1. Given a connected graph and a starting vertex, perform Breadth First Search (BFS) and print the order of traversal.

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

#define MAX 100

int adj[MAX][MAX];
int V;

void BFS(int start)
{
    bool visited[MAX] = {false};
    int queue[MAX], front = 0, rear = 0;

    visited[start] = true;
    queue[rear++] = start;

    printf("BFS Traversal: ");
    while (front < rear)
    {
        int u = queue[front++];
        printf("%d ", u);

        for (int v = 0; v < V; v++)
        {
            if (adj[u][v] && !visited[v])
            {
                visited[v] = true;
                queue[rear++] = v;
            }
        }
    }
    printf("\n");
}

int main()
{
    int E, u, v, start;
    printf("Enter number of vertices: ");
    scanf("%d", &V);
    printf("Enter number of edges: ");
    scanf("%d", &E);

    for (int i = 0; i < E; i++)
    {
        scanf("%d %d", &u, &v);
        adj[u][v] = adj[v][u] = 1;
    }

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

    BFS(start);
    return 0;
}
```

2. Perform Depth First Search (DFS) from a given start node and print the order of visited Vertices.

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

#define MAX 100
```

```c
int adj[MAX][MAX];
bool visited[MAX];
int V;

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

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

int main()
{
    int E, u, v, start;
    printf("Enter number of vertices: ");
    scanf("%d", &V);
    printf("Enter number of edges: ");
    scanf("%d", &E);

    for (int i = 0; i < E; i++)
    {
        scanf("%d %d", &u, &v);
        adj[u][v] = adj[v][u] = 1;
    }
    printf("Enter start vertex: ");
    scanf("%d", &start);

    printf("DFS Traversal: ");
    DFS(start);
    printf("\n");

    return 0;
}
```

3. Given an undirected graph, determine whether the graph is connected (i.e., all vertices are reachable from any vertex).

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

#define MAX 100

int adj[MAX][MAX];
bool visited[MAX];
int V;

void DFS(int u)
{
    visited[u] = true;
    for (int v = 0; v < V; v++)
    {
        if (adj[u][v] && !visited[v])
        {
            DFS(v);
        }
    }
```

```
}

int main()
{
    int E, u, v;
    printf("Enter number of vertices: ");
    scanf("%d", &V);
    printf("Enter number of edges: ");
    scanf("%d", &E);

    for (int i = 0; i < E; i++)
    {
        scanf("%d %d", &u, &v);
        adj[u][v] = adj[v][u] = 1;
    }

    for (int i = 0; i < V; i++)
        visited[i] = false;

    DFS(0); // start from vertex 0

    bool connected = true;
    for (int i = 0; i < V; i++)
    {
        if (!visited[i])
        {
            connected = false;
            break;
        }
    }

    if (connected)
```

```
        printf("Graph is connected\n");
    else
        printf("Graph is NOT connected\n");

    return 0;
}
```

4. Given an undirected graph, check whether it contains any cycle.

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

#define MAX 100

int adj[MAX][MAX];
bool visited[MAX];
int V;

bool cycleUtil(int u, int parent)
{
    visited[u] = true;

    for (int v = 0; v < V; v++)
    {
        if (adj[u][v])
        {
            if (!visited[v])
            {
                if (cycleUtil(v, u))
                    return true;
            }
            else if (v != parent)
            {
                return true; // found a cycle
            }
        }
    }
    return false;
}

int main()
{
    int E, u, v;
    printf("Enter number of vertices: ");
    scanf("%d", &V);
    printf("Enter number of edges: ");
    scanf("%d", &E);

    for (int i = 0; i < E; i++)
    {
        scanf("%d %d", &u, &v);
        adj[u][v] = adj[v][u] = 1;
    }

    for (int i = 0; i < V; i++)
        visited[i] = false;

    bool cycle = false;
    for (int i = 0; i < V; i++)
    {
        if (!visited[i])
        {
            if (cycleUtil(i, -1))
            {
                cycle = true;
                break;
            }
        }
    }

    if (cycle)
        printf("Graph contains a cycle\n");
    else
        printf("Graph does NOT contain a cycle\n");

    return 0;
}
```