St. Francis Institute of Technology, Mumbai-400 103
Department of Information Technology

A.Y. 2020-2021
Class: SE-ITA/B, Semester: III
Subject: DATA STRUCTURE LAB

Experiment – 8 Study of Infix to Postfix transformation

1. Aim: Write a program to convert an expression from infix to postfix expression.

2. Objectives: After study of this experiment, the student will be able to
- Understand how to convert infix to postfix
- To learn the applications of stacks
- Implement an algorithm using computer to solve the given problem

3. Outcomes: After study of this experiment, the student will be able to • Illustrate and examine the methods of stacks to various real time problems • Develop an algorithm for various problem on infix to postfix

4. Prerequisite: Stack and its operations

5. Requirements: PC and Turbo C compiler version 3.0

6. Pre-Experiment Exercise:
Brief Theory:

A. What is Polish Notation?

Polish notation is a notation form for expressing arithmetic, logic and algebraic equations. Its most basic distinguishing feature is that operators are placed on the left of their operands. If the operator has a defined fixed number of operands, the syntax does not require brackets or parenthesis to lessen ambiguity.

Polish notation is also known as prefix notation, prefix Polish notation, normal Polish notation, Warsaw notation and Lukasiewicz notation.

Infix notation with parenthesis: (3 + 2) * (5 – 1)

Polish notation: * + 3 2 – 5 1

B. Definition

The way to write arithmetic expression is known as a **notation**. An arithmetic expression can be written in three different but equivalent notations, i.e., without changing the essence or output of an expression. These notations are −

- Infix Notation
- Prefix (Polish) Notation
- Postfix (Reverse-Polish) Notation

These notations are named as how they use operator in expression. We shall learn the same here in this chapter.

i)Infix Notation

We write expression in **infix** notation, e.g. a - b + c, where operators are used **in**-between operands. It is easy for us humans to read, write, and speak in infix notation but the same does not go well with computing devices. An algorithm to process infix notation could be difficult and costly in terms of time and space consumption.

ii)Prefix Notation

In this notation, operator is **prefix**ed to operands, i.e. operator is written ahead of operands. For example, **+ab**. This is equivalent to its infix notation **a + b**. Prefix notation is also known as **Polish Notation**.

iii)Postfix Notation

This notation style is known as **Reversed Polish Notation**. In this notation style, the operator is **postfix**ed to the operands i.e., the operator is written after the operands. For example, **ab+**. This is equivalent to its infix notation **a + b**.

C. The following table briefly tries to show the difference in all three notations −

Sr.No.	Infix Notation	Prefix Notation	Postfix Notation
1	a + b	+ a b	a b +
2	(a + b) ∗ c	∗ + a b c	a b + c ∗
3	a ∗ (b + c)	∗ a + b c	a b c + ∗
4	a / b + c / d	+ / a b / c d	a b / c d / +
5	(a + b) ∗ (c + d)	∗ + a b + c d	a b + c d + ∗
6	((a + b) ∗ c) - d	- ∗ + a b c d	a b + c ∗ d -

7. Laboratory Exercise

A. Procedure

Write a C program to convert an expression from infix to postfix expression.

```c
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define MAX 100
char st[MAX];
int top = -1;
```

```c
void push(char st[], char);
char pop(char st[]);
void InfixtoPostfix(char source[], char target[]);
int getPriority(char);

int main()
{
    char infix[100], postfix[100];
    printf("\n Enter any infix expression : ");
    gets(infix);
    strcpy(postfix, "");
    InfixtoPostfix(infix, postfix);
    printf("\n The corresponding postfix expression is : ");
    puts(postfix);
    return 0;
}
void InfixtoPostfix(char source[], char target[])
{
    int i=0, j=0;
    char temp;
    strcpy(target, "");
    while(source[i]!='\0')
    {
        if(source[i]=='(')
        {
            push(st, source[i]);
            i++;
        }
        else if(source[i] == ')')
        {
            while((top!=-1) && (st[top]!='('))
            {
                target[j] = pop(st);
                j++;
            }
            if(top==-1)
            {
                printf("\n INCORRECT EXPRESSION");
                exit(1);
            }
            temp = pop(st);//remove left parenthesis
            i++;
        }
```

```
        else if(isdigit(source[i]) || isalpha(source[i]))
        {
            target[j] = source[i];
            j++;
            i++;
        }
        else if (source[i] == '+' || source[i] == '-' || source[i] == '*'
|| source[i] == '/' || source[i] == '%')
        {
            while( (top!=-1) && (st[top]!= '(') && (getPriority(st[top])
>= getPriority(source[i])))
            {
                target[j] = pop(st);
                j++;
            }
            push(st, source[i]);
            i++;

        }
        else
        {
            printf("\n INCORRECT ELEMENT IN EXPRESSION");
            exit(1);
        }
    }
    while((top!=-1) && (st[top]!='('))
    {
        target[j] = pop(st);
        j++;
    }
    target[j]='\0';
}
int getPriority(char op)
{
    if(op=='/' || op == '*' || op=='%')
        return 1;
    else if(op=='+' || op=='-')
        return 0;
}

void push(char st[], char val)
{
    if(top==MAX-1)
```

```
        printf("\n STACK OVERFLOW");
    else
    {
        top++;
        st[top]=val;
    }
}

char pop(char st[])
{
    char val=' ';
    if(top==-1)
        printf("\n STACK UNDERFLOW");
    else
    {
        val=st[top];
        top--;
    }
    return val;
}
```

B. Result/Observation/Program code:

Observe the output for the above code and print it.

```
D:\College\DSA\Experiments>cd Exp8

D:\College\DSA\Experiments\Exp8>Exp8

 Enter any infix expression : A+B-C*D

 The corresponding postfix expression is : AB+CD*-

D:\College\DSA\Experiments\Exp8>
```

8. Post-Experiments Exercise

A. Questions:

1. Convert the given expression from infix to postfix

   A – (B / C + (D % E * F) / G)* H

B. Conclusion:

1. Summary of Experiment
2. Importance of Experiment

C. References:

1. S. K Srivastava, Deepali Srivastava; Data Structures through C in Depth; BPB Publications; 2011.
2. Reema Thareja; Data Structures using C; Oxford.
3. Data Structures A Pseudocode Approach with C, Richard F. Gilberg & Behrouz A. Forouzan, second edition, CENGAGE Learning.

Yash Mahajan SE IT B 04 191061

Page: Date: 1/1

A Questions :-

1. Convert the following expression from infix to postfix

A-(B/C+(D%E$\cancel{*}$*F)/G)*H

Element	Stack	Postfix	Action
A		A	print A
-	-	A	push -
(	-(	A	push (
B	-(	AB	print B
/	-(/	AB	push /
C	-(/	ABC	print B
+	-(+	ABC/	pop /, push +
(	-(+(	ABC/	Push (
D	-(+(	ABC/D	print D
%	-(+(%	ABC/D	push %
E	-(+(%	ABC/DE	print E
*	-(+(*	ABC/DE%	pop %, push *
F	-(+(*	ABC/DE%F	print F
)	-(+	ABC/DE%F*	pop *, (
/	-(+/	ABC/DE%F*	push /
G	-(+/	ABC/DE%F*G	print G
)	-	ABC/DE%F*G/+	pop /, pop +
*	-*	ABC/DE%F*G/+	push *
H	*-*	ABC/DE%F*G/+H	print H
	-	ABC/DE%F*G/+H*	pop *
		ABC/DE%F*G/+H*-	pop -

∴ ABCID E↑F*G/+H*- is the corresponding postfix expression.

B. Conclusion :-

In this experiment we have written C programs to implement application of stack data structure i.e. to convert a infix expression to postfix expression using the stack datastructure.

One disadvantage of the infix notation is that we need to use parenthesis to control the evaluation of the operators. This makes the evaluation of infix expression more difficult In postfix expression, operator precedence does not matter. The operator that occurs first in the expression is operated first on the operands. This makes evaluation simpler.