**Department of CSE**

**Compiler Lab (CSE 352)**

**Lab Report 07**

Construction of DAG for an Arithmetic Expression

**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.: 08**

**Experiment Name:** Construction of DAG for an Arithmetic Expression

**Problem Statement**

The objective of this experiment is to implement a C program to construct a Directed Acyclic Graph (DAG) for a given arithmetic expression. The DAG helps optimize expressions by eliminating common subexpressions and representing expressions more efficiently.

The expression used in this experiment is:

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

**C Code**

#include <stdio.h>

#include <stdlib.h>

#include <string.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 printPostOrder(struct Node\* root) {

if (root == NULL) return;

printPostOrder(root->left);

printPostOrder(root->right);

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

}

int main() {

// Common subexpression (b - c)

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

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

struct Node\* sub\_bc = createOpNode("-", b, c);

struct Node\* a1 = createNode("a");

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

struct Node\* mul1 = createOpNode("\*", a1, sub\_bc);

struct Node\* add1 = createOpNode("+", a1, mul1);

struct Node\* mul2 = createOpNode("\*", sub\_bc, d);

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

printf("DAG Postfix Expression:\n");

printPostOrder(root);

printf("\n");

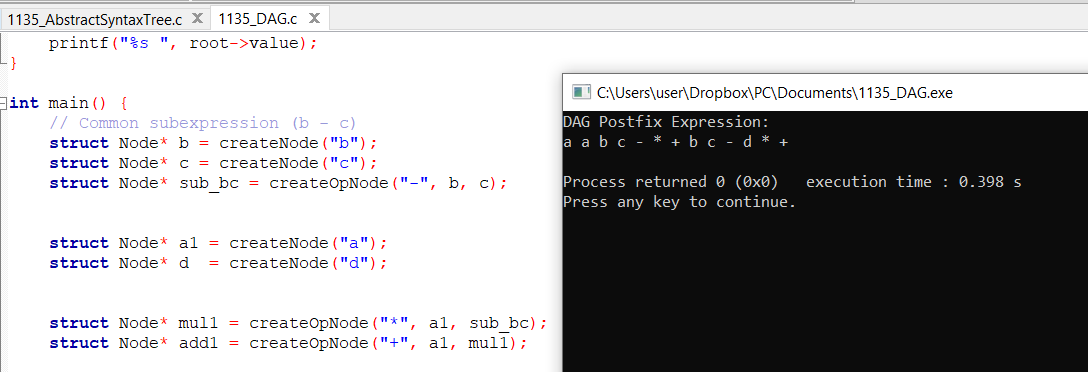
return 0;

}

**Input**

No external input is required. The expression is constructed directly in the code using tree nodes.

**Output**



**Conclusion**

This experiment successfully demonstrates how to construct a Directed Acyclic Graph (DAG) for the expression:

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

In the DAG, the common subexpression (b - c) is represented only once, which reduces redundant computation and optimizes the evaluation of the expression.

Using DAGs is essential in compiler optimization, particularly in common subexpressionelimination and intermediate code generation. This experiment improves understanding of how expressions are optimized at the syntactic and semantic analysis stages in compiler design.