Erick Gonzalez Parada #178145

Andre Francois Duhamel Gutiérrez #177315

Alejandro González Díaz #178645

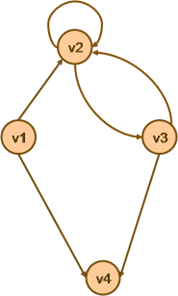
# Activity #9 [Graphs and trees](https://click.udlap.mx/webapps/assignment/uploadAssignment?content_id=_3011619_1&course_id=_36461_1&group_id=_262937_1&mode=view)

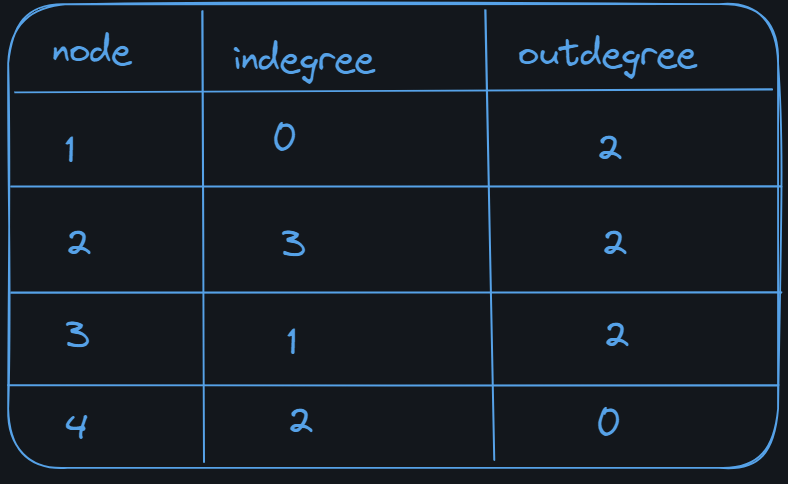
## Answers:

1) Using the following directed graph:

a. Show that the indegree sum of all nodes is equal to the sum of the outdegree of all nodes.

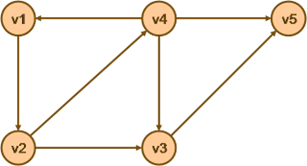
b. Show that the calculated sum in the previous point is equal to the number of edges.

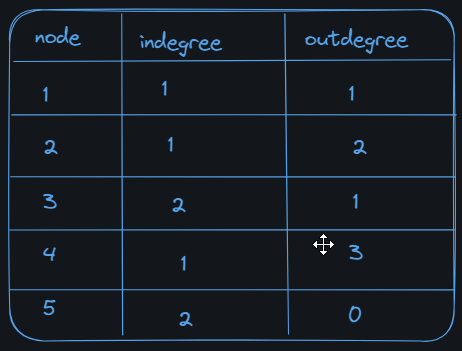


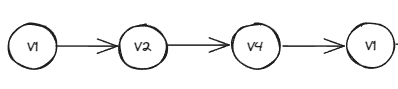


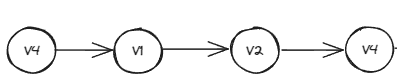
If we make the sum of the in and outdegrees, we observe that the number of outdegree is 6. The number of indegree is 6. The number of edges is 6, which is the same as the value of in and outdegrees

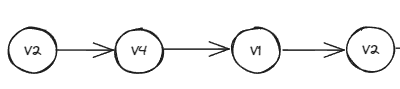
1. Find the indegree and outdegree of each node in the following graph and give all the elementary cycles (paths) in it. Obtain an acyclic directed graph by erasing one of the edges of the graph.



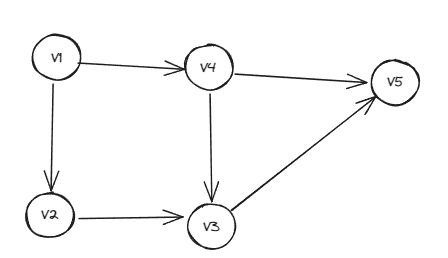


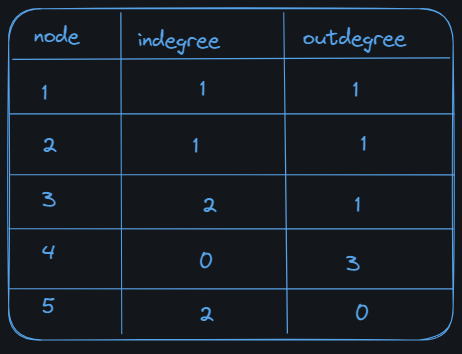




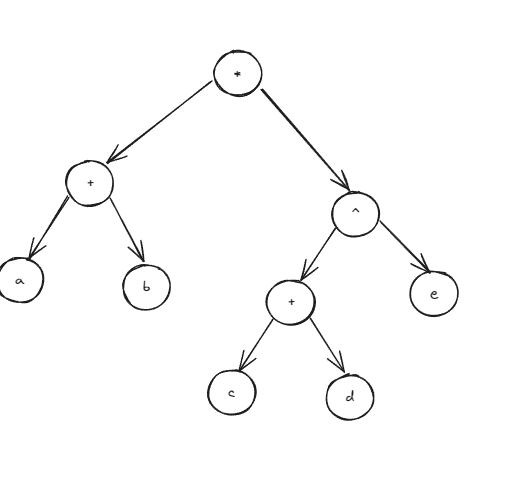


To convert the graph to an acyclic graph we need to remove the edge from V2 to V4



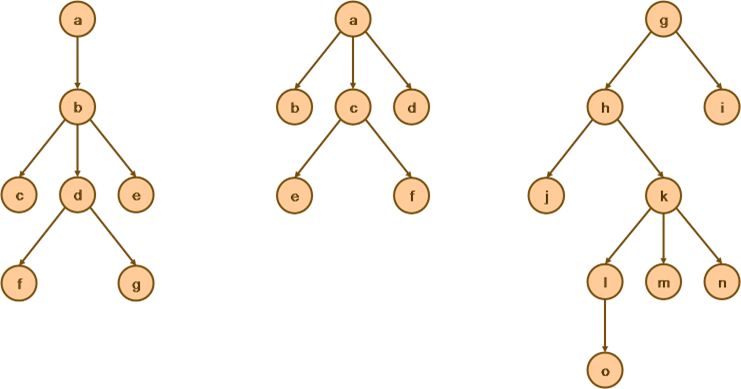


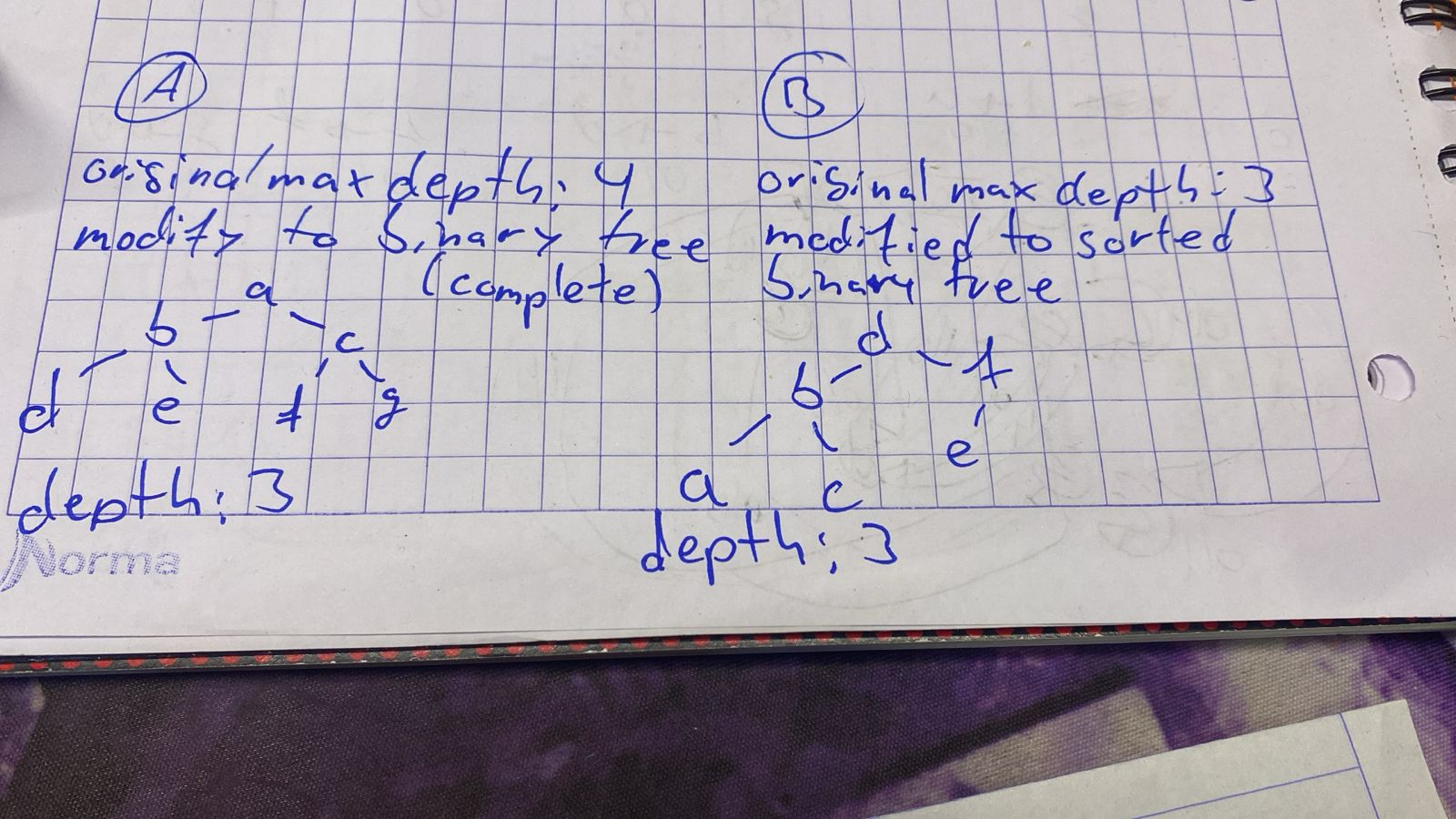
1. Represent the following formula: (a + b) \* (c + d)^e using a binary tree.

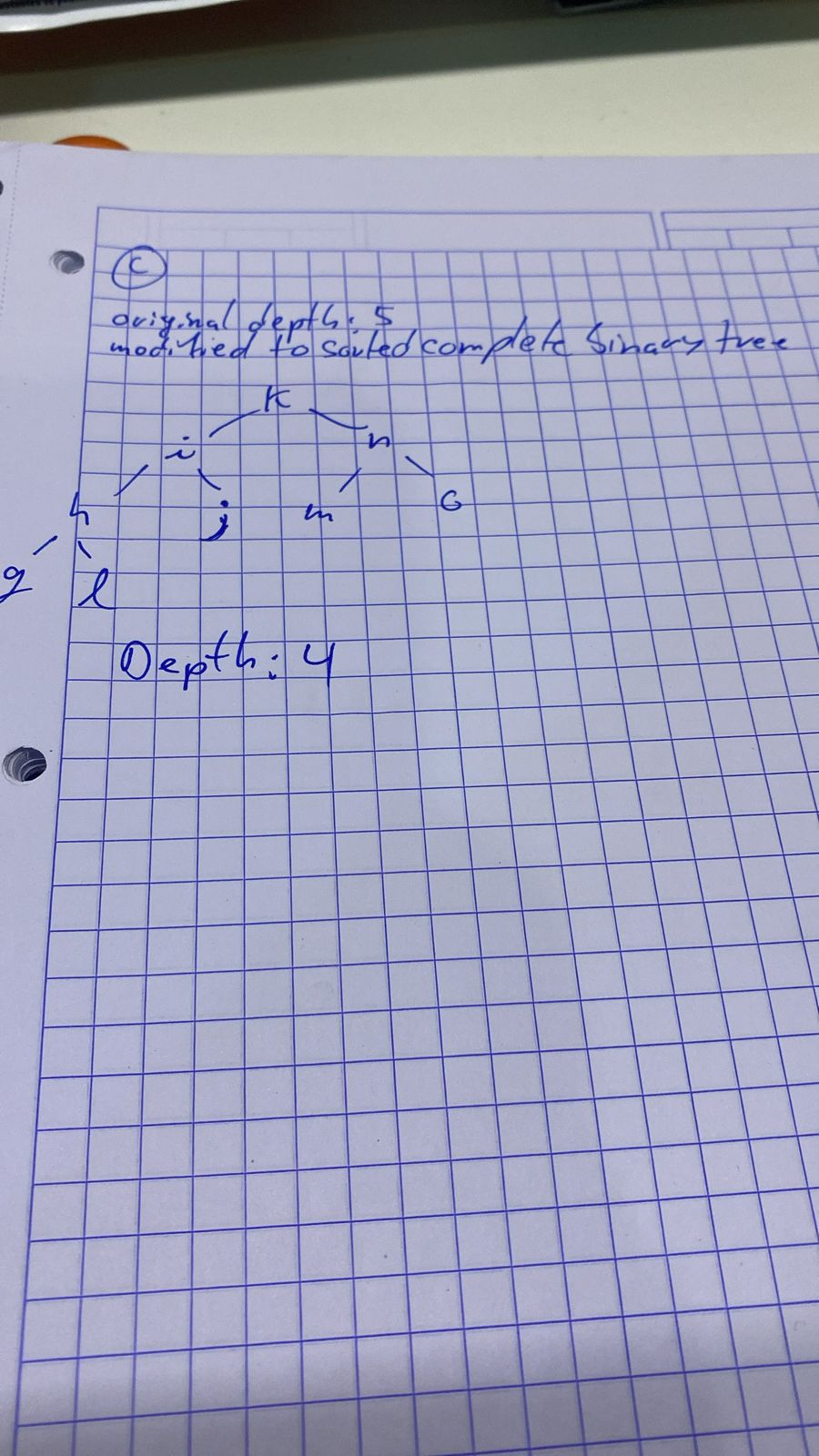


1. Consider the following forest made up of trees A, B, C, find the maximum depth of each tree and get three binary trees:
   1. A complete one representing information from A.
   2. A sort one representing information from B.
   3. A sort and complete one that represents the information from C.

a b c







## Implementations

For the section of code implementation, we as a team agreed to use different language (c++) just for fun because there was not language assigned.

## 1) Problem 1, approach adjacency matrix

We choose this approach because first of the viability, we easily reached the correct answers and even this allows us to check our answers if there were right or not, the run time of this approach is of O(n x n) where n is number of nodes, and again the reason we didn’t choose the adjacency list although this is O(n) when you have directed graphs it is hard to see easily the degrees (outdegree & indegree) of each nodes as for the adjacency matrix it’s literally counting.

## Code:

#include <iostream>

#include <vector>

using namespace std;

void calculateDegree(vector<vector<int>>& matrix) {

int totalIndegree = 0, totalOutdegree = 0;

int n = matrix.size();

for (int i = 0; i < n; i++) {

int outdegree = 0;

int indegree = 0;

for (int j = 0; j < n; j++) {

if (matrix[i][j] == 1) {

outdegree++;

totalOutdegree++;

}

if(matrix[j][i] == 1) {

indegree++;

totalIndegree++;

}

}

cout << "Node " << i + 1 << ": Indegree = " << indegree << ", Outdegree = " << outdegree << endl << endl;

}

cout << "Total indegrees = " << totalIndegree << endl << "Total outdegree = " << totalOutdegree << endl;

}

void printMatrix(vector<vector<int>>& matrix) {

int n = matrix.size();

for(int i = 0; i < n; i++) {

for(int j = 0; j < n; j++) {

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

int main() {

//aqui movemos la adjacency matrix

vector<vector<int>> matrix = {{0, 1, 0, 1},

{0, 1, 1, 0},

{0, 1, 0, 1},

{0, 0, 0, 0}};

cout << "Adjacency Matrix:" << endl;

printMatrix(matrix);

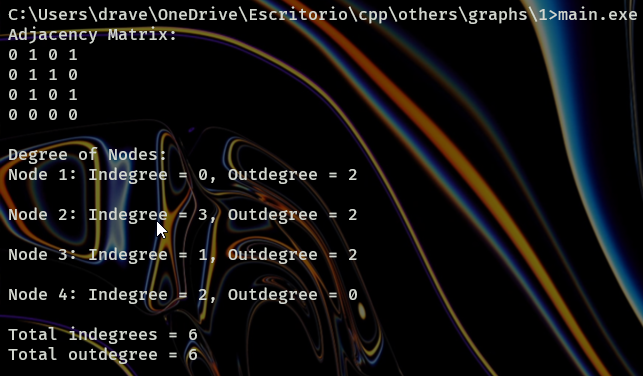
cout << "\nDegree of Nodes:" << endl;

calculateDegree(matrix);

return 0;

}

Result



As we can see we can prove and answer all the requested questions

## 2) Problem 2, adjacency matrix with DFS

Again, here for problem 2 we choose the versatility of the adjacency matrix to use them with DFS (Depth First Search algorithm) algorithms to get different paths, for the unmodified graph the dfs algorithm was challenging but we manage to get it to detect all possible cycles, for the acyclic graph we simply showcased (for fun) a normal dfs algorithm that can look up all the paths from a origin node to a destination node.

## Code:

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void printPath(vector<int> const &path)

{

for (int i : path)

cout << "node " << (i + 1) << " ";

cout << endl;

}

void DFS(vector<vector<int>> const &adjMatrix, vector<int> &path, vector<bool> &visited, int v, int u)

{

visited[v] = true;

path.push\_back(v);

if (v == u)

{

printPath(path);

}

else

{

for (int i = 0; i < adjMatrix[v].size(); i++)

{

if (adjMatrix[v][i] == 1 && !visited[i])

{

DFS(adjMatrix, path, visited, i, u);

}

}

}

visited[v] = false;

path.pop\_back();

}

void findAllPaths(vector<vector<int>> const &adjMatrix, int source, int destination)

{

vector<bool> visited(adjMatrix.size(), false);

vector<int> path;

DFS(adjMatrix, path, visited, source, destination);

}

void calculateDegree(vector<vector<int>>& matrix) {

int totalIndegree = 0, totalOutdegree = 0;

int n = matrix.size();

for (int i = 0; i < n; i++) {

int outdegree = 0;

int indegree = 0;

for (int j = 0; j < n; j++) {

if (matrix[i][j] == 1) {

outdegree++;

totalOutdegree++;

}

if(matrix[j][i] == 1) {

indegree++;

totalIndegree++;

}

}

cout << "Node " << i + 1 << ": Indegree = " << indegree << ", Outdegree = " << outdegree << endl << endl;

}

cout << "Total indegrees = " << totalIndegree << endl << "Total outdegree = " << totalOutdegree << endl;

}

void cyclicDFS(int v, int s, vector<int>& stack, vector<vector<int>>& cycles, const vector<vector<int>>& adjMatrix) {

stack.push\_back(v);

for (int i = 0; i < adjMatrix[v].size(); ++i) {

if (adjMatrix[v][i] == 1) {

if (i == s) {

cycles.push\_back(stack);

} else if (!count(stack.begin(), stack.end(), i)) {

cyclicDFS(i, s, stack, cycles, adjMatrix);

}

}

}

stack.pop\_back();

}

vector<vector<int>> findCycles(vector<vector<int>>& adjMatrix) {

vector<vector<int>> cycles;

for (int i = 0; i < adjMatrix.size(); ++i) {

vector<int> stack;

cyclicDFS(i, i, stack, cycles, adjMatrix);

for (int j = 0; j <= i; ++j) {

adjMatrix[j][i] = 0;

}

}

return cycles;

}

void printMatrix(vector<vector<int>>& matrix) {

int n = matrix.size();

for(int i = 0; i < n; i++) {

for(int j = 0; j < n; j++) {

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

int main() {

vector<vector<int>> matrix = {{0, 1, 0, 0, 0},

{0, 0, 1, 1, 0},

{0, 0, 0, 0, 1},

{1, 0, 1, 0, 1},

{0, 0, 0, 0, 0}};

cout << endl << "Adjacency Matrix the not modifying one:" << endl;

printMatrix(matrix);

cout << "\nDegree of Nodes:" << endl;

calculateDegree(matrix);

cout << endl << "paths" << endl;

//finding all possible silly elementary cycles

vector<vector<int>> cycles = findCycles(matrix);

for (const auto& cycle : cycles) {

for (int v : cycle) {

cout << v+1 << ' ';

}

cout << '\n';

}

vector<vector<int>> matrix2 = {{0, 1, 0, 0, 0},

{0, 0, 1, 0, 0},

{0, 0, 0, 0, 1},

{1, 0, 1, 0, 1},

{0, 0, 0, 0, 0}};

cout << endl << "Adjacency Matrix the modifying one:" << endl;

printMatrix(matrix2);

cout << "\nDegree of Nodes:" << endl;

calculateDegree(matrix2);

cout << endl << "paths" << endl;

//finding all possible silly paths

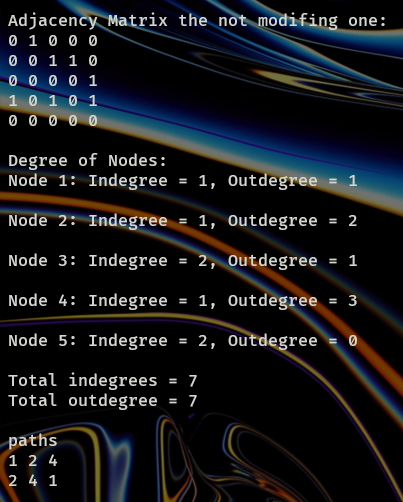
int source2 = 0, destination2 = 4;

findAllPaths(matrix2, source2, destination2);

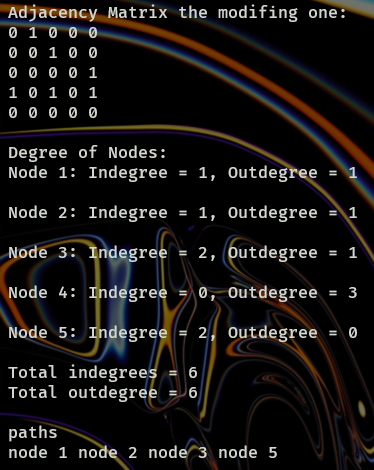
return 0;

}

## Result:



Where it prints "paths” and numbers, those are the cycles detected (in this case 2)



## 3) Writing equation and print Inorder

This problem asks us to represent (a + b) \* (c + d)^e in a binary tree. This is a good practice to understand the hierarchy in binary trees. In this implementation we decided to declare a struct and make the manual assignation in the Childrens of each node. After that, we need to make the inorder print to print the equation as it appears in the example.

#include <stdio.h>

#include <stdlib.h>

typedef struct treeNode

{//structure of the node and its children, as it is a binary, we need a left and right children

char data;

struct treeNode \*leftChild;

struct treeNode \*rightChild;

} treeNode;

treeNode \*createNode(char *data*) {

//We need to allocate the root value

treeNode \*result = malloc(sizeof(treeNode));

if (!result) {

printf("Memory allocation failed");

return NULL;

}

//we need to initialize the structure to null in all the structure

result->data = *data*;

result->leftChild = result->rightChild = NULL;

return result;

}

void printInOrder(treeNode \**node*) {

if (*node* == NULL) {

return;

}

*// Recursively print the left subtree*

printInOrder(*node*->leftChild);

*// Print the current node's data*

printf("*%c* ", *node*->data);

*// Recursively print the right subtree*

printInOrder(*node*->rightChild);

}

int main() {

*// Construct the tree for the expression "a + b \* c + d^e"*

treeNode \*root = createNode('+');

treeNode \*n2 = createNode('+');

treeNode \*n3 = createNode('\*');

treeNode \*n4 = createNode('^');

treeNode \*n5 = createNode('a');

treeNode \*n6 = createNode('b');

treeNode \*n7 = createNode('c');

treeNode \*n8 = createNode('d');

treeNode \*n9 = createNode('e');

//making the necessary assignations to the children of each node

root->leftChild = n2;

root->rightChild = n4;

n2->leftChild = n5;

n2->rightChild = n3;

n3->leftChild = n6;

n3->rightChild = n7;

n4->leftChild = n8;

n4->rightChild = n9;

Printing in order

printf("In-Order Traversal:\n");

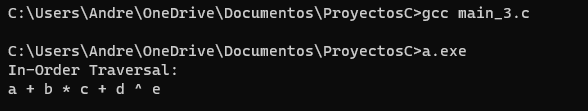
printInOrder(root);

printf("\n");

return 0;

}

OUTPUT



For the last problem we could not find a way in code to turn a ternary tree into a binary tree, at least it was a complex problem to solve.