```
#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

Struct node

{

Char info;

Struct node *left;

Struct node *right;

};

Typedef struct node *NODE;

Struct stack

{

Int top;

NODE data[10];

};

Typedef struct stack STACK;

Int preced(char item){

Switch(item){

Case '^': return 5;

Case '*':

Case '/': return 3;

Case '+':

Case '-': return 1;

}

}

Void preorder(NODE root){

If(root != NULL){
```

```
Printf("%c\t", root->info);
Preorder(root->left);
Preorder(root->right);
}
}
Void inorder(NODE root){
If(root != NULL){
Inorder(root->left);
Printf("%c\t", root->info);
Inorder(root->right);
}
}
Void postorder(NODE root){
If(root != NULL){
Postorder(root->left);
Postorder(root->right);
Printf("%c\t", root->info);
}
}
Void push(STACK *s, NODE temp){
s->data[++(s->top)] = temp;
}
NODE pop(STACK *s){
Return (s->data[(s->top)--]);
}
NODE createnode(char item)
{
NODE temp;
Temp = (NODE)malloc(sizeof(struct node));
```

```
Temp->info = item;

Temp->left = NULL;

Temp->right = NULL;

Return temp;

}

NODE createExpTree(char expr[20])

{

STACK tree, operator;

Tree.top = -1;

Operator.top = -1;

Char symbol;

Int I;

NODE temp, t, l, r;

For (i=0; expr[i] != ‘\0’; i++)

{

Symbol = expr[i];

Temp = createnode(symbol);

If(isalnum(symbol))

Push(&tree, temp);

Else{

If(operator.top == -1)

Push(&operator, temp);

Else{

While(operator.top != -1 && preced((operator.data[operator.top])->info) >=

Preced(symbol))

{

T = pop(&operator);

R = pop(&tree);

L = pop(&tree);
t->right = r;
t->left = l;
push(&tree, t);
}
Push(&operator, temp);
}
}
}
While(operator.top != -1){
T = pop(&operator);
R = pop(&tree);
L = pop(&tree);
t->right = r;
t->left = l;
push(&tree, t);
}
Return pop(&tree);
}
Int main()
{
NODE root = NULL;
Char expr[20];
Printf("Read expression\n");
Scanf("%s", expr);
Root = createExpTree(expr);
Printf("\nInorder::");
Inorder(root);
Printf("\nPreorder:");
Preorder(root);
```

```
Printf("\nPostorder:");

Postorder(root);

Return 0;

}
```

Output:

```
Read expression

a+d-c*b


Inorder::a      +       d       -       c       *       b

Preorder:-      +       a       d       *       c       b

Postorder:a     d       +       c       b       *       -
```