LAB PROGRAM 9:

BINARY TREE AND DOUBLY LINKED LIST

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

typedef struct Node {
    int data;
    struct Node *left, *right;
} node;

node* createNode(int data) {
    node* new1 = (node*)malloc(sizeof(node));
    new1->data = data;
    new1->left = new1->right = NULL;
    return new1;
}

node* insertNode(node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else {
        root->right = insertNode(root->right, data);
    }
    return root;
```

```
}

void inorderTraversal(node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

void preorderTraversal(node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

void postorderTraversal(node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}

void displayTree(node* root, int space) {
```

```
    if (root == NULL) {
        return;
    }

    space += 10;

    displayTree(root->right, space);

    printf("\n");
    for (int i = 10; i < space; i++) {
        printf(" ");
    }
    printf("%d\n", root->data);

    displayTree(root->left, space);
}

int main() {
    node* root = NULL;
    int choice, value;

    printf("Binary Search Tree Operations:\n");
    while (1) {
        printf("\n1. Insert\n2. In-order Traversal\n3. Pre-order Traversal\n4. Post-order
Traversal\n5. Display Tree\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch (choice) {
    case 1:
        printf("Enter the value to insert: ");
        scanf("%d", &value);
        root = insertNode(root, value);
        break;
    case 2:
        printf("In-order Traversal: ");
        inorderTraversal(root);
        printf("\n");
        break;
    case 3:
        printf("Pre-order Traversal: ");
        preorderTraversal(root);
        printf("\n");
        break;
    case 4:
        printf("Post-order Traversal: ");
        postorderTraversal(root);
        printf("\n");
        break;
    case 5:
        printf("Tree Representation:\n");
        displayTree(root, 0);
        printf("\n");
        break;
    case 6:
```

```
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
    }
  }

  return 0;
}
```

```
Binary Search Tree Operations:

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 50

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 40

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 75

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 10

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 25

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 80
```

```
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 20

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 2
In-order Traversal: 10 20 25 40 50 75 80

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 3
Pre-order Traversal: 50 40 10 25 20 75 80

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 4
Post-order Traversal: 20 25 10 40 80 75 50

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 5
```

```
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display Tree
6. Exit
Enter your choice: 5
Tree Representation:

                    80

          75

50

          40

                              25

                                        20

                    10
```

2)DLL

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

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

typedef struct Node node;
node *start = NULL;
node *new1, *curr, *ptr;

void create();
void display();
void InsertLeft();
void DeleteSpecificElement();

void main() {
    int ch;
    while (1) {
        printf("\n1. Create \n2. Display \n3. Insert Left \n4. Delete Specific Element \n5. Exit");
        printf("\nEnter Your Choice: ");
        scanf("%d", &ch);

        switch (ch) {
```

```
            case 1: create();
                break;
            case 2: display();
                break;
            case 3: InsertLeft();
                break;
            case 4: DeleteSpecificElement();
                break;
            case 5: exit(0);
        }
    }
}

void create() {
    char ch;

    do {
        new1 = (node*)malloc(sizeof(node));
        printf("\nEnter Value: ");
        scanf("%d", &new1->data);
        new1->left = NULL;
        new1->right = NULL;

        if (start == NULL) {
            start = new1;
            curr = new1;
        } else {
```

```
            curr->right = new1;
            new1->left = curr;
            curr = new1;
        }

        printf("Do You Want to Add an Element (Y/N)? ");
        scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'Y');
}

void display() {
    if (start == NULL) {
        printf("\nLinked List is Empty.");
        return;
    }

    ptr = start;
    printf("\nElements in Linked List: \n");

    while (ptr != NULL) {
        printf("%d ", ptr->data);
        ptr = ptr->right;
    }
    printf("\n");
}

void InsertLeft() {
```

```
int val;
printf("\nEnter Value: ");
scanf("%d", &val);

new1 = (node*)malloc(sizeof(node));
new1->data = val;
new1->left = NULL;
new1->right = NULL;

printf("\nEnter the Value to Insert Left of: ");
scanf("%d", &val);

ptr = start;
while (ptr != NULL && ptr->data != val) {
    ptr = ptr->right;
}

if (ptr != NULL) {
    new1->right = ptr;
    new1->left = ptr->left;
    if (ptr->left != NULL) {
        ptr->left->right = new1;
    }
    ptr->left = new1;
    if (ptr == start) {
        start = new1;
    }
```

```
    } else {
        printf("\nValue not found.\n");
    }
}

void DeleteSpecificElement() {
    int value;
    printf("\nEnter Value to Delete: ");
    scanf("%d", &value);

    ptr = start;
    while (ptr != NULL && ptr->data != value) {
        ptr = ptr->right;
    }

    if (ptr == NULL) {
        printf("\nValue not found.\n");
        return;
    }

    if (ptr->left != NULL) {
        ptr->left->right = ptr->right;
    }
    if (ptr->right != NULL) {
        ptr->right->left = ptr->left;
    }
    if (ptr == start) {
```

```
        start = ptr->right;
    }

    free(ptr);
    printf("\nElement with value %d deleted.\n", value);
}
```

```
1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 1

Enter Value: 10
Do You Want to Add an Element (Y/N)? y

Enter Value: 20
Do You Want to Add an Element (Y/N)? y

Enter Value: 30
Do You Want to Add an Element (Y/N)? n

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 3

Enter Value: 40

Enter the Value to Insert Left of: 20

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 2

Elements in Linked List:
10 40 20 30
```

```
1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 4

Enter Value to Delete: 20

Element with value 20 deleted.

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 2

Elements in Linked List:
10 40 30

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 5

Process returned 0 (0x0)   execution time : 397.571 s
Press any key to continue.
```

LAB PROGRAM 9:

bfs and dfs

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

#define MAX 10

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

void enqueue(int item) {
    if (rear == MAX - 1) {
        printf("Queue is Full\n");
        return;
    }
    if (front == -1){
            front = 0;
    }
    queue[++rear] = item;
}

int dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue is Empty\n");
        return -1;
    }
    return queue[front++];
}
```

```
void bfs(int graph[MAX][MAX], int visited[MAX], int start, int n) {
    int i;
    enqueue(start);
    visited[start] = 1;

    printf("BFS Traversal: ");
    while (front <= rear) {
        int current = dequeue();
        printf("%d ", current);

        for (i = 0; i < n; i++) {
            if (graph[current][i] == 1 && visited[i] == 0){
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
    printf("\n");
}

void main() {
    int n, i, j, start;
    int graph[MAX][MAX], visited[MAX] = {0};

    printf("Enter the Number of Vertices: ");
    scanf("%d", &n);
```

```
    printf("Enter the Adjacency Matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    printf("Enter the Starting Vertex: ");
    scanf("%d", &start);

    bfs(graph, visited, start, n);

}
```

```
Enter the Number of Vertices: 5
Enter the Adjacency Matrix:
0 0 1 1 1
0 0 0 1 1
1 0 0 1 0
1 1 1 0 0
1 1 0 0 0
Enter the Starting Vertex: 1
BFS Traversal: 1 3 4 0 2

Process returned 10 (0xA)   execution time : 30.652 s
Press any key to continue.
```

3) DFS

```
#include <stdio.h>

#define MAX 10

int a[MAX][MAX], vis[MAX], n;

void dfs(int v);
int isConnected();

void main() {
    int i, j;

    printf("Enter Number of Vertices: ");
    scanf("%d", &n);

    printf("Enter Adjacency Matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    printf("\nDFS Traversal: ");

    if (isConnected()) {
        printf("\nThe graph is connected.\n");
```

```
    } else {
        printf("\nThe graph is disconnected.\n");
    }

    for (i = 0; i < n; i++) {
        vis[i] = 0;
    }

    printf("DFS Traversal: ");
    for (i = 0; i < n; i++) {
        if (vis[i] == 0) {
            dfs(i);
        }
    }

    printf("\n");
}

void dfs(int v) {
    printf("%d ", v);
    vis[v] = 1;

    for (int i = 0; i < n; i++) {
        if (a[v][i] == 1 && vis[i] == 0) {
            dfs(i);
        }
    }
```

```
}

int isConnected() {
    int i;

    for (i = 0; i < n; i++) {
        vis[i] = 0;
    }

    dfs(0);

    for (i = 0; i < n; i++) {
        if (vis[i] == 0) {
            return 0;
        }
    }
    return 1;
}
```