**Program 8 :**

**Given a String representing a parentheses-free infix arithmetic expression, implement a program to place it in a tree in the infix form. Assume that a variable name is a single letter. Traverse the tree to produce an equivalent postfix and prefix expression string.**

**Program :**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct node

{

int data;

struct node \*lc;

struct node \*rc;

};

typedef struct node \*NODE;

struct stack {

NODE items[100];

int top;

};

struct node \*GetNode()

{

struct node \*p;

p = (NODE) malloc(sizeof(struct node));

if(p== NULL)

{

printf("Insufficient Memory");

}

p->lc = NULL;

p->rc=NULL;

return p;

}

void push( struct stack \*s , NODE p)

{

s->top++;

s->items[s->top]=p;

}

NODE pop( struct stack \*s )

{

NODE p;

p=s->items[s->top];

s->top--;

return p;

}

int icp(char x)

{

switch(x)

{

case '+':

case '-': return 1;

case '/':

case '\*': return 3;

case '^': return 6;

}

}

int isp(char x)

{

switch(x)

{

case '+':

case '-': return 2;

case '/':

case '\*': return 4;

case '^': return 5;

}

}

NODE CreateExpTree( char a[])

{

struct stack opers,opnds;

opers.top=-1;

opnds.top=-1;

int i=0;

char symb;

NODE p,q,r,r1,r2;

while(a[i] != '\0')

{

symb =a[i++];

printf("%c",symb);

p = GetNode(symb);

p->data =symb;

if((symb >='a' && symb <='z' ) || (symb >='A' && symb <='Z' ))

push(&opnds, p);

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

{

while( opers.top != -1 )

{

q= opers.items[opers.top];

if(isp(q->data) >= icp(symb))

{

q= pop(&opers);

r2=pop(&opnds);

r1=pop(&opnds);

q->lc=r1;

q->rc=r2;

push(&opnds,q);

}

else

break;

}

push(&opers, p);

}

}

while( opers.top != -1 )

{

q= pop(&opers);

r2=pop(&opnds);

r1=pop(&opnds);

q->lc=r1;

q->rc=r2;

push(&opnds,q);

}

return (pop(&opnds));

}

void pretrav(NODE p)

{

if(p != NULL)

{

printf("%c", p->data);

pretrav(p->lc);

pretrav(p->rc);

}

}

void intrav(NODE p)

{

if(p != NULL)

{

intrav(p->lc);

printf("%c", p->data);

intrav(p->rc);

}

}

void postrav(NODE p)

{

if(p != NULL)

{

postrav(p->lc);

postrav(p->rc);

printf("%c", p->data);

}

}

int main()

{

NODE tree = NULL;

char ch;

int i=0;

char a[50];

printf("Enter infix expression\n");

scanf("%s",a);

tree = CreateExpTree( a);

printf("\nInfix Expression\n");

intrav(tree);

printf("\npostfix Expression\n");

postrav(tree);

printf("\nprefix Expression\n");

pretrav(tree);

}

**OUTPUTS :**

**OUTPUT 1 :**

Enter infix expression

a-b-c+d/e^f^g/h+i-j\*k

a-b-c+d/e^f^g/h+i-j\*k

Infix Expression

a-b-c+d/e^f^g/h+i-j\*k

postfix Expression

ab-c-defg^^/h/+i+jk\*-

prefix Expression

-++--abc//d^e^fghi\*jk

**OUTPUT 2 :**

Enter infix expression

a/b-c\*d^e+f/g/h\*i

a/b-c\*d^e+f/g/h\*i

Infix Expression

a/b-c\*d^e+f/g/h\*i

postfix Expression

ab/cde^\*-fg/h/i\*+

prefix Expression

+-/ab\*c^de\*//fghi

**OUTPUT 3 :**

Enter infix expression

a+b-c/d+e^f-g\*h

a+b-c/d+e^f-g\*h

Infix Expression

a+b-c/d+e^f-g\*h

postfix Expression

ab+cd/-ef^+gh\*-

prefix Expression

-+-+ab/cd^ef\*gh