1. Singly Linked List (SLL) – Generation and Display

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *next;
};
void display(struct Node *head) {
    struct Node *temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
int main() {
    struct Node *head = NULL, *temp, *newNode;
    int n, data;

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

    for (int i = 0; i < n; i++) {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = NULL;

        if (head == NULL)
            head = temp = newNode;
        else {
            temp->next = newNode;
            temp = newNode;
        }
    }

    printf("\nSingly Linked List: ");
    display(head);
    return 0;
}
```

Output:

```rust
Enter number of nodes: 3
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30

Singly Linked List: 10 -> 20 -> 30 -> NULL
```

2. Doubly Linked List (DLL) – Generation and Display

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *prev, *next;
};
void display(struct Node *head) {
    struct Node *temp = head;
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
int main() {
    struct Node *head = NULL, *temp, *newNode;
    int n, data;

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

    for (int i = 0; i < n; i++) {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = newNode->prev = NULL;

        if (head == NULL)
            head = temp = newNode;
        else {
            temp->next = newNode;
            newNode->prev = temp;
            temp = newNode;
        }
    }

    printf("\nDoubly Linked List: ");
    display(head);
    return 0;
}
```

Output:

```rust
Enter number of nodes: 3
Enter data for node 1: 5
Enter data for node 2: 15
Enter data for node 3: 25

Doubly Linked List: 5 <-> 15 <-> 25 <-> NULL
```

3. Circular Linked List (CLL) – Generation and Display

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *next;
};
void display(struct Node *head) {
    struct Node *temp = head;
    if (head == NULL) return;
    do {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("(back to head)\n");
}
int main() {
    struct Node *head = NULL, *temp, *newNode;
    int n, data;

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

    for (int i = 0; i < n; i++) {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = NULL;

        if (head == NULL)
            head = temp = newNode;
        else {
            temp->next = newNode;
            temp = newNode;
        }
        temp->next = head;
    }

    printf("\nCircular Linked List: ");
    display(head);
    return 0;
}
```

Output:

```rust

Enter number of nodes: 3
Enter data for node 1: 1
Enter data for node 2: 2
Enter data for node 3: 3

Circular Linked List: 1 -> 2 -> 3 -> (back to head)
```

4. Stack using Array

```c
#include <stdio.h>
#define SIZE 5
int stack[SIZE], top = -1;
void push(int val) {
    if (top == SIZE - 1)
        printf("Stack Overflow!\n");
    else
        stack[++top] = val;
}
void pop() {
    if (top == -1)
        printf("Stack Underflow!\n");
    else
        printf("Popped: %d\n", stack[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();
    return 0;
}
```

Output:

```yaml
Stack elements: 30 20 10
Popped: 30
Stack elements: 20 10
```

5. Queue using Array

```c
#include <stdio.h>
#define SIZE 5
int queue[SIZE];
int front = -1, rear = -1;
void enqueue(int val) {
    if (rear == SIZE - 1)
        printf("Queue Overflow!\n");
    else {
        if (front == -1) front = 0;
        queue[++rear] = val;
    }
}
void dequeue() {
    if (front == -1 || front > rear)
        printf("Queue Underflow!\n");
    else
        printf("Dequeued: %d\n", queue[front++]);
}
void display() {
    if (front == -1 || front > rear)
        printf("Queue is empty\n");
    else {
        printf("Queue elements: ");
        for (int i = front; i <= rear; i++)
            printf("%d ", queue[i]);
        printf("\n");
    }
}
int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    display();
    dequeue();
    display();
    return 0;
}
```

Output:

```yaml
Queue elements: 10 20 30
Dequeued: 10
Queue elements: 20 30
```

6. Binary Tree – Generation and Display

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *left, *right;
};
struct Node* createNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}
void display(struct Node *root) {
    if (root == NULL) return;
    printf("%d ", root->data);
    display(root->left);
    display(root->right);
}
int main() {
    struct Node *root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);

    printf("Tree (Preorder Display): ");
    display(root);
    return 0;
}
```

Output:

```
mathematica

Tree (Preorder Display): 1 2 4 5 3
```

7. Tree Traversals – Inorder, Preorder, Postorder

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

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

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

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

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

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

int main() {
    struct Node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    printf("Inorder: ");
```

```
    inorder(root);
    printf("\nPreorder: ");
    preorder(root);
    printf("\nPostorder: ");
    postorder(root);
    return 0;
}
```

Output:

makefile

```
Inorder: 4 2 5 1 3
Preorder: 1 2 4 5 3
Postorder: 4 5 2 3 1
```

8. Linear Search

```c
#include <stdio.h>

int main() {
    int a[5] = {10, 20, 30, 40, 50}, key, i, found = 0;

    printf("Enter element to search: ");
    scanf("%d", &key);

    for (i = 0; i < 5; i++) {
        if (a[i] == key) {
            printf("Element found at position %d\n", i + 1);
            found = 1;
            break;
        }
    }
    if (!found)
        printf("Element not found\n");

    return 0;
}
```

Output:

```
pgsql

Enter element to search: 40
Element found at position 4
```

9. Binary Search

```c
#include <stdio.h>

int main() {
    int a[5] = {10, 20, 30, 40, 50}, key, low = 0, high = 4, mid;
    printf("Enter element to search: ");
    scanf("%d", &key);

    while (low <= high) {
        mid = (low + high) / 2;
        if (a[mid] == key) {
            printf("Element found at position %d\n", mid + 1);
            return 0;
        } else if (a[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    printf("Element not found\n");
    return 0;
}
```

Output:

```
pgsql

Enter element to search: 30
Element found at position 3
```

10. Bubble Sort

```c
#include <stdio.h>

int main() {
    int a[5] = {64, 34, 25, 12, 22}, n = 5, temp;

    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (a[j] > a[j + 1]) {
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }

    printf("Sorted array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:

```
php

Sorted array: 12 22 25 34 64
```

11. Insertion Sort

```c
#include <stdio.h>

int main() {
    int a[5] = {5, 2, 4, 6, 1}, n = 5, key, j;

    for (int i = 1; i < n; i++) {
        key = a[i];
        j = i - 1;
        while (j >= 0 && a[j] > key) {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = key;
    }

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

```
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:

```
php

Sorted array: 1 2 4 5 6
```

12. Graph Generation and Display (Adjacency Matrix)

```
#include <stdio.h>

int main() {
    int n;
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    int graph[n][n];
    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);

    printf("\nGraph adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            printf("%d ", graph[i][j]);
        printf("\n");
    }
    return 0;
}
```

Output:

yaml

```
Enter number of vertices: 3
Enter adjacency matrix:
0 1 0
1 0 1
0 1 0

Graph adjacency matrix:
0 1 0
1 0 1
0 1 0
```