**Department of CSE**

**Compiler Lab (CSE 352)**

**Lab Report 07**

Construction and Post-order Traversal of an Abstract Syntax Tree (AST)

**Submitted By :**

**Fahima Abida Chowdhury**

ID: 0432220005101135

Semester: Spring-2025

Batch: 52(6B1)

**Submitted To :**

[**Md. Ismail**](https://uits.edu.bd/md-ismail/)

Lecturer

Department of CSE

University of Information Technology and Sciences

**Experiment No.: 07**  
**Experiment Name:** Construction and Post-order Traversal of an Abstract Syntax Tree (AST)

### ****Problem Statement****

The objective of this lab is to design and implement a C program that constructs an Abstract Syntax Tree (AST) for a given arithmetic expression and traverses it in post-order. The AST is used to represent the hierarchical structure of the expression and the post-order traversal is typically used for evaluating expressions.

### ****Expression Represented****

The AST is built for the following arithmetic expression:

**a+a\*(b-c)+(b-c)\*d**

**C Code**

**#include<stdio.h>**

**#include<string.h>**

**#include <stdlib.h>**

**struct Node {**

**char value[10];**

**struct Node\* left;**

**struct Node\* right;**

**};**

**struct Node\* createNode(char\* val) {**

**struct Node\* node = (struct Node\*)malloc(sizeof(struct Node));**

**strcpy(node->value, val);**

**node->left = node->right = NULL;**

**return node;**

**}**

**struct Node\* createOpNode(char\* op, struct Node\* left, struct Node\* right) {**

**struct Node\* node = createNode(op);**

**node->left = left;**

**node->right = right;**

**return node;**

**}**

**void printTree(struct Node\* root) {**

**if (root == NULL) return;**

**printTree(root->left);**

**printTree(root->right);**

**printf("%s ", root->value);**

**}**

**int main(){**

**struct Node\* N1 = createNode("a");**

**struct Node\* N2 = createNode("a");**

**struct Node\* N3 = createNode("b");**

**struct Node\* N4 = createNode("c");**

**struct Node\* N5 = createNode("b");**

**struct Node\* N6 = createNode("c");**

**struct Node\* N7 = createNode("d");**

**struct Node\* sub1 = createOpNode("-", N3, N4);**

**struct Node\* mul1 = createOpNode("\*", N2, sub1);**

**struct Node\* add1 = createOpNode("+", N1, mul1);**

**struct Node\* sub2 = createOpNode("-", N5, N6);**

**struct Node\* mul2 = createOpNode("\*", sub2, N7);**

**struct Node\* root = createOpNode("+", add1, mul2);**

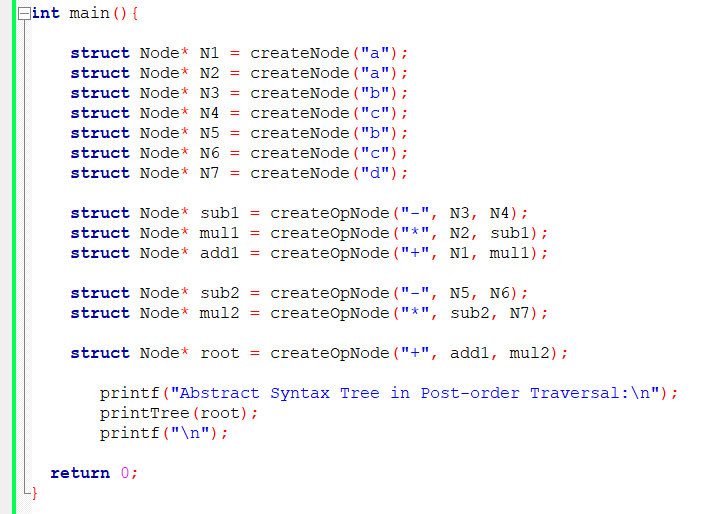
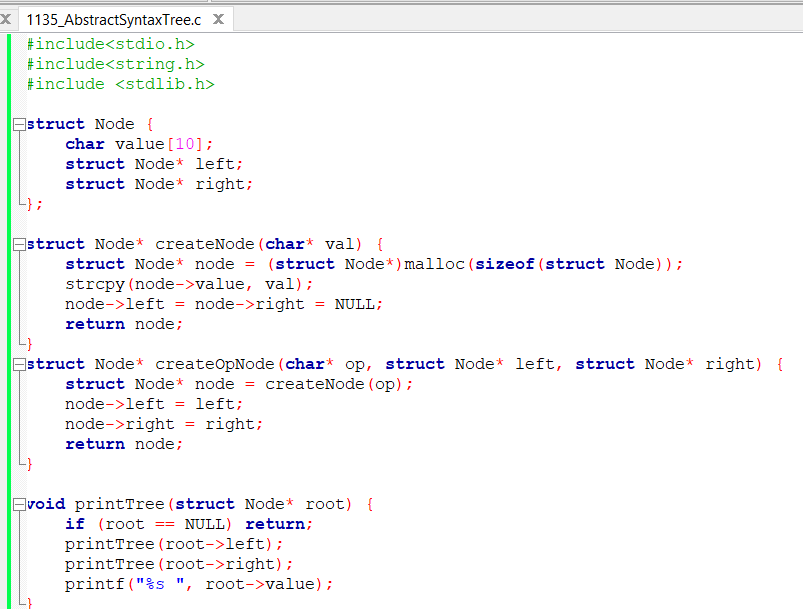
**printf("Abstract Syntax Tree in Post-order Traversal:\n");**

**printTree(root);**

**printf("\n");**

**return 0;**

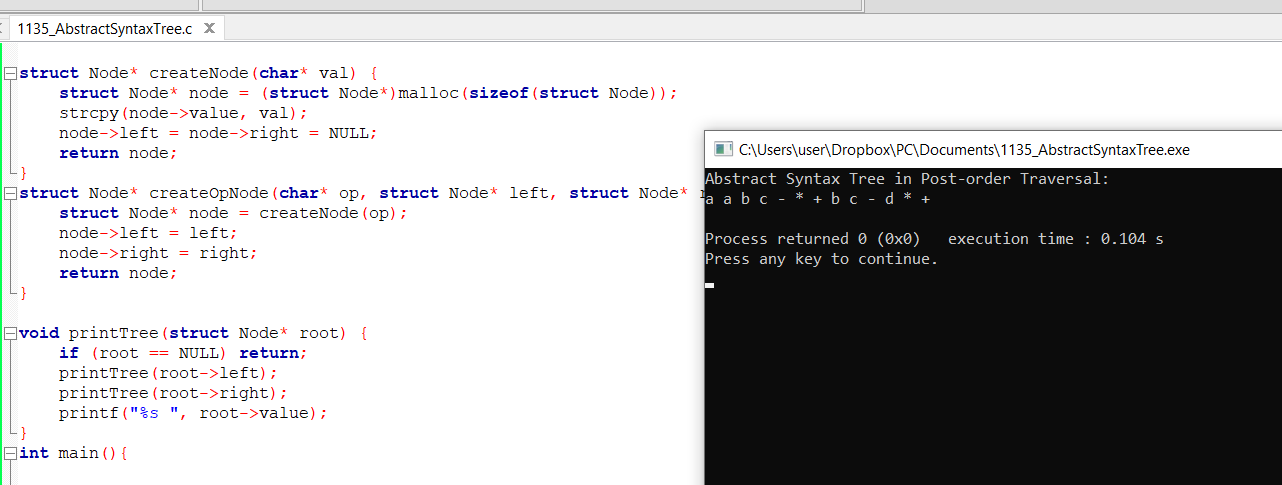
**}**

****

### ****Input****

This program does not require external input. All nodes and operations are hardcoded to construct a specific abstract syntax tree.

**Output**



### ****Conclusion****

In this lab, an Abstract Syntax Tree (AST) was constructed for the expression:

**a+a\*(b-c)+(b-c)\*d**

Using post-order traversal, the program successfully printed the expression in the order of its evaluation:

a a b c - \* + b c - d \* +

This traversal simulates the postfix form of the expression, which is essential for generating intermediate code in compilers or evaluating expressions in interpreters. The experiment demonstrates a key concept in compiler design—how syntax trees represent the structure of source code for further processing like parsing and code generation.