Name	Sujal Dingankar
UID no.	DSE24100720
Experiment No.	6

AIM:	Create an experssion tree from a given postorder traversal and perform the evalutaion.
THEORY:	**Expression Tree** • An Expression Tree is a binary tree data structure used to represent arithmetic expressions. In an expression tree: • Leaf nodes represent operands (constants or variables). • Internal nodes represent operators (e.g., +, -, *, /). • The tree structure inherently respects the operator precedence and allows for easy evaluation of expressions. **Operations on Expression Tree:** ➢ **Construction**: • Constructed using postfix, prefix, or infix expressions. • Postfix construction involves pushing operands onto a stack and using operators to combine them as nodes. ➢ **In-Order Traversal** (Infix Notation): • Visit the left subtree, root, then right subtree. • Outputs the expression in **infix** form (e.g., a + b). ➢ **Pre-Order Traversal** (Prefix Notation): • Visit the root, left subtree, then right subtree. • Outputs the expression in **prefix** form (e.g., + a b). ➢ **Post-Order Traversal** (Postfix Notation): • Visit the left subtree, right subtree, then root. • Outputs the expression in **postfix** form (e.g., a b +). ➢ **Evaluation**: • Recursively evaluate from root to leaves by applying operators to operand values.

Applications of Expression Tree:

➢ **Compilers and Interpreters:**

- Expression trees are used in compilers to parse and evaluate expressions. They convert arithmetic expressions from source code into an intermediate tree structure, which makes it easier to generate machine code or intermediate code for execution.

➢ **Calculators:**

- Expression trees are used in calculators for evaluating complex arithmetic expressions. They store the expression in a tree form to handle operator precedence and parentheses without ambiguity.

➢ **Symbolic Computation:**

- Expression trees are used in symbolic algebra systems (like Mathematica) to represent and manipulate algebraic expressions. Operations like differentiation and integration can be performed on these trees.

➢ **Code Generation:**

- In programming languages, expression trees are used to represent expressions in Abstract Syntax Trees (ASTs) which are crucial in code generation during compilation.

➢ **Optimization:**

- Compilers use expression trees to perform optimizations like constant folding, where constant sub-expressions are precomputed to reduce runtime overhead.

ALGORITHM:	**1. Traverse the Postfix Expression:** ○ **Loop through each character of the postfix expression.** **2. Check if Character is an Operand:** ○ **If the current character is an operand (like a number or letter), create a tree node with this operand as its data.** ○ **Push this new node onto the stack because operands are leaf nodes.** **3. Check if Character is an Operator:** ○ **If the current character is an operator (+, -, *, /), do the following:** ▪ **Pop the top two nodes from the stack (these are the right and left children of the operator node).** ▪ **Create a new tree node with this operator as its data.** ▪ **Set the two popped nodes as the left and right children of this new operator node.** ▪ **Push the newly created operator node back onto the stack.** **4. End of Loop:** ○ **After going through the entire postfix expression, the stack will contain one node — this is the root of the expression tree.** **5. In-Order Traversal:** ○ **Perform an in-order traversal (left subtree, root, right subtree) of the tree to print the corresponding infix expression.**

PROBLEM SOLVING:

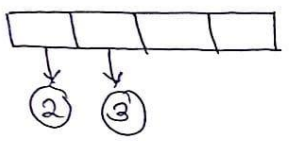
Construction of Expression tree from postorder traversal

Name- Sujal Sandeep Dinganhar

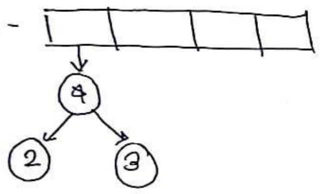
Date - 29-09-2024.

Postfix Expression - 23*54*+9-

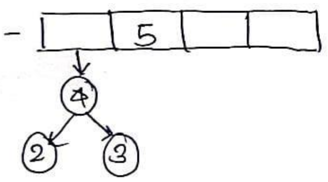
Step 1 -



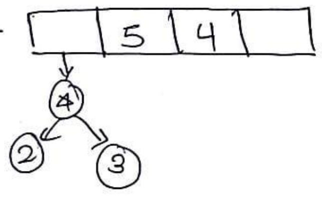
Step 2 -



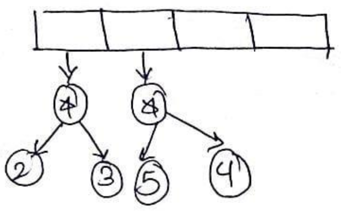
Step 3 -



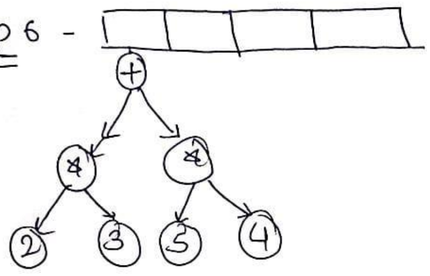
Step 4 -



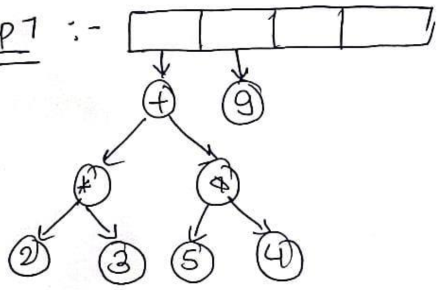
Step 5 -



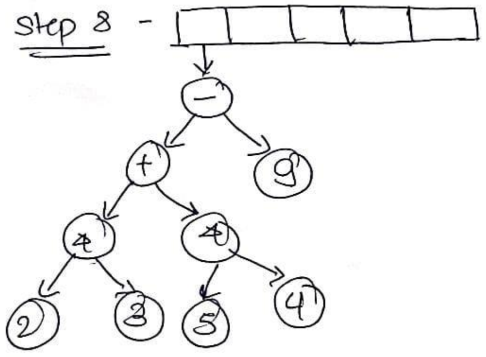
Step 6 -



Step 7 :-

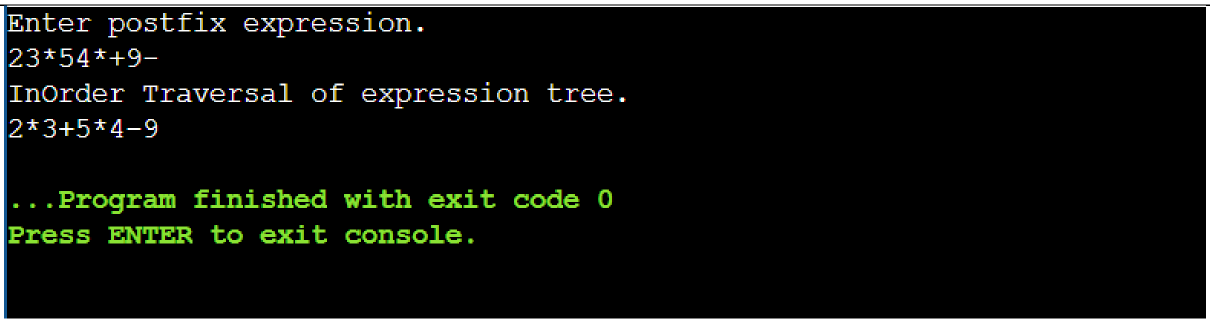


Step 8 -



PROGRAM :-

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

struct Node{
    char data;
    struct Node *left;
    struct Node *right;
};

struct Node *createNode(int data){
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;

    return newNode;
}

struct Node *expressionTreeFromPostfix(char *postfix){
    struct Node *stack[100];
    int top = -1;

    for(int i=0;postfix[i] != '\0';i++){
        if(isalnum(postfix[i])){
            struct Node *newNode = createNode(postfix[i]);
            stack[++top] = newNode;
        }else{
            struct Node *right = stack[top--];
            struct Node *left = stack[top--];
            struct Node *newNode = createNode(postfix[i]);
            newNode->left = left;
            newNode->right = right;
            stack[++top] = newNode;
        }
    }
    return stack[top];
}

void inOrderTraversal(struct Node *root){
    if(root == NULL){
        return;
    }

    inOrderTraversal(root->left);
    printf("%c",root->data);
    inOrderTraversal(root->right);
}

int main(){
    char postfix[100];

    printf("Enter postfix expression.\n");
    scanf("%s",postfix);

    struct Node *root = expressionTreeFromPostfix(postfix);

    printf("InOrder Traversal of expression tree.\n");
    inOrderTraversal(root);

    return 0;
}
```

RESULT :-	

```
Enter postfix expression.
23*54*+9-
InOrder Traversal of expression tree.
2*3+5*4-9

...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion :-

From this experiment with Expression Trees, I learned how to construct and evaluate arithmetic expressions in a tree-based format. Implementing different traversal methods (in-order, pre-order, and post-order) allowed me to represent expressions in various notations (infix, prefix, and postfix). By handling operators and operands as nodes, I gained insight into the efficient evaluation of complex expressions. This experience has enhanced my understanding of tree-based structures, which are crucial for compilers, calculators, and symbolic computation, providing a solid foundation for managing and evaluating expressions dynamically.