MIT Art Design and Technology University

MIT School of Computing, Pune

Department of Computer Science and Engineering

Second Year B. Tech

Academic Year 2022-2023. (SEM-II)

Subject: Advance Data Structures Laboratory

Assignment 1

Assignment Title: Accept prefix expressions, and construct a binary tree and perform recursive and non-recursive traversals.

Aim: Implement Binary tree traversals.

Prerequisite:
1. Basic tree terminology and concepts
2. Tree traversals
3. Concept of Prefix/Postfix (Polish Notations)

Objectives:
1. To understand how Binary Tree (BT) is used to represent prefix expression in hierarchical manner
2. Understand the different type of traversals (recursive & non-recursive).

Outcomes:
Upon Completion of the assignment the students will be able to
1. Construct a BT from prefix expression.
2. Understand and analyse various recursive and non-recursive traversals of the BT

Theory:
Binary Tree
- A binary tree is a data structure which is defined as a collection of elements called nodes. Every node contains a "left" pointer, a "right" pointer, and a data element. Every binary tree has a root element pointed by a "root" pointer. The root element is the topmost node in the tree. If root = NULL, then it means the tree is empty.

- If the root node R is not NULL, then the two trees T1 and T2 are called the left and right subtrees of R. if T1 is non-empty, then T1 is said to be the left successor of R. likewise, if T2 is non-empty then, it is called the right successor of R (Ref. Figure1)

- In a binary tree every node has 0, 1 or at the most 2 successors. A node that has no successors or 0 successors is called the leaf node or the terminal node.

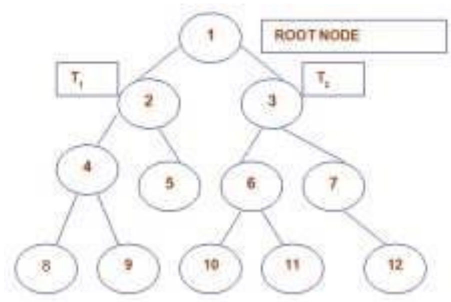


Figure1: Binary Tree

Expression Tree:

- Binary trees are widely used to store algebraic expressions. For example, consider the algebraic expression Exp given as,
- Exp = (2 x 3) + (8 / 4)
- This expression can be represented using a binary tree as shown in figure

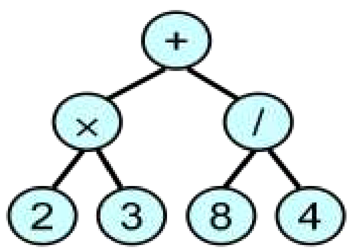


Figure1: Expression Tree

Construction of expression tree by using prefix expression

- Scan the given prefix expression from **Right to Left.**
- If the character encountered is an **operand,** then
  - Create a new a node; set left and right pointers of the newly created node as NULL
  - Push the address of newly created node to stack.
- If character encountered is an **operator,** then
  - Create a new a node; Pop 2 values (address of the nodes already pushed on to the stack) from stack
  - 1^{st} popped node is attached as **left** child
  - 2^{nd} popped node is attached as **right** child.
  - Push the address of the newly created node back on to stack.
- Repeat the above process till expression is scanned and stack is empty

Traversing of a Binary Tree (Expression Tree)

- Traversing a binary tree is the process of visiting each node in the tree exactly once, in a systematic way. Unlike linear data structures in which the elements are traversed sequentially, tree is a non-linear data structure in which the elements can be traversed in many different ways. There are different algorithms for tree traversals. These algorithms differ in the order in which the nodes are visited.

- **Pre-order recursive algorithm**

  To traverse a non-empty binary tree in preorder, the following operations are performed recursively at each node. The algorithm starts with the root node of the tree and continues by,
  - Visiting the root node.
  - Traversing the left subtree.
  - Traversing the right subtree.

  Pre-order Traversal of the Expression Tree shown in Figure 1 is: + x 2 3 / 8 4
- **In-order recursive algorithm**

  To traverse a non-empty binary tree in in-order, the following operations are performed recursively at each node. The algorithm starts with the root node of the tree and continues by,
  - Traversing the left subtree.
  - Visiting the root node.
  - Traversing the right subtree.

- In-order Traversal of the Expression Tree shown in Figure 1 is: 2 x 3 + 8 / 4

- **Post-order recursive algorithm**

  To traverse a non-empty binary tree in post-order, the following operations are performed recursively at each node. The algorithm starts with the root node of the tree and continues by,
  - Traversing the left subtree.
  - Traversing the right subtree.
  - Visiting the root node.

  Post-order Traversal of the Expression Tree shown in Figure 1 is: 2 3 x 8 4 / +

- **Conclusion:**
  - The expression tree when traversed in the Pre-order manner then the output is a prefix expression
  - The expression tree when traversed in the In-order manner then the output is a infix expression
  - The expression tree when traversed in the Post-order manner then the output is a postfix expression

Questions:

1. Construct an Expression Tree from following prefix expression + 3 * + 5 9 2 (step by step output expected)
2. Write a non-recursive Pre-order, In-order, Post-order traversal algorithm in a Binary Tree

MIT SCHOOL OF ENGINEERING
Rajbaug, Loni-Kalbhor, Pune

MIT-ADT UNIVERSITY

Name : Sourav Shailesh Toshniwal
Class : SY CSE-1 (B)
Roll No. : 2213047
Subject : Advance Data Structures Laboratory.

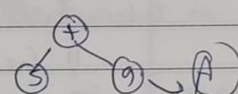
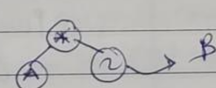
Assignment - 1.

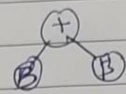
<u>Title</u> : Accept prefix expressions, and construct a binary tree and perform recursive and non-recursive traversals.

<u>Questions</u> :

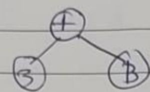
1. Construct an Expression Tree from following prefix +3*+592 (step by step output expected).

<u>Ans</u>

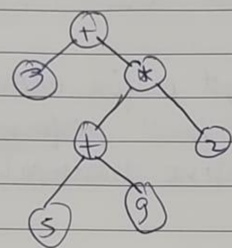
Stack	expression	expression tree
2	95+*3+	
9, 2	5+*3+	
5, 9, 2	+*3+	
2	*3+	
A, 2	*3+	
	3+	
B, 3	+	



∴ Expression tree is :



i.e.



2. Write a non-recursive Pre-order, In-order, Post-order traversal algorithm in a Binary Tree.

A) Algorithms.

a. Preorder non-recursive.

1) Create an empty stack nodestack and push root node to stack.
2) Do following while nodestack is not empty.
   a) Pop an item from stack and print it.
   b) Push right child of popped item to stack
   c) Push left child of popped item to stack.

b. Inorder non-recursive.

1) Create an empty stack S.
2) Initialize current node as root.
3) Push the current node to S and set current = current → left untill current is NULL.

4) If current is NULL and stack is not empty then
   a) Pop the top item from stack.
   b) Print the popped item, set current = popped - item → right.
   c) Go to step 3.
5) If current is NULL and stack is empty then we are done

C. Postorder non-recursive.
1) Create an empty stack.
2) Do following while root is not NULL.
   a) Push root's right child and then root to stack
   b) Set root as roots left child.
ii) Pop an item from stack and set it as root.
   a) If the popped item has a right child and the right child is at top of stack, then remove the right child from stack, push the root back and set root as root's right child.
   - Repeat i) and ii) while stack is not empty.

Conclusion :- The expression tree when traversed in the Pre-order manner then the output is a prefix expression.
- The expression tree when transvered in the In-order manner then the output is infix expression.
- The expression tree when transvered in the Post-order manner then the output is a postfix expression.

Code:

```
/*
    Construct a expression tree from postfix expression
    ====================================================
    Functions:
        1.Create
        2.Inorder Traversal (Recursive)
        3.Preorder Traversal (Recursive)
        4.Postorder Traversal (Recursive)
        5.Inorder Traversal(N.Recursive)
        6.Preorder Traversal(N.Recursive)
        7.Postorder Traversal(N.Recursive)
________________________________________________________
*/
#include<iostream>
#include<stdlib.h>
using namespace std;
#define MAX 30
int isoperand(char ch)
{
    if((ch>='A'  && ch<='Z')||(ch>='a'  && ch<='z')||(ch>='0'  && ch<='9'))
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
int isoperator(char ch)
{
    if(ch=='$'||ch=='^'||ch=='+'||ch=='-'||ch=='*'||ch=='/')
        return 1;
    else
        return 0;
}
struct Treenode
{
    Treenode *lchild;
    char data;
    Treenode *rchild;
};
class ET
{
    Treenode *root;
    public:
        ET();
        void create(char postfix[MAX]);
        void inorder();
```

```
        void inorder(Treenode *);
        void preorder();
        void preorder(Treenode *);
        void postorder();
        void postorder(Treenode *);
        void inorder_nrc();
        void preorder_nrc();
        void postorder_nrc();
};
ET::ET()
{
    root=NULL;
}
void ET::create(char prefix[MAX])
{
    Treenode *stack[MAX];
    int top=-1;
    int i,len,val;
    char ch;
    Treenode *temp;
    for(i=0;prefix[i]!='\0';i++);
    len=i-1;
    for(i=len;i>=0;i--)
    {
        ch=prefix[i];
        temp=new Treenode;
        temp->lchild=NULL;
        temp->data=ch;
        temp->rchild=NULL;
        if(isoperand(ch))
        {
            stack[++top]=temp;
        }
        else if(isoperator(ch))
        {
            temp->lchild=stack[top--];
            temp->rchild=stack[top--];
            stack[++top]=temp;
        }
        else
        {
            cout<<"\nWrong expression tree";
            cout<<"\nNode cannot be created";
            exit(0);
        }
    }
    root=stack[top--];
}
void ET::inorder()
```

```
{
    if(root)
        inorder(root);
    else
        cout<<"\nEmpty expression tree";
}
void ET::inorder(Treenode *root)
{
    if(root)
    {
        inorder(root->lchild);
        cout<<root->data<<" ";
        inorder(root->rchild);
    }
}
void ET::preorder()
{
    if(root)
        preorder(root);
    else
        cout<<"\nEmpty expression tree";
}
void ET::preorder(Treenode *root)
{
    if(root)
    {
        cout<<root->data<<" ";
        preorder(root->lchild);
        preorder(root->rchild);
    }
}
void ET::postorder()
{
    if(root)
        postorder(root);
    else
        cout<<"\nEmpty expression tree";
}
void ET::postorder(Treenode *root)
{
    if(root)
    {
        postorder(root->lchild);
        postorder(root->rchild);
        cout<<root->data<<" ";
    }
}
void ET::inorder_nrc()
{
```

```
    Treenode *curr=root;
    Treenode *stack[MAX];
    int top=-1;
    while(1)
    {
        while(curr!=NULL)
        {
            stack[++top]=curr;
            curr=curr->lchild;
        }
        if(top!=-1)
        {
            curr=stack[top--];
            cout<<curr->data<<" ";
            curr=curr->rchild;
        }
        else
            break;
    }
}
void ET::preorder_nrc()
{
    Treenode *curr=root;
    Treenode *stack[MAX];
    int top=-1;
    while(1)
    {
        while(curr!=NULL)
        {
            cout<<curr->data<<" ";
            stack[++top]=curr;
            curr=curr->lchild;
        }
        if(top!=-1)
        {
            curr=stack[top--];


            curr=curr->rchild;
        }
        else
            break;
    }
}
void ET::postorder_nrc()
{
    Treenode *curr=root;
    Treenode *stack[MAX];
    int top=-1,flag[MAX],f;
```

```
    while(1)
    {
        if(curr!=NULL)
        {
            stack[++top]=curr;
            flag[top]=0;
            curr=curr->lchild;
        }
        else
        {
                if(top!=-1)
                {
                    f=flag[top];
                    curr=stack[top--];
                    if(f==0)
                    {
                        stack[++top]=curr;
                        flag[top]=1;
                        curr=curr->rchild;
                    }
                    else if (f==1)
                    {
                        cout<<curr->data<<" ";
                        curr=NULL;
                    }
                }
                else
                    break;
        }


    }
}
int main()
{
    int ch;
    char prefix[MAX];
    ET e;
    cout<<"\nEnter a prefix expression";
    cin>>prefix;
    while(1)
    {
        cout<<"\n*********MENU*********";
        cout<<"\n1.Create a expression tree\n2.Inorder Traversal
(Recursive)\n3.Preorder Traversal (Recursive)";
        cout<<"\n4.Postorder (Recursive)\n5.Inorder Traversal(Non
Recursive)\n6.Preorder Traversal(Non Recursive)";
        cout<<"\n7.Post order Traversal(Non Recursive)\n8.Exit";
        cout<<"\nEnter your choice";
```

```
        cin>>ch;
        switch(ch)
        {
            case 1:
                e.create(prefix);
                cout<<"\nExpression Tree Created from Prefix Expression\n";
                break;
            case 2:
                e.inorder();
                break;
            case 3:
                e.preorder();
                break;
            case 4:
                e.postorder();
                break;
            case 5:
                e.inorder_nrc();
                break;
            case 6:
                e.preorder_nrc();
                break;
            case 7:
                e.postorder_nrc();
                break;
            case 8:
                exit(0);
            default:
                break;
        }
    }
}
```

Output:

```
PS C:\SOURAV\CODE\C++ language codes\ADS assignment> cd "c:\SOURAV\CODE\C++ language codes\ADS assignment\" ; if ($?)

Enter a prefix expression++234/54

*********MENU*********
1.Create a expression tree
2.Inorder Traversal (Recursive)
3.Preorder Traversal (Recursive)
4.Postorder (Recursive)
5.Inorder Traversal(Non Recursive)
6.Preorder Traversal(Non Recursive)
7.Post order Traversal(Non Recursive)
8.Exit
Enter your choice

1

Expression Tree Created from Prefix Expression

*********MENU*********
1.Create a expression tree
2.Inorder Traversal (Recursive)
3.Preorder Traversal (Recursive)
4.Postorder (Recursive)
5.Inorder Traversal(Non Recursive)
6.Preorder Traversal(Non Recursive)
7.Post order Traversal(Non Recursive)
8.Exit
Enter your choice2
2 + 3 + 4
*********MENU*********
1.Create a expression tree
2.Inorder Traversal (Recursive)
3.Preorder Traversal (Recursive)
4.Postorder (Recursive)
5.Inorder Traversal(Non Recursive)
6.Preorder Traversal(Non Recursive)
7.Post order Traversal(Non Recursive)
8.Exit
Enter your choice3
+ + 2 3 4
*********MENU*********
1.Create a expression tree
2.Inorder Traversal (Recursive)
3.Preorder Traversal (Recursive)
4.Postorder (Recursive)
5.Inorder Traversal(Non Recursive)
6.Preorder Traversal(Non Recursive)
7.Post order Traversal(Non Recursive)
8.Exit
2 3 + 4 +
*********MENU*********
1.Create a expression tree
2.Inorder Traversal (Recursive)
3.Preorder Traversal (Recursive)
4.Postorder (Recursive)
5.Inorder Traversal(Non Recursive)
6.Preorder Traversal(Non Recursive)
7.Post order Traversal(Non Recursive)
8.Exit
Enter your choice5
2 + 3 + 4
*********MENU*********
1.Create a expression tree
2.Inorder Traversal (Recursive)
3.Preorder Traversal (Recursive)
4.Postorder (Recursive)
5.Inorder Traversal(Non Recursive)
6.Preorder Traversal(Non Recursive)
7.Post order Traversal(Non Recursive)
8.Exit
Enter your choice6
+ + 2 3 4
*********MENU*********
1.Create a expression tree
2.Inorder Traversal (Recursive)
3.Preorder Traversal (Recursive)
4.Postorder (Recursive)
5.Inorder Traversal(Non Recursive)
6.Preorder Traversal(Non Recursive)
7.Post order Traversal(Non Recursive)
8.Exit
Enter your choice7
2 3 + 4 +
*********MENU*********
1.Create a expression tree
2.Inorder Traversal (Recursive)
3.Preorder Traversal (Recursive)
4.Postorder (Recursive)
5.Inorder Traversal(Non Recursive)
6.Preorder Traversal(Non Recursive)
7.Post order Traversal(Non Recursive)
8.Exit
Enter your choice8
```