ASSIGNMENT NO. 2

**Aim**: For given expression eg. a-b\*c-d/e+f construct inorder sequence and traverseit using inorder, preorder and postorder traversal ( non recursive).

**Objective**: To understand how to construct a binary expression tree and performvarious traversals.

**Outcomes**: Students will be able to convert an expression into a inorder, post ordersequence with the help of binary expression tree.

**Theory:**

A tree is said to be a **Binary tree** if each node of the tree has maximum of two child nodes. The children of the node of binary tree are ordered.

A **binary expression tree** is a specific kind of a binary tree used to represent expressions.

**Binary expression tree as an ADT:**

Defining a Binary expression tree:

Binary expression tree is an ADT which has ordering imposed on the nodes such that for each node:

1. The root node consists of operator values only.
2. All operands are either in the left or the right node

A structure is used for defining a single node of the binary tree as follows: typedef struct node{

char data; node \*left; node \*right;

}node;

Operation Performed on the binary expression tree:

* Create a binary expression tree
* Traversing by inorder, postorder method
* Display the binary expression tree

Functions for performing various operations on a binary tree:

* void exp :: create( )
* void exp :: inorder ( )
* void exp :: postorder ( )
* void exp :: preorder( )

Algorithms for various functions of a binary tree:

1. **Create ( )**

Step 1: Allocate the memory by using new keyword for a new node. Make its left and right node

Pointer NULL.

Step 2: Read the expression from left to right and separate the operand and operator into respective stacks.

Step 3: While storing the operator into operator stack compare the priority of operators. If in coming operator is having higher priority than existing operator on the top of stack, then pop operator from stack store into new node. Pop the operand from the operand stack and store its address into new nodes right , again pop 2 nd operand and store its address into new nodes left.

Now store new node into operand stack .

Step 4: Pop one operator from stack store it into new node, pop two operands, store 1 st operand in new nodes right and 2 nd operand in new nodes

left.

Step 5: Repeat the procedure till operator stack is empty.

1. **Inorder ( )**

Step 1: Declare a variable cn equal to root

Step 2: if cn !=NULL, then push the element(operator/operand) in the stack and shift left. Continue this until cn!=NULL is satisfied.

Step 3: Check if the stack is empty. If the condition is true, then return else pop the topmost element and display it, then move to the right of cn

Step 4: Continue steps 2,3 till stack of operand is empty.

1. **postorder\_nonRecursive( )**

Step 1: Declare two stacks S1 and S2.

Step 2: Initialize cn=root, and push cn->data into S1.

Step 3: till S1 is not empty, pop element from S1 and push into S2 and if S1->left != NULL or S1->right !=NULL push S1->left or S1->right or both into S1. Step 4: Once S1 is empty, then till S2 is not empty pop elements from S2 one by one and display it.

1. **preorder ( )**

step 1: declare a variable cn equal to root.

Step 2: until cn is not NULL, first display the root node then push the node into the stack , afterwards, shift cn=cn->left

Step 3: if the operand stack is empty then return, else pop the topmost element and shift cn to right by cn= cn->right

Step 4: Continue till stack is empty

**Time complexity:**

Time complexity of the various functions of the binary tree are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Sr.N | Function | Time |  |
| o. | create ( ) | Complexity |  |
| 1 | O(nlogn) |  |
| 2 | inorder ( ) | O(n) |  |
| 3 | postorder ( ) | O(n) |  |
| 4 | Preorder( ) | O(n) |  |

**Application and advantages:**

The binary expression tree is used for evaluating a expression and converting it into preorder, postorder format.

The binary expression tree helps to deconstruct a given expression and interpret the result at each step.

**INPUT:**

Accept the expression from user.

**\*Note:** Take one expression and show graphical (draw)step wise expression treecreation. Every one should take different expression.

**OUTPUT:**

Display result of each operation with error checking.

**Conclusion:**

Thus we have successfully implemented binary expression tree and written non-recursive inorder and postorder functions.

**// Sample Oral Questions :**

1. What is Expression tree?
2. What are the members of structure of tree & what is the size of structure?
3. What are rules to construct expression tree?
4. Explain tree traversal.
5. What is preorder, postorder, in order traversal?
6. Which data structure is used for tree traversal?