Computer Engineering Department, S.V.N.I.T. Surat.
B Tech (CO) –II[nd] 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:

```c
#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 Defination

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:

```
PS C:\Users\Admin\Desktop\INFIX_Tutorial_5> cd "c:\Users\Admin\Desktop\INFIX_Tuto
oth } ; if ($?) { .\1-InFix_to_Both }
Enter the Number of Infix Expression to Convert to PostFix & Prefix Expression :
7
Enter Valid Infix Expression : (A+B)
Postfix Expression : AB+
Prefix Expression  : +AB

Enter Valid Infix Expression : ((A+B)-C)
Postfix Expression : AB+C-
Prefix Expression  : -+ABC

Enter Valid Infix Expression : ((A+B)*(C-D))
Postfix Expression : AB+CD-*
Prefix Expression  : *+AB-CD

Enter Valid Infix Expression : (A-B/(C*D^E))
Postfix Expression : ABCDE^*/-
Prefix Expression  : -A/B*C^DE

Enter Valid Infix Expression : (3+((4*5)/6))
Postfix Expression : 345*6/+
Prefix Expression  : +3/*456

Enter Valid Infix Expression : ((a/b)+c)-(d+(e*f))
Postfix Expression : ab/c+def*+-
Prefix Expression  : -+/abc+d*ef

Enter Valid Infix Expression : (((A+B^C)*D)+(E^5))
Postfix Expression : ABC^+D*E5^+
Prefix Expression  : +*+A^BCD^E5
```

2A.) Postfix to Prefix Conversion

2B.) Postfix to Infix Conversion

Code:

```c
#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:

```
PS C:\Users\Admin\Desktop\INFIX_Tutorial_5> cd "c:\Users\Admin\Desktop\INFIX_Tutoria
to_Both } ; if ($?) { .\2-PostFix_to_Both }
Enter the Number of PostFix Expression to Convert to PreFix & Infix Expression :
7
Enter Valid PostFix Expression : AB+
Prefix Expression : +AB
Infix Expression  : A+B

Enter Valid PostFix Expression : AB+C-
Prefix Expression : -+ABC
Infix Expression  : A+B-C

Enter Valid PostFix Expression : AB+CD-*
Prefix Expression : *+AB-CD
Infix Expression  : A+B*C-D

Enter Valid PostFix Expression : ABCDE^*/-
Prefix Expression : -A/B*C^DE
Infix Expression  : A-B/C*D^E

Enter Valid PostFix Expression : 345*6/+
Prefix Expression : +3/*456
Infix Expression  : 3+4*5/6

Enter Valid PostFix Expression : ab/c+def*+-
Prefix Expression : -+/abc+d*ef
Infix Expression  : a/b+c-d+e*f

Enter Valid PostFix Expression : ABC^+D*E5^+
Prefix Expression : +*+A^BCD^E5
Infix Expression  : A+B^C*D+E^5
```

3A.) Prefix to Postfix Conversion

3B.) Prefix to Infix Conversion

Code:

```c
#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:

```
PS C:\Users\Admin\Desktop\INFIX_Tutorial_5> cd "c:\Users\Admin\Desktop\INFIX_Tuto
_Both } ; if ($?) { .\3-PreFix_to_Both }
Enter the Number of PreFix Expression to Convert to PostFix & Infix Expression :
7
Enter Valid PreFix Expression : +AB
PostFix Expression : AB+
Infix Expression   : A+B

Enter Valid PreFix Expression : -+ABC
PostFix Expression : AB+C-
Infix Expression   : A+B-C

Enter Valid PreFix Expression : *+AB-CD
PostFix Expression : AB+CD-*
Infix Expression   : A+B*C-D

Enter Valid PreFix Expression : -A/B*C^DE
PostFix Expression : ABCDE^*/-
Infix Expression   : A-B/C*D^E

Enter Valid PreFix Expression : +3/*456
PostFix Expression : 345*6/+
Infix Expression   : 3+4*5/6

Enter Valid PreFix Expression : -+/abc+d*ef
PostFix Expression : ab/c+def*+-
Infix Expression   : a/b+c-d+e*f

Enter Valid PreFix Expression : +*+A^BCD^E5
PostFix Expression : ABC^+D*E5^+
Infix Expression   : A+B^C*D+E^5
```

*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*