**TITLE 37**

Write a C program to convert the given in-fix expression to pre-fix expression using STACK

**OBJECTIVE:**

By the end of this problem we will be able to convert a in-fix expression to pre-fix expression using STACK

**PROBLEM STATEMENT:**

In this program we use STACK to convert a in-fix expression to pre-fix expression. Input from user:

Enter Infix expression:

Once the program is free of any errors then the output is printed.

**ALGORITHM:**

START

Define variables: top, stack

INPUT: Read from the keyboard

COMPUTATION: Computing how infix expression is converted to prefix expression using STACK

DISPLAY: Displaying the converted prefix expression

STOP

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

#define SIZE 100

// Global Variable Declaration

char stack[SIZE];

int top = -1;

//Global Function Declaration

void push(char c);

char pop();

int isoperator(char symbol);

int precedence(char symbol);

void InfixToPrefix(char infix\_exp[], char prefix\_exp[]);

// main() function begins

void main()

{

// Declare infix string and prefix string

char infix[SIZE], prefix[SIZE];

printf("\n\n Enter Infix expression : ");

gets(infix);

// Call to convert

InfixToPrefix(infix,prefix);

printf("\n Prefix Expression: ");

// Print prefix expression

puts(prefix);

}

void InfixToPrefix(char infix\_exp[], char prefix\_exp[])

{

int i, j, k, pos, len;

char item, x, rev[SIZE];

// Reverse the infix expression

pos=0;

len=strlen(infix\_exp);

for(k=len-1;k>=0;k--)

{

rev[pos]=infix\_exp[k];

pos++;

}

rev[pos]='\0';

strcpy(infix\_exp,rev);

// Make Every “ ( ” as “ ) ” and every “ ) ” as “ ( ”

for(i=0; infix\_exp[i]!='\0'; i++)

{

if(infix\_exp[i] == ')')

infix\_exp[i] = '(';

else if(infix\_exp[i] == '(')

infix\_exp[i] = ')';

}

//Convert expression to postfix form.

// push '(' onto stack

push('(');

// add ')' to infix expression

strcat(infix\_exp,")");

i=0;

j=0;

// Initialize before loop

item=infix\_exp[i];

// Run loop till end of infix expression

while(item != '\0')

{

if(item == '(')

{

push(item);

}

else if( isdigit(item) || isalpha(item))

{

// Add operand symbol to postfix expression

prefix\_exp[j] = item;

j++;

}

else if(isoperator(item) == 1)

{

// pop all higher precendence operator and add them to postfix expresion

x=pop();

while(isoperator(x) == 1 && precedence(x)>= precedence(item))

{

prefix\_exp[j] = x;

j++;

x = pop();

}

// push the last pop oprerator symbol onto stack

push(x);

// push current oprerator symbol onto stack

push(item);

}

// if current symbol is ')' then pop and keep popping until '(' encounterd

else if(item == ')')

{

x = pop();

while(x != '(')

{

prefix\_exp[j] = x;

j++;

x = pop();

}

}

else

{

// if current symbol is neither operand not '(' nor ')' and nor operator

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

break;

}

i++;

// Go to next symbol of infix expression

item = infix\_exp[i];

} //End while loop

if(top > 0)

printf("\n Invalid infix Expression.");

prefix\_exp[j] = '\0';

// Reverse the prefix expression.

pos=0;

len=strlen(prefix\_exp);

for(k=len-1;k>=0;k--)

{

rev[pos]=prefix\_exp[k];

pos++;

}

rev[pos]='\0';

strcpy(prefix\_exp,rev);

}

// Define push operation

void push(char c)

{

if(top >= SIZE-1)

printf("\n Stack Overflow.");

else

{

top++;

stack[top] = c;

}

}

// Define pop operation

char pop()

{

char c;

c='\0';

if(top < 0)

printf("\n Stack Underflow.");

else

{

c = stack[top];

top--;

}

return c;

}

// Define function that is used to determine whether any symbol is operator or not

int isoperator(char symbol)

{

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

return 1;

else

return 0;

}

// Define fucntion that is used to assign precendence to operator.

// In this fucntion we assume that higher integer value means higher precendence

int precedence(char symbol)

{

if(symbol == '^')

return(5);

else if(symbol == '/')

return(4);

else if(symbol == '\*')

return(3);

else if(symbol == '+')

return(2);

else if(symbol == '-')

return(1);

else

return(0);

}

**CONCLUSION:**

The simulation of the above C program helped me understand the conversion of infix to pre-fix expression using STACK.

**OUTPUT:**

Enter Infix expression : (A+B\*C)

Prefix Expression: +A\*BC