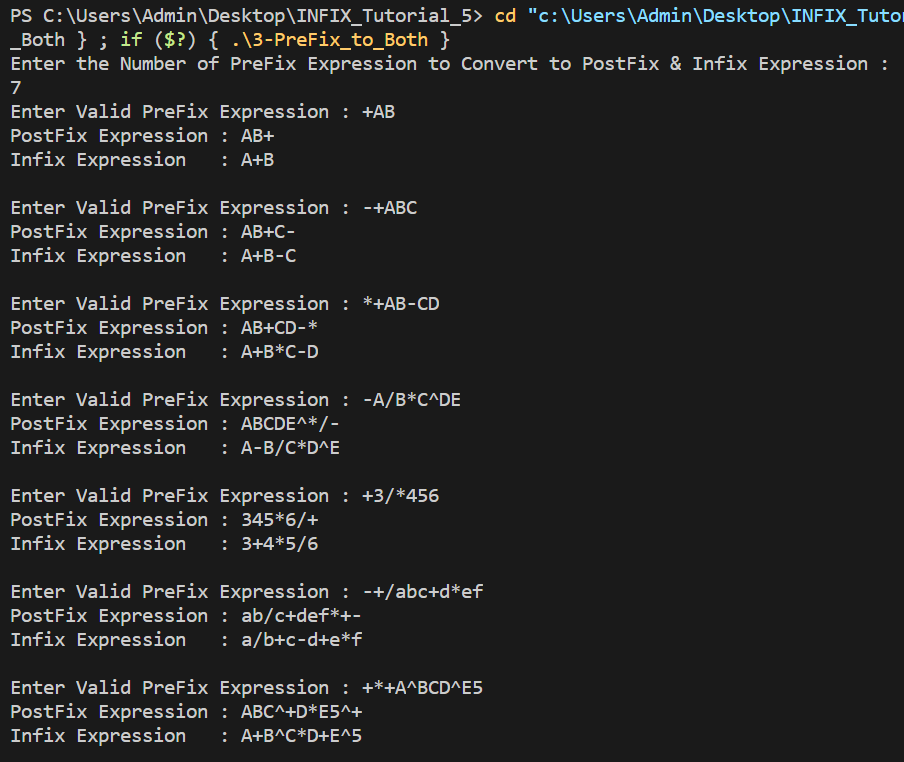
char pop2()

{

*return* stack2[top2--];

}

Test Cases:



*All the three Programs are Having Same 7 Expressions as Test Cases and* ***All Results in Same Infix, Prefix and Postfix Expressions****. Hence the Above Code for Conversion from any Form to Another is* ***Correct & Implemented in Right Way****.*

Submitted By:

Roll Number: **U19CS012** (*D-12*)

Name: *Bhagya Rana*