1. Write a C program to perform Matrix Multiplication

#include <stdio.h>

#define ROWS 2

#define COLS 2

void matrixMultiplication(int mat1[ROWS][COLS], int mat2[ROWS][COLS], int result[ROWS][COLS]) {

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

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

result[i][j] = 0;

for (int k = 0; k < COLS; k++) {

result[i][j] += mat1[i][k] \* mat2[k][j];

}

}

}

}

int main() {

int mat1[ROWS][COLS] = {{1, 2}, {3, 4}};

int mat2[ROWS][COLS] = {{1, 0}, {0, 1}};

int result[ROWS][COLS];

matrixMultiplication(mat1, mat2, result);

printf("Result of Matrix Multiplication:\n");

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

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

printf("%d ", result[i][j]);

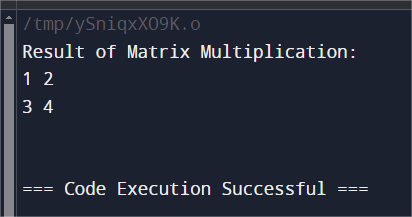
}

printf("\n");

}

return 0;

Output:



1. Write a C program to search a number using Linear Search method

#include <stdio.h>

int linearSearch(int arr[], int n, int key) {

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

if (arr[i] == key) {

return i;

}

}

return -1;

int main() {

int arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};

int n = sizeof(arr) / sizeof(arr[0]);

int key = 23;

int result = linearSearch(arr, n, key);

if (result != -1) {

printf("Element found at index: %d\n", result);

} else {

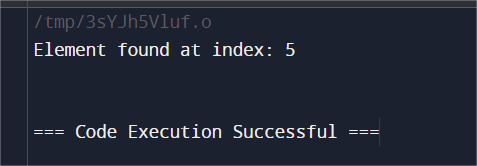
printf("Element not found\n");

}

return 0;

}

Output:



1. Write a C program to search a number using Binary Search method

#include <stdio.h>

int binarySearch(int arr[], int left, int right, int target) {

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == target)

return mid;

if (arr[mid] < target)

left = mid + 1;

else

right = mid - 1;

}

return -1;

}

int main() {

int arr[] = {2, 4, 6, 8, 10, 12, 14, 16};

int n = sizeof(arr) / sizeof(arr[0]);

int target = 10;

int result = binarySearch(arr, 0, n - 1, target);

if (result == -1)

printf("Element not found\n");

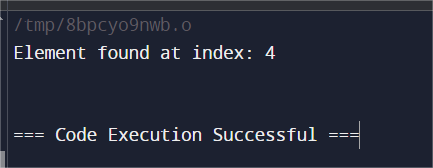
else

printf("Element found at index: %d\n", result);

return 0;

}

Output:



1. Write a C program to implement the Tree Traversals (Inorder, Preorder, Postorder)

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

void inorder(struct Node\* root) {

if (root == NULL) return;

inorder(root->left);

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

inorder(root->right);

}

void preorder(struct Node\* root) {

if (root == NULL) return;

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

preorder(root->left);

preorder(root->right);

}

void postorder(struct Node\* root) {

if (root == NULL) return;

postorder(root->left);

postorder(root->right);

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

}

int main() {

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

root->data = 1;

root->left = (struct Node\*)malloc(sizeof(struct Node));

root->right = (struct Node\*)malloc(sizeof(struct Node));

root->left->data = 2;

root->right->data = 3;

root->left->left = NULL;

root->left->right = NULL;

root->right->left = NULL;

root->right->right = NULL;

printf("Inorder traversal: ");

inorder(root);

printf("\n");

printf("Preorder traversal: ");

preorder(root);

printf("\n");

printf("Postorder traversal: ");

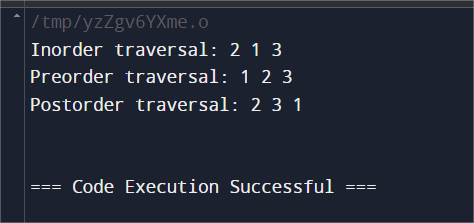
postorder(root);

printf("\n");

return 0;

}

Output:



1. Write a C program to search for a number, Min, Max from a BST

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*left, \*right;

};

struct Node\* insert(struct Node\* root, int data) {

if (root == NULL) {

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

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}

if (data < root->data)

root->left = insert(root->left, data);

else if (data > root->data)

root->right = insert(root->right, data);

return root;

}

int findMin(struct Node\* root) {

if (root == NULL)

return -1; // BST is empty

while (root->left != NULL)

root = root->left;

return root->data;

}

int findMax(struct Node\* root) {

if (root == NULL)

return -1; // BST is empty

while (root->right != NULL)

root = root->right;

return root->data;

}

int main() {

struct Node\* root = NULL;

root = insert(root, 50);

insert(root, 30);

insert(root, 20);

insert(root, 40);

insert(root, 70);

insert(root, 60);

insert(root, 80);

printf("Minimum value in the BST: %d\n", findMin(root));

printf("Maximum value in the BST: %d\n", findMax(root));

return 0;

}

Output:

