1. Write a C program to Implement Stack Using Array (Push, Pop, Display)

```
#include <stdio.h>
#define MAX_SIZE 5

int stack[MAX_SIZE];
int top = -1;

void push(int value) {
    if (top == MAX_SIZE - 1) {
        printf("Stack Overflow\n");
    } else {
        top++;
        stack[top] = value;
        printf("Pushed %d\n", value);
    }
}

void pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
    } else {
        printf("Popped %d\n", stack[top]);
        top--;
    }
}

void display() {
    if (top == -1) {
        printf("Stack is empty\n");
    } else {
        printf("Stack elements: ");
        for (int i = top; i >= 0; i--) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}

int main() {
    push(10);
    push(20);
```

```
    push(30);
    display();
    pop();
    display();
    push(40);
    push(50);
    push(60);
    push(70);
    display();
    return 0;
}
```

2. Write a C program Simple Queue Implementation Using Array

```
#include <stdio.h>
#define MAX_SIZE 5

int queue[MAX_SIZE];
int front = -1;
int rear = -1;

void enqueue(int value) {
    if ((rear + 1) % MAX_SIZE == front) {
        printf("Queue is full. Cannot enqueue %d.\n", value);
    } else {
        if (front == -1) front = 0;
        rear = (rear + 1) % MAX_SIZE;
        queue[rear] = value;
        printf("Enqueued %d\n", value);
    }
}

void dequeue() {
    if (front == -1) {
        printf("Queue is empty\n");
    } else {
        printf("Dequeued %d\n", queue[front]);
        if (front == rear) {
            front = -1;
            rear = -1;
        } else {
            front = (front + 1) % MAX_SIZE;
```

```
        }
    }
}

void display() {
    if (front == -1) {
        printf("Queue is empty\n");
    } else {
        printf("Queue elements: ");
        int i = front;
        while (1) {
            printf("%d ", queue[i]);
            if (i == rear) break;
            i = (i + 1) % MAX_SIZE;
        }
        printf("\n");
    }
}

int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    display();
    dequeue();
    display();
    enqueue(40);
    enqueue(50);
    enqueue(60);
    display();
    return 0;
}
```

3. Write a C program to Create a Linked List with 3 Nodes

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

struct Node {
    int data;
    struct Node* next;
};
```

```
void displayList(struct Node* node) {
    printf("Linked List: ");
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;

    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));

    head->data = 10;
    head->next = second;

    second->data = 20;
    second->next = third;

    third->data = 30;
    third->next = NULL;

    displayList(head);

    free(head);
    free(second);
    free(third);
    return 0;
}
```

4. Write a C program to Insert at Beginning in Linked List

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

struct Node {
```

```
    int data;
    struct Node* next;
};

void insertAtBeginning(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
    printf("Inserted %d at the beginning\n", new_data);
}

void displayList(struct Node* node) {
    if (node == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("List: ");
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    displayList(head);
    insertAtBeginning(&head, 30);
    insertAtBeginning(&head, 20);
    insertAtBeginning(&head, 10);
    displayList(head);
    return 0;
}
```

5. Write a C program to Count Nodes in a Linked List

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

struct Node {
    int data;
```

```
    struct Node* next;
};

void insertAtBeginning(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

int countNodes(struct Node* head) {
    int count = 0;
    struct Node* current = head;
    while (current != NULL) {
        count++;
        current = current->next;
    }
    return count;
}

void displayList(struct Node* node) {
    if (node == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("List: ");
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    printf("Total number of nodes: %d\n", countNodes(head));
    insertAtBeginning(&head, 10);
    insertAtBeginning(&head, 20);
    insertAtBeginning(&head, 30);
    displayList(head);
    printf("Total number of nodes: %d\n", countNodes(head));
    return 0;
}
```

6. Write a C program Search an Element in a Linked List

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

struct Node {
    int data;
    struct Node* next;
};

void insertAtBeginning(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

void searchElement(struct Node* head, int key) {
    struct Node* current = head;
    int position = 1;
    while (current != NULL) {
        if (current->data == key) {
            printf("Element %d found at position %d\n", key, position);
            return;
        }
        current = current->next;
        position++;
    }
    printf("Element %d not found in the list\n", key);
}

void displayList(struct Node* node) {
    if (node == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("List: ");
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
```

```
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    insertAtBeginning(&head, 50);
    insertAtBeginning(&head, 40);
    insertAtBeginning(&head, 30);
    displayList(head);

    searchElement(head, 40);
    searchElement(head, 99);
    return 0;
}
```

7. Write a C program Reverse a list of numbers

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

struct Node {
    int data;
    struct Node* next;
};

void insertAtBeginning(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

void reverseList(struct Node** head_ref) {
    struct Node* prev = NULL;
    struct Node* current = *head_ref;
    struct Node* next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
```

```
    *head_ref = prev;
}

void displayList(struct Node* node) {
    if (node == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("List: ");
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    insertAtBeginning(&head, 3);
    insertAtBeginning(&head, 2);
    insertAtBeginning(&head, 1);

    printf("Original ");
    displayList(head);

    reverseList(&head);
    printf("List has been reversed.\n");

    printf("Reversed ");
    displayList(head);
    return 0;
}
```

8. Write a C program to implement Stack Using Linked List and perform Push and Display

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

struct Node {
    int data;
    struct Node* next;
```

```
};

struct Node* top = NULL;

void push(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = top;
    top = newNode;
    printf("Pushed %d to stack\n", value);
}

void pop() {
    if (top == NULL) {
        printf("Stack Underflow\n");
    } else {
        struct Node* temp = top;
        printf("Popped %d\n", temp->data);
        top = top->next;
        free(temp);
    }
}

void display() {
    if (top == NULL) {
        printf("Stack is empty\n");
    } else {
        struct Node* temp = top;
        printf("Stack (top to bottom): ");
        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}

int main() {
    push(100);
    push(200);
    push(300);
    display();
    pop();
```

```
    display();
    pop();
    pop();
    pop();
    return 0;
}
```

9. Write a C program to implement Queue Using Linked List - Enqueue and Display

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

struct Node {
    int data;
    struct Node* next;
};

struct Node* front = NULL;
struct Node* rear = NULL;

void enqueue(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    if (rear == NULL) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    printf("Enqueued %d\n", value);
}

void dequeue() {
    if (front == NULL) {
        printf("Queue is empty\n");
    } else {
        struct Node* temp = front;
        printf("Dequeued %d\n", temp->data);
        front = front->next;
```

```
        if (front == NULL) {
            rear = NULL;
        }
        free(temp);
    }
}

void display() {
    if (front == NULL) {
        printf("Queue is empty\n");
    } else {
        struct Node* temp = front;
        printf("Queue (front to rear): ");
        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}

int main() {
    enqueue(10);
    enqueue(20);
    display();
    dequeue();
    display();
    enqueue(30);
    display();
    dequeue();
    dequeue();
    display();
    dequeue();
    return 0;
}
```

10. Write a C program for insertion in binary search tree.

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

```
struct Node {
    int key;
    struct Node *left, *right;
};

struct Node* newNode(int item) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

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

struct Node* insert(struct Node* node, int key) {
    if (node == NULL) return newNode(key);

    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    return node;
}

int main() {
    struct Node* root = NULL;
    root = insert(root, 50);
    printf("Inserted 50\n");
    root = insert(root, 30);
    printf("Inserted 30\n");
    root = insert(root, 20);
    printf("Inserted 20\n");
    root = insert(root, 40);
    printf("Inserted 40\n");
    root = insert(root, 70);
    printf("Inserted 70\n");
```

```
    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");
    return 0;
}
```

11. Write a C program for insertion in binary search tree (Duplicate)

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

struct Node {
    int key;
    struct Node *left, *right;
};

struct Node* newNode(int item) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

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

struct Node* insert(struct Node* node, int key) {
    if (node == NULL) return newNode(key);

    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    return node;
}
```

```
int main() {
    struct Node* root = NULL;
    root = insert(root, 50);
    printf("Inserted 50\n");
    root = insert(root, 30);
    printf("Inserted 30\n");
    root = insert(root, 20);
    printf("Inserted 20\n");
    root = insert(root, 40);
    printf("Inserted 40\n");
    root = insert(root, 70);
    printf("Inserted 70\n");

    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");
    return 0;
}
```

12. Write a C program Search for a Value in BST

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

struct Node {
    int key;
    struct Node *left, *right;
};

struct Node* newNode(int item) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

struct Node* insert(struct Node* node, int key) {
    if (node == NULL) return newNode(key);
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
```

```
        node->right = insert(node->right, key);
    return node;
}

struct Node* search(struct Node* root, int key) {
    if (root == NULL || root->key == key)
        return root;

    if (root->key < key)
        return search(root->right, key);

    return search(root->left, key);
}

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

int main() {
    struct Node* root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 70);
    root = insert(root, 20);
    root = insert(root, 40);

    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");

    int valueToFind = 40;
    if (search(root, valueToFind) != NULL)
        printf("Value %d found in the BST\n", valueToFind);
    else
        printf("Value %d not found in the BST\n", valueToFind);

    valueToFind = 99;
    if (search(root, valueToFind) != NULL)
        printf("Value %d found in the BST\n", valueToFind);
```

```
    else
        printf("Value %d not found in the BST\n", valueToFind);

    return 0;
}
```

13. Write a C program for Count Nodes in BST

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

struct Node {
    int key;
    struct Node *left, *right;
};

struct Node* newNode(int item) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

struct Node* insert(struct Node* node, int key) {
    if (node == NULL) return newNode(key);
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    return node;
}

int countNodes(struct Node* root) {
    if (root == NULL)
        return 0;
    else
        return 1 + countNodes(root->left) + countNodes(root->right);
}

int main() {
    struct Node* root = NULL;
```

```
    printf("Total number of nodes in BST: %d\n", countNodes(root));

    root = insert(root, 50);
    printf("Inserted 50\n");
    root = insert(root, 30);
    printf("Inserted 30\n");
    root = insert(root, 70);
    printf("Inserted 70\n");

    printf("Total number of nodes in BST: %d\n", countNodes(root));
    return 0;
}
```

14. Write a C program for Performing Binary Search on a Sorted Array

```
#include <stdio.h>

int binarySearch(int arr[], int left, int right, int x) {
    if (right >= left) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == x)
            return mid;

        if (arr[mid] > x)
            return binarySearch(arr, left, mid - 1, x);

        return binarySearch(arr, mid + 1, right, x);
    }
    return -1;
}

int main(void) {
    int arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Array: ");
    for(int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
```

```
    int x = 23;
    int result = binarySearch(arr, 0, n - 1, x);
    if (result == -1)
        printf("Element %d is not present in array\n", x);
    else
        printf("Element %d is present at index %d\n", x, result);

    x = 10;
    result = binarySearch(arr, 0, n - 1, x);
    if (result == -1)
        printf("Element %d is not present in array\n", x);
    else
        printf("Element %d is present at index %d\n", x, result);

    return 0;
}
```

15. Write a C program for Insertion Sort on an Array

```
#include <stdio.h>

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

int main() {
```

```
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Unsorted array: \n");
    printArray(arr, n);

    insertionSort(arr, n);

    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

16. Write a C program for Reverse an linked list (Duplicate)

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

struct Node {
    int data;
    struct Node* next;
};

void insertAtBeginning(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

void reverseList(struct Node** head_ref) {
    struct Node* prev = NULL;
    struct Node* current = *head_ref;
    struct Node* next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
}
```

```
void displayList(struct Node* node) {
    if (node == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("List: ");
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    insertAtBeginning(&head, 3);
    insertAtBeginning(&head, 2);
    insertAtBeginning(&head, 1);

    printf("Original ");
    displayList(head);

    reverseList(&head);
    printf("List has been reversed.\n");

    printf("Reversed ");
    displayList(head);
    return 0;
}
```

17. Write a C program DFS Traversal Using Adjacency Matrix (Basic Version)

```
#include <stdio.h>
#define V 5

int graph[V][V];
int visited[V];

void addEdge(int u, int v) {
    graph[u][v] = 1;
```

```
    graph[v][u] = 1;
}

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

    for (int i = 0; i < V; i++) {
        if (graph[v][i] == 1 && !visited[i]) {
            DFS(i);
        }
    }
}

int main() {
    for (int i = 0; i < V; i++) {
        visited[i] = 0;
        for (int j = 0; j < V; j++) {
            graph[i][j] = 0;
        }
    }

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

    int startVertex = 0;
    printf("Depth First Traversal (starting from vertex %d): ", startVertex);
    DFS(startVertex);
    printf("\n");

    return 0;
}
```

18. Write a C program for BSF traversal using adjacency matrix

```
#include <stdio.h>
#define V 5

int graph[V][V];
```

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

void enqueue(int v) {
    if (rear == V - 1) return;
    if (front == -1) front = 0;
    queue[++rear] = v;
}

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

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

void addEdge(int u, int v) {
    graph[u][v] = 1;
    graph[v][u] = 1;
}

void BFS(int startVertex) {
    for (int i = 0; i < V; i++) visited[i] = 0;

    visited[startVertex] = 1;
    enqueue(startVertex);

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

        for (int i = 0; i < V; i++) {
            if (graph[currentVertex][i] == 1 && !visited[i]) {
                visited[i] = 1;
                enqueue(i);
            }
        }
    }
}
```

```
int main() {
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) graph[i][j] = 0;
    }

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

    int startVertex = 0;
    printf("Breadth First Traversal (starting from vertex %d): ", startVertex);
    BFS(startVertex);
    printf("\n");

    return 0;
}
```

19. Write a C program to Check if Stack is Empty or Full

```
#include <stdio.h>
#define MAX_SIZE 3

int stack[MAX_SIZE];
int top = -1;

int isFull() {
    return (top == MAX_SIZE - 1);
}

int isEmpty() {
    return (top == -1);
}

void push(int value) {
    if (isFull()) {
        printf("Stack is Full! Cannot push %d.\n", value);
    } else {
        stack[++top] = value;
        printf("Pushed %d\n", value);
    }
```

```
}

void pop() {
    if (isEmpty()) {
        printf("Stack is Empty! Cannot pop.\n");
    } else {
        printf("Popped %d\n", stack[top--]);
    }
}

void checkStatus() {
    if (isFull())
        printf("Stack is FULL.\n");
    else if (isEmpty())
        printf("Stack is EMPTY.\n");
    else
        printf("Stack is neither empty nor full. (%d elements)\n", top + 1);
}

int main() {
    checkStatus();
    push(1);
    checkStatus();
    push(2);
    push(3);
    checkStatus();
    push(4);
    pop();
    checkStatus();
    pop();
    pop();
    checkStatus();
    pop();
    return 0;
}
```

20. Write a C program to find Maximum Element in an Array

```
#include <stdio.h>

int main() {
    int arr[] = {10, 32, 45, 9, 150, 4};
```

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

    if (n <= 0) {
        printf("Array must have at least one element.\n");
        return 1;
    }

    printf("Array: ");
    for(int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    int maxElement = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > maxElement) {
            maxElement = arr[i];
        }
    }

    printf("The maximum element in the array is: %d\n", maxElement);
    return 0;
}
```

21. Write a C program to Find the Middle Node of a Linked List

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

struct Node {
    int data;
    struct Node* next;
};

void insertAtBeginning(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

void findMiddle(struct Node* head) {
```

```
    struct Node* slow_ptr = head;
    struct Node* fast_ptr = head;

    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }

    while (fast_ptr != NULL && fast_ptr->next != NULL) {
        fast_ptr = fast_ptr->next->next;
        slow_ptr = slow_ptr->next;
    }
    printf("The middle element is: %d\n", slow_ptr->data);
}

void displayList(struct Node* node) {
    if (node == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("List: ");
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    insertAtBeginning(&head, 5);
    insertAtBeginning(&head, 4);
    insertAtBeginning(&head, 3);
    insertAtBeginning(&head, 2);
    insertAtBeginning(&head, 1);

    displayList(head);
    findMiddle(head);

    insertAtBeginning(&head, 0);
    displayList(head);
    findMiddle(head);
    return 0;
```

```
}
```

22. Write a C program for find Height of a Binary Tree (Simple Recursion)

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

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

int max(int a, int b) {
    return (a > b) ? a : b;
}

struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

struct Node* insert(struct Node* node, int key) {
    if (node == NULL) return newNode(key);
    if (key < node->data)
        node->left = insert(node->left, key);
    else if (key > node->data)
        node->right = insert(node->right, key);
    return node;
}

int findHeight(struct Node* node) {
    if (node == NULL)
        return 0;
    else
        return 1 + max(findHeight(node->left), findHeight(node->right));
}

int main() {
    struct Node* root = NULL;
```

```
    printf("The height of the empty tree is: %d\n", findHeight(root));

    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 70);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 80);

    printf("The height of the populated tree is: %d\n", findHeight(root));
    return 0;
}
```

23. Write a C program to Implement Queue Using Array (Duplicate)

```
#include <stdio.h>
#define MAX_SIZE 5

int queue[MAX_SIZE];
int front = -1;
int rear = -1;

void enqueue(int value) {
    if ((rear + 1) % MAX_SIZE == front) {
        printf("Queue is full. Cannot enqueue %d.\n", value);
    } else {
        if (front == -1) front = 0;
        rear = (rear + 1) % MAX_SIZE;
        queue[rear] = value;
        printf("Enqueued %d\n", value);
    }
}

void dequeue() {
    if (front == -1) {
        printf("Queue is empty\n");
    } else {
        printf("Dequeued %d\n", queue[front]);
        if (front == rear) {
            front = -1;
            rear = -1;
```

```
        } else {
            front = (front + 1) % MAX_SIZE;
        }
    }
}

void display() {
    if (front == -1) {
        printf("Queue is empty\n");
    } else {
        printf("Queue elements: ");
        int i = front;
        while (1) {
            printf("%d ", queue[i]);
            if (i == rear) break;
            i = (i + 1) % MAX_SIZE;
        }
        printf("\n");
    }
}

int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    display();
    dequeue();
    display();
    enqueue(40);
    enqueue(50);
    enqueue(60);
    display();
    return 0;
}
```

24. Write a C program to Create and Display a Simple Graph using Adjacency Matrix

```
#include <stdio.h>
#define V 5

void init(int arr[][V]) {
```

```
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            arr[i][j] = 0;
        }
    }
}

void addEdge(int arr[][V], int src, int dest) {
    if (src >= 0 && src < V && dest >= 0 && dest < V) {
        arr[src][dest] = 1;
        arr[dest][src] = 1;
        printf("Added edge between %d and %d\n", src, dest);
    } else {
        printf("Invalid vertices. Must be between 0 and %d\n", V - 1);
    }
}

void displayMatrix(int arr[][V]) {
    printf("\nAdjacency Matrix (%dx%d):\n", V, V);
    printf("   ");
    for (int i = 0; i < V; i++) printf("%d ", i);
    printf("\n--+");
    for (int i = 0; i < V; i++) printf("--");
    printf("\n");

    for (int i = 0; i < V; i++) {
        printf("%d | ", i);
        for (int j = 0; j < V; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int adjMatrix[V][V];
    init(adjMatrix);

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

```
    addEdge(adjMatrix, 2, 3);
    addEdge(adjMatrix, 3, 4);

    displayMatrix(adjMatrix);
    return 0;
}
```

25. Write a C program to Perform Insertion Sort on an Array (Duplicate)

```
#include <stdio.h>

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

int main() {
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Unsorted array: \n");
    printArray(arr, n);

    insertionSort(arr, n);

    printf("Sorted array: \n");
```

```
    printArray(arr, n);
    return 0;
}
```