Lab Manual: Undirected and Unweighted Graphs in C++

1. Objective

- Understand basic graph concepts.
- Implement an **undirected**, **unweighted** graph.
- Perform basic operations: **add vertex**, **add edge**, **display graph**.

---

2. Theory

A **graph** is a collection of **vertices** (nodes) and **edges** (connections between vertices).

- **Undirected**: edges do not have a direction (edge A–B is the same as B–A).
- **Unweighted**: edges have no cost or value assigned to them.

We commonly use an **adjacency list** or **adjacency matrix** to represent graphs.

---

3. Tools Required

- **C++ compiler** (e.g., g++, clang++)
- **Text editor** or **IDE** (e.g., Visual Studio Code)

---

4. Code Example

Basic Undirected Unweighted Graph using Adjacency List

```
#include <iostream>
#include <vector>
#include <list>
using namespace std;

class Graph {
private:
    int V; // Number of vertices
    vector<list<int>> adjList;

public:
    Graph(int V) {
        this->V = V;
        adjList.resize(V);
    }

    // Add an undirected edge
    void addEdge(int u, int v) {
        adjList[u].push_back(v);
        adjList[v].push_back(u); // because the graph is undirected
    }

    // Display the graph
    void display() {
        for (int i = 0; i < V; ++i) {
            cout << "Vertex " << i << ":";
            for (auto neighbor : adjList[i]) {
                cout << " -> " << neighbor;
            }
            cout << endl;
        }
    }
};

int main() {
    int vertices = 5; // example with 5 vertices
```

```
    Graph g(vertices);

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

    cout << "Undirected and Unweighted Graph:" << endl;
    g.display();

    return 0;
}
```

---

5. Sample Output

```
Undirected and Unweighted Graph:
Vertex 0: -> 1 -> 4
Vertex 1: -> 0 -> 2 -> 3 -> 4
Vertex 2: -> 1 -> 3
Vertex 3: -> 1 -> 2 -> 4
Vertex 4: -> 0 -> 1 -> 3
```

---

6. Exercises

Tasks:

1. Modify the graph to allow adding a new vertex dynamically.
2. Implement a function to **remove** an edge.

3. Implement a function to **check if two vertices are connected**.
4. Write code to **count the number of edges** in the graph.
