Name-Harshita Totala Roll No-SEITA14

Div-A

Assignment-4
Expression Tree

AIM: Construct and expression tree from postfix/prefix expression and perform recursive and non- recursive In-order, pre-order and post-order traversals.

OBJECTIVE:

1. Understand the concept of expression tree and binary tree.
2. Understand the different type of traversals (recursive & non-recursive).

THEORY:

1. **Definition of an expression tree with diagram.**

Algebraic expressions such as a/b + (c-d) e

The terminal nodes (leaves) of an expression tree are the variables or constants in the expression (a, b, c, d, and e). The non-terminal nodes of an expression tree are the operators (+, -, , and). Notice that the parentheses which appear in Equation do not appear in the tree. Nevertheless, the tree representation has captured the intent of the parentheses since the subtraction is lower in the tree than the multiplication.

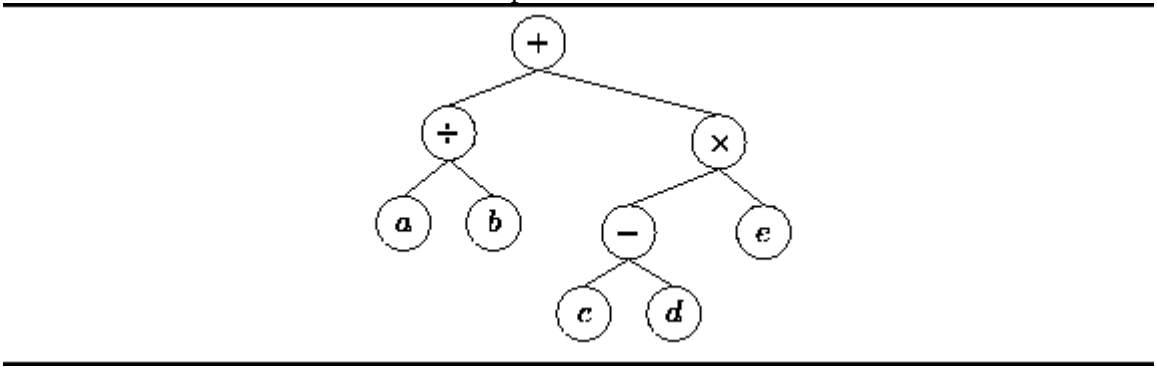


Figure: Tree representing the expression a/b+(c-d)e.

2. **Show the different type of traversals with example**

To traverse a non-empty binary tree in **preorder**,

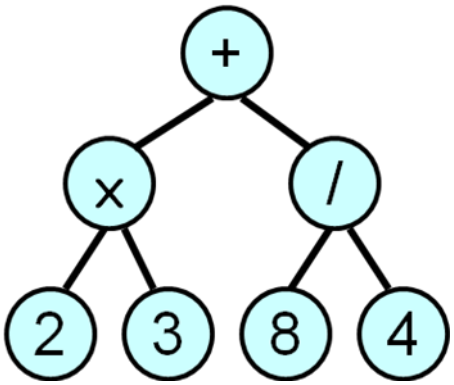
1. Visit the root.
2. Traverse the left subtree.
3. Traverse the right subtree.

To traverse a non-empty binary tree in **inorder**:

1. Traverse the left subtree.
2. Visit the root.
3. Traverse the right subtree.

To traverse a non-empty binary tree in **postorder**,

1. Traverse the left subtree.
2. Traverse the right subtree.
3. Visit the root.



- **Pre-order (prefix)**
  + □ 2 3 / 8 4
- **In-order (infix)**
  2 □ 3 + 8 / 4
- **Post-order (postfix)**
  2 3 □ 8 4 / +

ALGORITHM:

Define structure for Binary Tree (Information, Left Pointer & Right Pointer) **Create Expression Tree:**

CreateTree()

Root& Node pointer variable of type structure. Stack is an pointer array of type structure. String is character array which contains postfix expression. Top & I are variables of type integer.

Step 1: Top = -1 , I = 0;

Step 2: Do Steps 3,4,5,6 While String[I] != NULL Step 3: Create Node of type of structure

Step 4:

Node>Data=String[I]

; Step 5: If
isalnum(String[I])
Then
Stack[Top++] =
Node; Else
Node->Right = Stack
[--Top]; Node->Left =
Stack[--Top]; Stack[
Top++] =
Node; Step 6: Increment
I;
Step 7: Root =
Stack[0]; Step
8: Return Root

Inorder Traversal Recursive : Tree
is pointer of type structure.
InorderR(Tree)
Step 1: If Tree != NULL
Step 2: InorderR(Tree->Left
) Step 3:
Print Tree->Data
Step 4: InorderR(Tree->Right)

Postorder Traversal
Recursive: Tree is
pointer of type
structure.
PostorderR(Tree)
Step 1: If Tree != NULL Step
2: PostorderR(Tree>Left)
Step 3:
PostorderR(Tree->Right)
Step 4: Print Tree->Data

Preorder Traversal Recursive:
Tree is pointer of type
structure.
PreorderR(Tree)
Step 1: If Tree

!= NULL Step
2: Print Tree-
>Data Step 3:
PreorderR(Tree>Left)
Step 4:
PreorderR(Tree->Right
)

Postorder Traversal Nonrecursive :

NonR_Postorder(Tree)
Tree, Temp is pointer of type structure. Stack is pointer array of type structure.Top variable of type integer.
Step 1: Temp = Tree // current pointer pointing to root
Step 2: if Temp != NULL then push current pointer along with its initial flag value _L‘ on to the stack and traverse on the left side.
Step 3: Otherwise if the stack is not empty then pop an address from stack along with its flag value.
Step 4: For the current pointer if the flag value is _L‘ then change it to _R‘ and push current pointer along with its flag value _R‘ on to the stack and traverse on the right side.
Step 5: otherwise, For the current pointer if the flag value is _R‘ then display the element and make the current pointer NULL (i.e.temp=NULL) Step 6: repeat steps 2, 3, 4, 5 until the stack becomes empty.

Preorder Traversal Nonrecursive :

NonR_Preorder(Tree)
Tree, Temp is pointer of type structure. Stack is pointer array of type structure. Top variable of type integer.
Step 1: Temp = Tree
Step 2: Do Steps3,4,5,6,7,& 8 While Temp != NULL And Stack is not
Empty Step 3: Do Steps 4,5&6 While Temp != NULL
Step 4: Print Temp->Data
Step 5: Stack[++ Top] = Temp //Push
Element Step 6: Temp = Temp->Left
Step 7: Temp = Stack [Top --] //Pop
Element Step 8: Temp = Temp->Right

Inorder Traversal Nonrecursive :

NonR_Inorder(Tree)
Tree, Temp is pointer of type structure. Stack is pointer array of type structure. Top variable of type integer.
Step 1: Temp = Tree
Step 2: Do Steps3,4,5,6,7,&8 While Temp != NULL And Stack is not
Empty Step 3: Do Steps 4,5 While Temp != NULL
Step 4: Stack[++ Top] = Temp;//Push

Element Step 5: Temp = Temp->Left
Step 6: Temp = Stack[Top --]//Pop
Element Step 7: Print Temp->Data
Step 8: Temp = Temp->Right

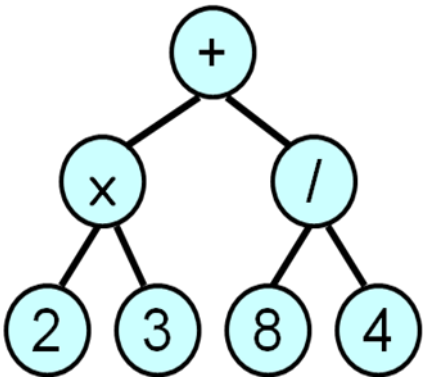
INPUT:

Postfix Expression: **2 3 □ 8 4 / +**

OUTPUT:

Display result of each operation with error checking.

Expression tree



Name-Harshita Totala Roll No-SEITA14

Div-A

Program-

```
#include<iostream>
using namespace std;

typedef struct node                //node of tree
{
    char data;
    struct node *left;
    struct node *right;

}node;

typedef struct stacknode
{
    node* data;
    struct stacknode *next;
}stacknode;

class stack
{
    stacknode *top;
    public:
    stack()
    {
        top=NULL;
    }
    node* topp()
    {
        return (top->data);
    }
    int isempty()
    {
        if(top==NULL)
            return 1;

        else
          return 0;
    }

    void push(node* a)
    {
        stacknode *p;
```

```
        p=new stacknode();
        p->data=a;
        p->next=top;
        top=p;
    }

    node* pop()
    {
        stacknode *p;
        node* x;
        x=top->data;
        p=top;
        top=top->next;

        return x;
    }
};

node* create_pre(char prefix[10]);
node* create_post(char postfix[10]);
void inorder_non_recursive(node *t);
void inorder(node *p);
void preorder(node *p);
void postorder(node *p);
void preorder_non_recursive(node *t);
void postorder_non_recursion(node *t);

node* create_post(char postfix[10])
{
    node *p;
    stack s;
    for(int i=0;postfix[i]!='\0';i++)
    {
        char token=postfix[i];
        if(isalnum(token))
        {
            p=new node();
            p->data=token;
            p->left=NULL;
            p->right=NULL;
            s.push(p);
        }
        else
        {
            p=new node();
```

```
            p->data=token;
            p->right=s.pop();
            p->left=s.pop();
            s.push(p);
        }
    }
    return s.pop();


}

node* create_pre(char prefix[10])
{node *p;
stack s;
int i,j;
        for(i=0;prefix[i]!='\0';i++);
        i--;

    for( j=0;i>=0;j++,i--)
    {
        char token=prefix[i];
        if(isalnum(token))
        {
            p=new node();
            p->data=token;
            p->left=NULL;
            p->right=NULL;
            s.push(p);
        }
        else
        {
            p=new node();
            p->data=token;
            p->left=s.pop();
            p->right=s.pop();
            s.push(p);
        }
    }
    return s.pop();


}

int main()
{
```

```
    node *r=NULL;
    node *r1;
    char postfix[10],prefix[10];
    int x;
    int ch,choice;
do
{
    cout<<"\n---TREE OPERATIONS---\n1.Construct tree from postfix
expression/prefix expression\n2.Inorder traversal"
    "\n3.Preorder traversal\n4.Postorder traversal\n5.Exit\nEnter your
choice=";
    cin>>ch;
    switch(ch)
    {
        case 1:cout<<"Enter choice:\n1.Postfix expression\n2.Prefix
expression\n";
            cin>>choice;
            if(choice==1)
            {
                cout<<"\nEnter postfix expression=";
                cin>>postfix;
                r=create_post(postfix);
            }
            else
            {
                cout<<"\nEnter prefix expression=";
                cin>>prefix;
                r=create_pre(prefix);
            }
            cout<<"\n\nTree created successfully";
            break;


        case 2:cout<<"\nInorder Traversal of tree:\n";
                inorder(r);
                cout<<"\n Without recursion:\t";
                inorder_non_recursive(r);
                break;
        case 3:cout<<"\nPreorder Traversal of tree:\n";
                preorder(r);
                cout<<"\npreorder traversal without recursion:\t";
                preorder_non_recursive(r);
                break;
        case 4:cout<<"\nPostorder Traversal of tree:\n";
                postorder(r);
```

```
                cout<<"\npostorder traversal without recursion";
                postorder_non_recursion(r);
                break;
    }
}while(ch!=5);
    return 0;
}

void inorder(node *p)
{
        if(p!=NULL)
        {
            inorder(p->left);
            cout<<p->data;
            inorder(p->right);
        }
}

void preorder(node *p)
{
        if(p!=NULL)
        {
            cout<<p->data;
            preorder(p->left);
            preorder(p->right);
        }
}


void postorder(node *p)
{
        if(p!=NULL)
        {
            postorder(p->left);
            postorder(p->right);
            cout<<p->data;
        }
}




void inorder_non_recursive(node *t)
{
```

```
    stack s;
    while(t!=NULL)
    {
        s.push(t);
        t=t->left;
    }

    while(s.isempty()!=1)
    {
        t=s.pop();
        cout<<t->data;
        t=t->right;
        while(t!=NULL)
    {
        s.push(t);
        t=t->left;
    }


    }


}



void preorder_non_recursive(node *t)
{

    stack s;
    while(t!=NULL)
    {
        cout<<t->data;
        s.push(t);
        t=t->left;
    }

    while(s.isempty()!=1)
    {
        t=s.pop();

        t=t->right;
        while(t!=NULL)
    {
        cout<<t->data;
```

```
        s.push(t);
        t=t->left;
    }


    }
}

void postorder_non_recursion(node *t)
{
    stack s,s1;
node *t1;
    while(t!=NULL)
    {
        s.push(t);
        s1.push(NULL);
        t=t->left;
    }
    while(s.isempty()!=1)
    {
        t=s.pop();
        t1=s1.pop();
        if(t1==NULL)
        {
            s.push(t);
            s1.push((node *)1);
            t=t->right;
                while(t!=NULL)
                {
                    s.push(t);
                    s1.push(NULL);
                    t=t->left;
                }

        }
        else
        cout<<t->data;
    }


}
```

Name-Harshita Totala Roll No-SEITA14

Div-A

Output-

```
---TREE OPERATIONS---
1.Construct tree from postfix expression/prefix expression
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice=1
Enter choice:
1.Postfix expression
2.Prefix expression
1

Enter postfix expression=52-82/+1-


Tree created successfully
---TREE OPERATIONS---
1.Construct tree from postfix expression/prefix expression
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice=2

Inorder Traversal of tree:
5-2+8/2-1
 Without recursion:     5-2+8/2-1
---TREE OPERATIONS---
1.Construct tree from postfix expression/prefix expression
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice=3

Preorder Traversal of tree:
-+-52/821
preorder traversal without recursion:   -+-52/821
---TREE OPERATIONS---
1.Construct tree from postfix expression/prefix expression
```

2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice=4

Postorder Traversal of tree:
52-82/+1-
postorder traversal without recursion52-82/+1-
---TREE OPERATIONS---
1.Construct tree from postfix expression/prefix expression
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice=5

Conclusion: Thus we have Constructed an expression tree from postfix/prefix expression and performed recursive and non- recursive In-order, pre-