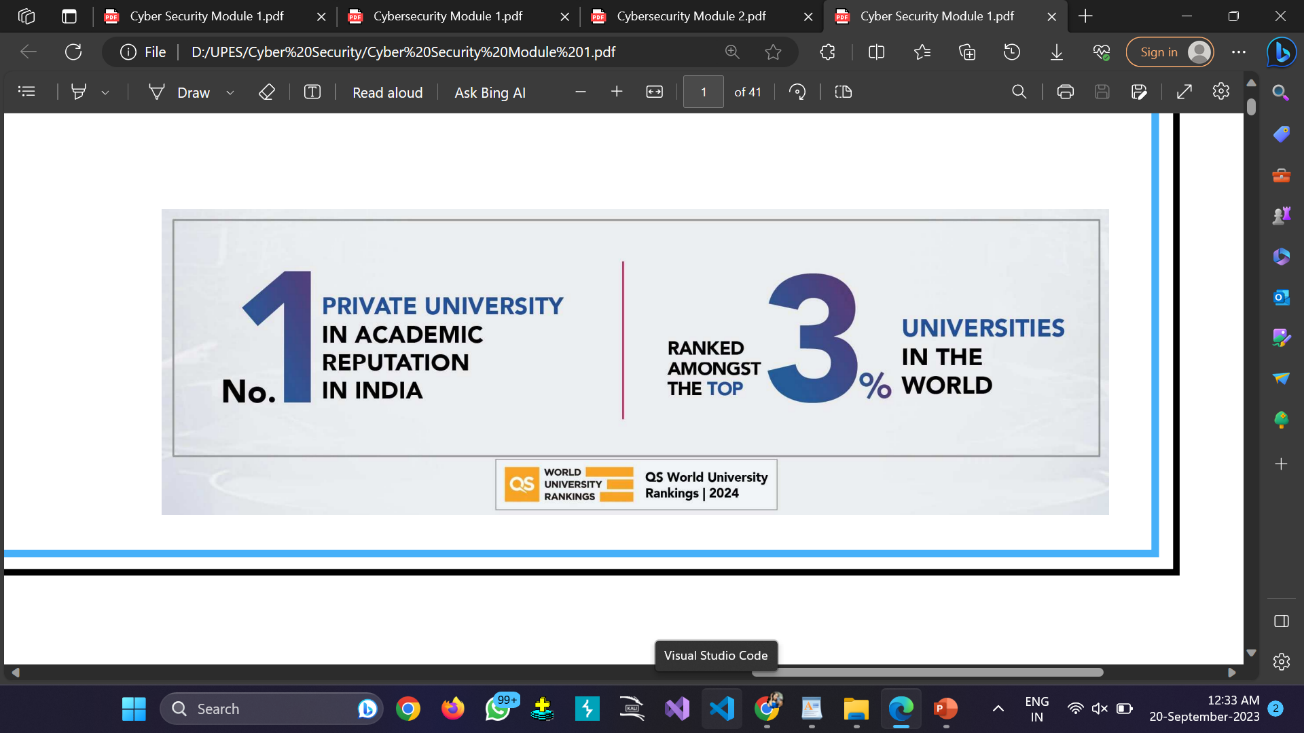
A picture containing text, clipart

Description automatically generated

1

**Lab Experiment: 09**

**Student Detail:**

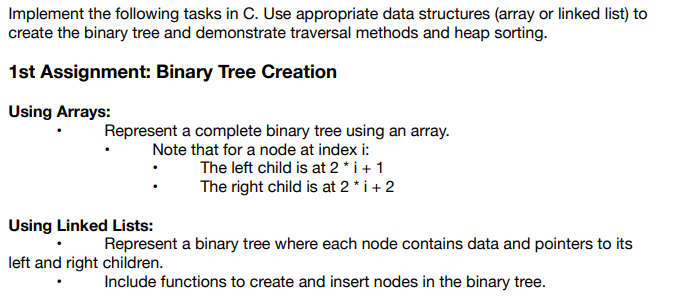
**• Name:** Prashant Joshi

**• Student ID:** 590010879

**• Branch:** MCA

**• Batch:** B1

**• Instructor:** Dr. Sourbh Kumar



Solution:

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum size of the array to store the binary tree

void insertInArray(int tree[], int \*size, int value) {

if (\*size < MAX\_SIZE) {

tree[\*size] = value;

(\*size)++;

} else {

printf("Array is full, cannot insert more elements.\n");

}

}

void displayArrayTree(int tree[], int size) {

printf("Binary Tree represented as an array:\n");

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

printf("%d ", tree[i]);

}

printf("\n");

}

int main() {

int tree[MAX\_SIZE];

int size = 0;

// Insert elements into the binary tree

insertInArray(tree, &size, 1); // Root node

insertInArray(tree, &size, 2); // Left child of root

insertInArray(tree, &size, 3); // Right child of root

insertInArray(tree, &size, 4); // Left child of node at index 1

insertInArray(tree, &size, 5); // Right child of node at index 1

// Display the array representation of the binary tree

displayArrayTree(tree, size);

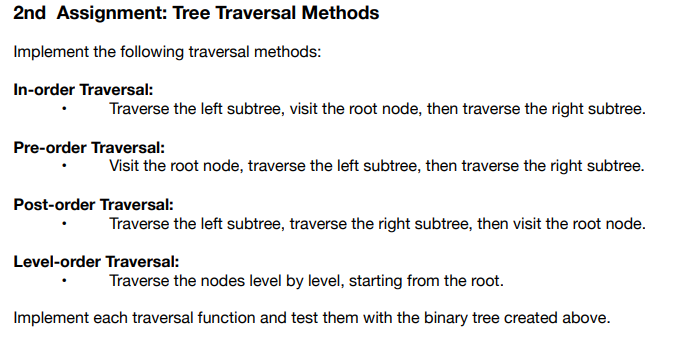
return 0;

}

Output:

**A screenshot of a computer screen

Description automatically generated**



Solution:

#include <stdio.h>

#include <stdlib.h>

// Definition of a node in the binary tree

struct TreeNode {

int data;

struct TreeNode\* left;

struct TreeNode\* right;

};

// Function to create a new node

struct TreeNode\* createNode(int value) {

struct TreeNode\* newNode = (struct TreeNode\*)malloc(sizeof(struct TreeNode));

newNode->data = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

// Traversal functions

void inOrderTraversal(struct TreeNode\* root) {

if (root != NULL) {

inOrderTraversal(root->left);

printf("%d ", root->data);

inOrderTraversal(root->right);

}

}

void preOrderTraversal(struct TreeNode\* root) {

if (root != NULL) {

printf("%d ", root->data);

preOrderTraversal(root->left);

preOrderTraversal(root->right);

}

}

void postOrderTraversal(struct TreeNode\* root) {

if (root != NULL) {

postOrderTraversal(root->left);

postOrderTraversal(root->right);

printf("%d ", root->data);

}

}

// Level-order Traversal (Breadth-first traversal)

void levelOrderTraversal(struct TreeNode\* root) {

if (root == NULL) return;

struct TreeNode\* queue[100];

int front = 0, rear = 0;

queue[rear++] = root;

while (front < rear) {

struct TreeNode\* current = queue[front++];

printf("%d ", current->data);

if (current->left != NULL) queue[rear++] = current->left;

if (current->right != NULL) queue[rear++] = current->right;

}

}

// Insert helper functions

void insertLeft(struct TreeNode\* parent, int value) {

parent->left = createNode(value);

}

void insertRight(struct TreeNode\* parent, int value) {

parent->right = createNode(value);

}

int main() {

// Create the binary tree

struct TreeNode\* root = createNode(1);

insertLeft(root, 2);

insertRight(root, 3);

insertLeft(root->left, 4);

insertRight(root->left, 5);

printf("In-order Traversal: ");

inOrderTraversal(root);

printf("\n");

printf("Pre-order Traversal: ");

preOrderTraversal(root);

printf("\n");

printf("Post-order Traversal: ");

postOrderTraversal(root);

printf("\n");

printf("Level-order Traversal: ");

levelOrderTraversal(root);

printf("\n");

return 0;

}

Output:

