12. Problem Statement: Infix to Prefix

**Problem Analysis:**

An Infix expression is converted into Prefix using two stacks, one for operator and another for operands. The infix sting is read in an array, from which symbols/characters are fetched one by one and the following checks are performed:

• If symbol is an operand, push it in operand’s stack.

• Initialize operator stack with a dummy operator with the least precedence.

• If the symbol is an operator, then do the following:

o Pop an operator from the operator stack.

o Check its precedence with the symbol.

o If the precedence of the symbol is higher, then simply push the popped operator and then the symbol.

o Else if the precedence of symbol is lower (or equal to) than the popped operator, pop two operands, (opnd2 and opnd1) and perform concatenation operation (optr, opnd1, opnd2).

o Push the result in operand’s stack.

o Repeat the above steps until the symbol becomes less than the popped operator.

• After the string ends, start popping an operator from the operator stack and two operands from operand stack.

• Repeat step (d) until dummy operator (#) is found.

• Pop the expression from the operand stack and return it, which is the desired Prefix equivalent of given Infix string.

**Algorithm:**

*Step 1.* Push “)” onto STACK, and add “(“ to end of the A

*Step 2.* Scan A from right to left and repeat step 3 to 6 for each element of A until the STACK is empty

*Step 3.* If an operand is encountered add it to B

*Step 4.* If a right parenthesis is encountered push it onto STACK

*Step 5.* If an operator is encountered then:

a. Repeatedly pop from STACK and add to B each operator (on the top of STACK) which has same or higher precedence than the operator.

b. Add operator to STACK

*Step 6.* If left parenthesis is encountered then

a. Repeatedly pop from the STACK and add to B (each operator on top of stack until a left parenthesis is encountered)

b. Remove the left parenthesis

*Step 7.* Exit.

**Source Code:**

#define SIZE 50 */\* Size of Stack \*/*

#include<string.h>

#include<ctype.h>

#include<stdio.h>

char s[SIZE]; int top=-1;

push(char elem)

{

s[++top]=elem;

}

char pop()

{

return(s[top--]);

}

int pr(char elem)

{ */\* Function for precedence \*/*

switch(elem)

{

case '#': return 0;

case ')': return 1;

case '+':

case '-': return 2;

case '\*':

case '/':return 3;

}

}

int main()

{

char infx[50],prfx[50],ch,elem;

int i=0,k=0;

printf("Read the Infix Expression:\n");

scanf("%s",infx);

push('#');

strrev(infx);

while( (ch=infx[i++]) != '\0')

{

if( ch == ')')

push(ch);

else if(isalnum(ch))

prfx[k++]=ch;

else if( ch == '(')

{

while( s[top] != ')')

prfx[k++]=pop();

elem=pop(); /\* Remove ) \*/

}

else

{

while( pr(s[top]) >= pr(ch) )

prfx[k++]=pop(); push(ch);

}

}

while( s[top] != '#') */\* Pop from stack till empty \*/*

prfx[k++]=pop();

prfx[k]='\0'; */\* Make prfx as valid string \*/*

strrev(prfx);

strrev(infx);

printf("\nGiven Infix Expn:\n%s \n\nPrefix Expn:\n%s\n",infx,prfx);

}

**Sample Input:**

Read the Infix Expression:

A\*B+C/D

**Sample Output:**

Given Infix Expn:

A\*B+C/D

Prefix Expn:

+\*AB/CD