PRACTICE 5

PRATICE:5.1

AIM: Write a C program on BST

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
// Node structure
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};
// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Insert into BST
struct Node* insert(struct Node* root, int value) {
    if (root == NULL)
        return createNode(value);

    if (value < root->data)
```

```
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);

    return root;
}

// Inorder traversal (Display)
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

// Main function
int main() {
    struct Node* root = NULL;
    int choice, value;

    while (1) {
        printf("\n1. Insert\n2. Display (Inorder)\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2:
                printf("BST Inorder Traversal: ");
                inorder(root);
                printf("\n");
                break;
            case 3:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

OUTPUT:

1. Insert

2. Display (Inorder)

3. Exit

Enter your choice: 1

Enter value to insert: 100

1. Insert

2. Display (Inorder)

3. Exit

Enter your choice: 1

Enter value to insert: 50

1. Insert

2. Display (Inorder)

3. Exit

Enter your choice: 1

Enter value to insert: 60

1. Insert

2. Display (Inorder)

3. Exit

Enter your choice: 1

Enter value to insert: 150

1. Insert

2. Display (Inorder)

3. Exit

Enter your choice: 2

BST Inorder Traversal: 50 60 100 150

1. Insert

2. Display (Inorder)

3. Exit

Enter your choice: 3

PRATICE:5.2

AIM: Write a C program on BST with Traversals

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>

// Node structure
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Insert node into BST
```

```
struct Node* insert(struct Node* root, int value) {
    if (root == NULL)
        return createNode(value);

    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);

    return root;
}

// Inorder Traversal (Left, Root, Right)
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

// Preorder Traversal (Root, Left, Right)
void preorder(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
```

```
        preorder(root->right);
    }
}

// Postorder Traversal (Left, Right, Root)
void postorder(struct Node* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

// Main function
int main() {
    struct Node* root = NULL;
    int choice, value;

    while (1) {
        printf("\n1. Insert\n2. Inorder Traversal\n3. Preorder Traversal\n4.
Postorder Traversal\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
```

```
            printf("Enter value to insert: ");
            scanf("%d", &value);
            root = insert(root, value);
            break;
        case 2:
            printf("Inorder Traversal: ");
            inorder(root);
            printf("\n");
            break;
        case 3:
            printf("Preorder Traversal: ");
            preorder(root);
            printf("\n");
            break;
        case 4:
            printf("Postorder Traversal: ");
            postorder(root);
            printf("\n");
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid choice\n");
    }
}
```

```
    return 0;
}
```

OUTPUT:

```
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 1
Enter value to insert: 100

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 1
Enter value to insert: 50

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 1
Enter value to insert: 200
```

1. Insert

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Exit

Enter your choice: 2

Inorder Traversal: 50 100 200

1. Insert

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Exit

Enter your choice: 3

Preorder Traversal: 100 50 200

1. Insert

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Exit

Enter your choice: 4

Postorder Traversal: 50 200 100

1. Insert

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Exit

Enter your choice: 5

PRATICE:5.3

AIM: Write a C program on DFS

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100
int i;
// Structure for adjacency list node
struct Node {
    int vertex;
    struct Node* next;
};

// Structure for the graph
struct Graph {
    int numVertices;
    struct Node** adjLists;
    int* visited;
};
```

```
// Create a node
struct Node* createNode(int vertex) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->vertex = vertex;
    newNode->next = NULL;
    return newNode;
}

// Create a graph
struct Graph* createGraph(int vertices) {
    struct Graph* graph = malloc(sizeof(struct Graph));
    graph->numVertices = vertices;

    graph->adjLists = malloc(vertices * sizeof(struct Node*));
    graph->visited = malloc(vertices * sizeof(int));

    for (i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }

    return graph;
}

// Add edge (undirected)
```

```
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to dest
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Add edge from dest to src (since it's undirected)
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// DFS function
void DFS(struct Graph* graph, int vertex) {
    struct Node* adjList = graph->adjLists[vertex];
    struct Node* temp = adjList;

    graph->visited[vertex] = 1;
    printf("%d ", vertex);

    while (temp != NULL) {
        int connectedVertex = temp->vertex;

        if (!graph->visited[connectedVertex]) {
            DFS(graph, connectedVertex);
        }
```

```
        temp = temp->next;
    }
}

int main() {
    int vertices, edges, src, dest, start;

    printf("Enter number of vertices: ");
    scanf("%d", &vertices);

    struct Graph* graph = createGraph(vertices);

    printf("Enter number of edges: ");
    scanf("%d", &edges);

    printf("Enter edges (src dest):\n");
    for (i = 0; i < edges; i++) {
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }

    printf("Enter starting vertex for DFS: ");
    scanf("%d", &start);

    printf("DFS traversal starting from vertex %d: ", start);
    DFS(graph, start);
```

```
    return 0;
}
```

OUTPUT:

```
Enter number of vertices: 6
Enter number of edges: 7
Enter edges (src dest):
0 1
0 2
1 3
1 4
2 4
3 5
4 5
Enter starting vertex for DFS: 0
DFS traversal starting from vertex 0: 0 2 4 5 3 1
```

PRATICE:5.4

AIM: Write a C program on BFS

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

// Queue implementation
```

```
int queue[MAX], front = -1, rear = -1;

void enqueue(int value) {
    if (rear == MAX - 1)
        printf("Queue Overflow\n");
    else {
        if (front == -1)
            front = 0;
        queue[++rear] = value;
    }
}

int dequeue() {
    if (front == -1 || front > rear)
        return -1;
    return queue[front++];
}

int isEmpty() {
    return (front == -1 || front > rear);
}

// Graph structure using adjacency list
struct Node {
    int vertex;
    struct Node* next;
```

```
};

struct Graph {
    int numVertices;
    struct Node** adjLists;
    int* visited;
};

// Create a node
struct Node* createNode(int v) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

// Create a graph
struct Graph* createGraph(int vertices) {
    struct Graph* graph = malloc(sizeof(struct Graph));
    graph->numVertices = vertices;
    graph->adjLists = malloc(vertices * sizeof(struct Node*));
    graph->visited = malloc(vertices * sizeof(int));

    for (int i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
```

```
    }

    return graph;
}

// Add edge
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to dest
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Since it's undirected graph, add from dest to src as well
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// BFS algorithm
void bfs(struct Graph* graph, int startVertex) {
    graph->visited[startVertex] = 1;
    enqueue(startVertex);

    while (!isEmpty()) {
        int currentVertex = dequeue();
        printf("%d ", currentVertex);
```

```
        struct Node* temp = graph->adjLists[currentVertex];

        while (temp) {
            int adjVertex = temp->vertex;

            if (graph->visited[adjVertex] == 0) {
                graph->visited[adjVertex] = 1;
                enqueue(adjVertex);
            }
            temp = temp->next;
        }
    }
}

// Main function
int main() {
    int vertices = 6;

    struct Graph* graph = createGraph(vertices);

    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 4);
```

```
    addEdge(graph, 3, 5);
    addEdge(graph, 4, 5);

    printf("BFS traversal starting from vertex 0:\n");
    bfs(graph, 0);

    return 0;
}
```

OUTPUT:

```
BFS traversal starting from vertex 0:
0 2 1 4 5 3
```