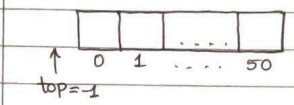
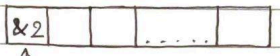
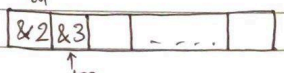
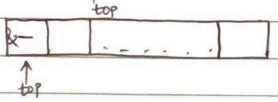
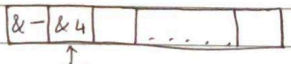
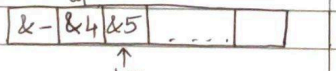
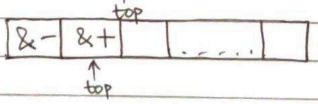
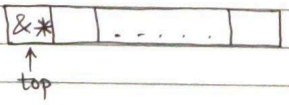
Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.
Department of AI-ML

Experiment	6
Aim	Implement the given problem statement
Objective	Given as input Postfix expression, output its expression tree. Sample input: A B C * + D / Output: Binary Tree computing A + B * C / D according to BODMAS rule.
Name	Balla Mahadev Shrikrishna
UCID	2023300010
Class	A
Batch	A
Date of Submission	20-09-24

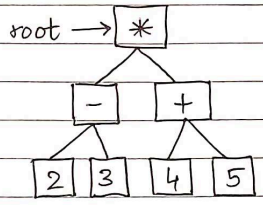
Explanation of the technique used

Postfix Expression : 2 3 - 4 5 + *

postToTree function :

char	stack	working
Initially		
2		2 node is created and pushed into stack
3		3 node is created and pushed into stack
-		right = 3 left = 2 - node is created and pushed to the stack
4		4 node is created pushed into stack
5		5 node is created and pushed into stack
+		right = 5 , left = 4 + is created and pushed into stack
*		right = + , left = - * is created and pushed into stack

stack[top] = * is returned as the root node.



Resultant Expression Tree

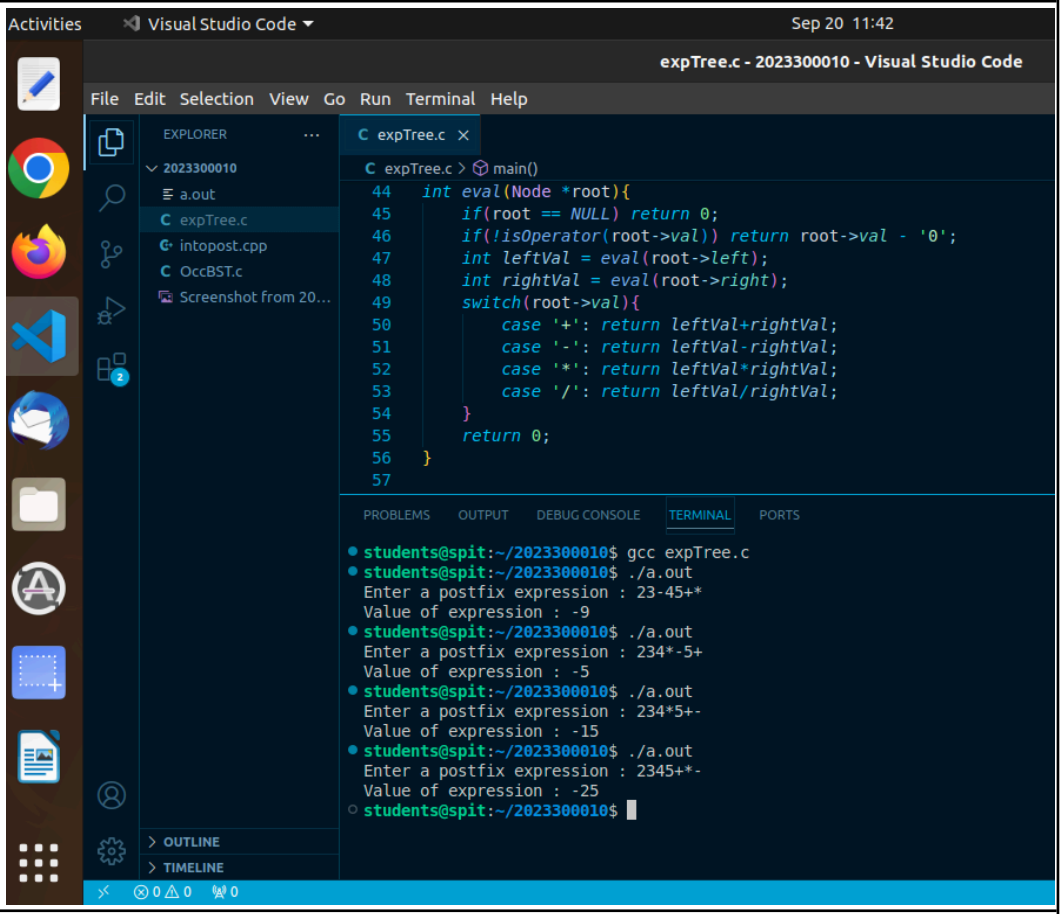
	The postToTree function converts a postfix expression into an expression tree using a stack-based approach i.e accessing only the top element. postToTree function : 1. Initialize stack : A stack is used to hold the nodes of the expression tree. top = -1 initializes the stack to be empty. 2. Iterate through the postfix expression (s): For each character s[i] in the expression: If the character is a digit (isdigit(s[i])): Create a new node using createNode(s[i]). Push this node onto the stack (stack[++top] = node). If the character is an operator (isOperator(s[i])): Pop the top two nodes from the stack (these will be the right and left operands). right = stack[top--] and left = stack[top--]. Create a new node for the operator (createNode(s[i])). Set left and right as the left and right children of the new node. Push this new node back onto the stack. 3. Return the root of the expression tree: After processing the entire expression, the last remaining element on the stack is the root of the constructed expression tree (return stack[top]). eval function : Base Case : If the root is NULL, return 0. Leaf Node (Operand) : If the node is not an operator (it's a digit), convert the character stored in the node to its integer value (root->val - '0') and return it. Internal Node (Operator) : 1. Recursively evaluate the left subtree and store the result in leftVal. 2. Recursively evaluate the right subtree and store the result in rightVal. 3. Based on the operator stored in the current node (root->val), perform the corresponding arithmetic operation using the results from the left and right subtrees (leftVal and rightVal)

Program(Code)	#include <stdio.h>
	#include <stdlib.h>
	#include <string.h>
	#include <ctype.h>
	#include <stdbool.h>
	
	typedef struct Node{
	char val;
	struct Node *left, *right;
	}Node;
	
	Node *createNode(char val){
	Node *new = (Node *)malloc(sizeof(Node));
	new->val = val;
	new->left = NULL;

```
    new->right = NULL;
    return new;
}

bool isOperator(char c){
    return c=='+' || c=='-' || c=='*' || c=='/';
}

Node *postToTree(char* s, int n){
    Node *stack[50], *node, *left, *right;
    int top=-1;
    for(int i=0; i<n; i++){
        if(isdigit(s[i])){
            node = createNode(s[i]);
            stack[++top] = node;
        }
        else if(isOperator(s[i])){
            right = stack[top--];
            left = stack[top--];
            node = createNode(s[i]);
            node->left = left;
            node->right = right;
            stack[++top] = node;
        }
    }
    return stack[top];
}

int eval(Node *root){
    if(root == NULL) return 0;
    if(!isOperator(root->val)) return root->val - '0';
    int leftVal = eval(root->left);
    int rightVal = eval(root->right);
    switch(root->val){
        case '+': return leftVal+rightVal;
        case '-': return leftVal-rightVal;
        case '*': return leftVal*rightVal;
        case '/': return leftVal/rightVal;
    }
    return 0;
}

void freeTree(Node *root){
    if(root != NULL){
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}
```

```
int main(){
    char s[50];
    printf("Enter a postfix expression : ");
    fgets(s, 50, stdin);
    Node *root = postToTree(s,strlen(s));
    printf("Value of expression : %d\n", eval(root));
    freeTree(root);
    return 0;
}
```

Output



Conclusion

This implementation demonstrates an essential application of binary trees in parsing and evaluating arithmetic expressions.