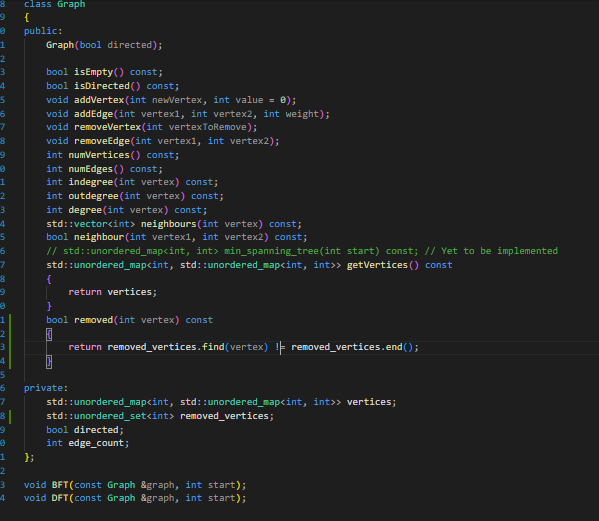
**Name: Akash kafle (26), Nabin kafle (27)**

**# Graph**

* Graphs are a fundamental data structure in computer science used to represent relationships between objects. They consist of vertices (also called nodes) and edges that connect these vertices.
  + **Key components of a graph:**
    - **Vertices:** The objects or entities in the graph.
    - **Edges:** Connections between vertices.
* **Types of graphs:**
  + - **Directed Graph (Digraph)**: Edges have a direction.
    - **Undirected Graph:** Edges have no direction.
    - **Weighted Graph:** Edges have associated weights or costs.
* **Common representations:**
* **Adjacency Matrix:** A 2D array where matrix[i][j] represents an edge from vertex i to j.
* **Adjacency List:** An array of lists, where each list contains the neighbors of a vertex.
* **Basic operations**:
* Adding vertices and edges
* Removing vertices and edges
* Checking if an edge exists between two vertices
* Finding neighbors of a vertex
* **Common algorithms:**
  + Depth-First Search (DFS)
  + Breadth-First Search (BFS)
  + Dijkstra's Algorithm (shortest path)
  + Kruskal's and Prim's Algorithms (minimum spanning tree)

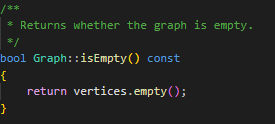
**# Graph.h**



**# Graph.cpp**

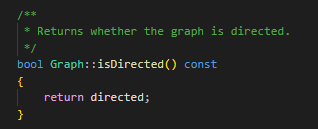
1. isEmpty():

* Returns true if the graph is empty, and false otherwise



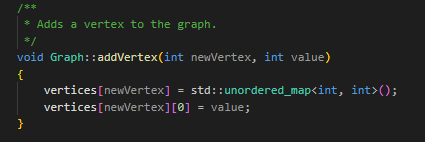
1. isDirected():

* Returns true if the graph is directed, and false otherwise



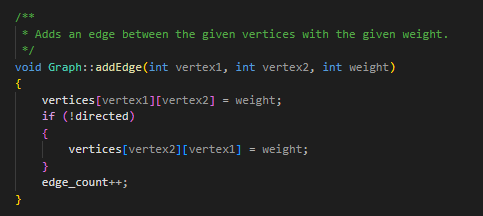
1. addVertex(newVertex):

* Inserts a new vertex to the graph



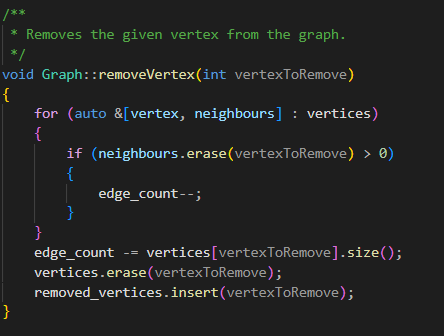
1. addEdge(vertex1, vertex2):

* Adds an edge from vertex1 to vertex2



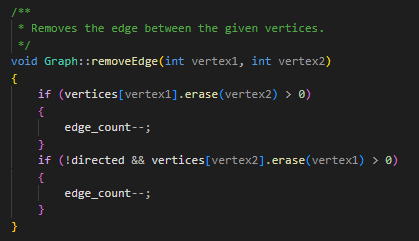
1. removeVertex(vertexToRemove):

* Remove a vertex from the graph

****

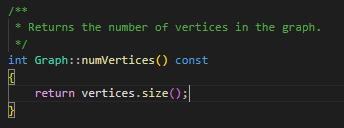
1. removeEdge(vertex1, vertex2):

* Remove an edge from the graph



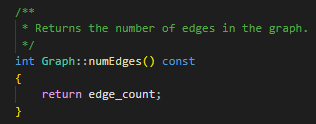
1. numVertices():

* Returns the number of vertices in the graph



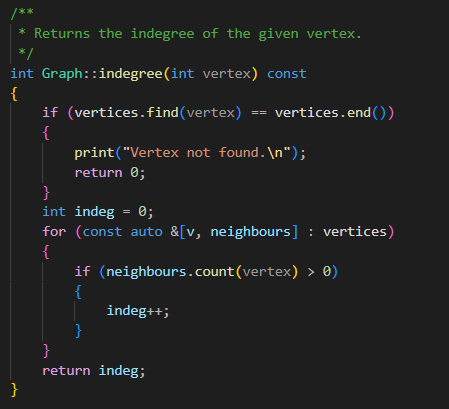
1. numEdges():

* Returns the number of edges in the graph



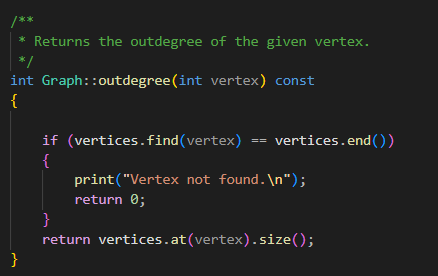
1. indegree(vertex):

* Returns the indegree of a vertex

****

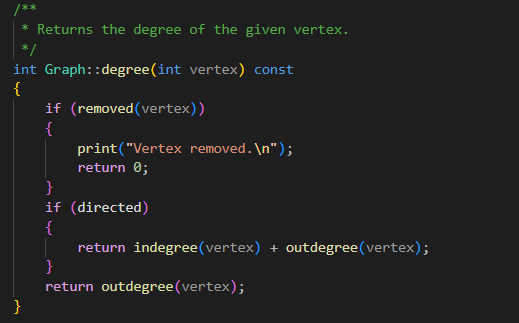
1. outdegree(vertex):

* Returns the outdegree of a vertex

****

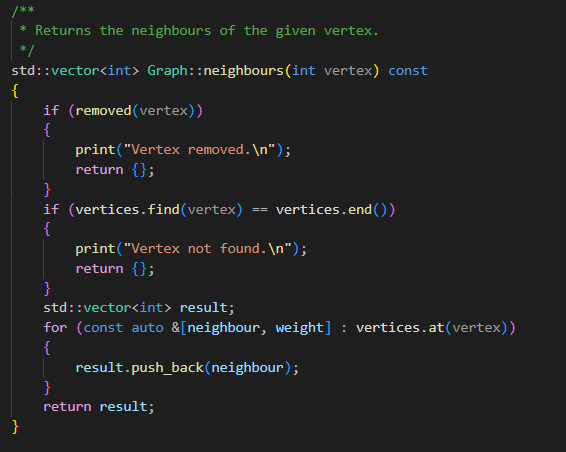
1. degree(vertex):

* Returns the degree of a vertex

****

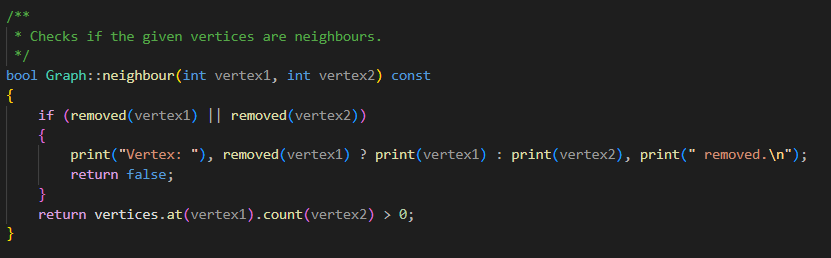
1. neighbours(vertex):

* Returns the neighbours of a vertex

****

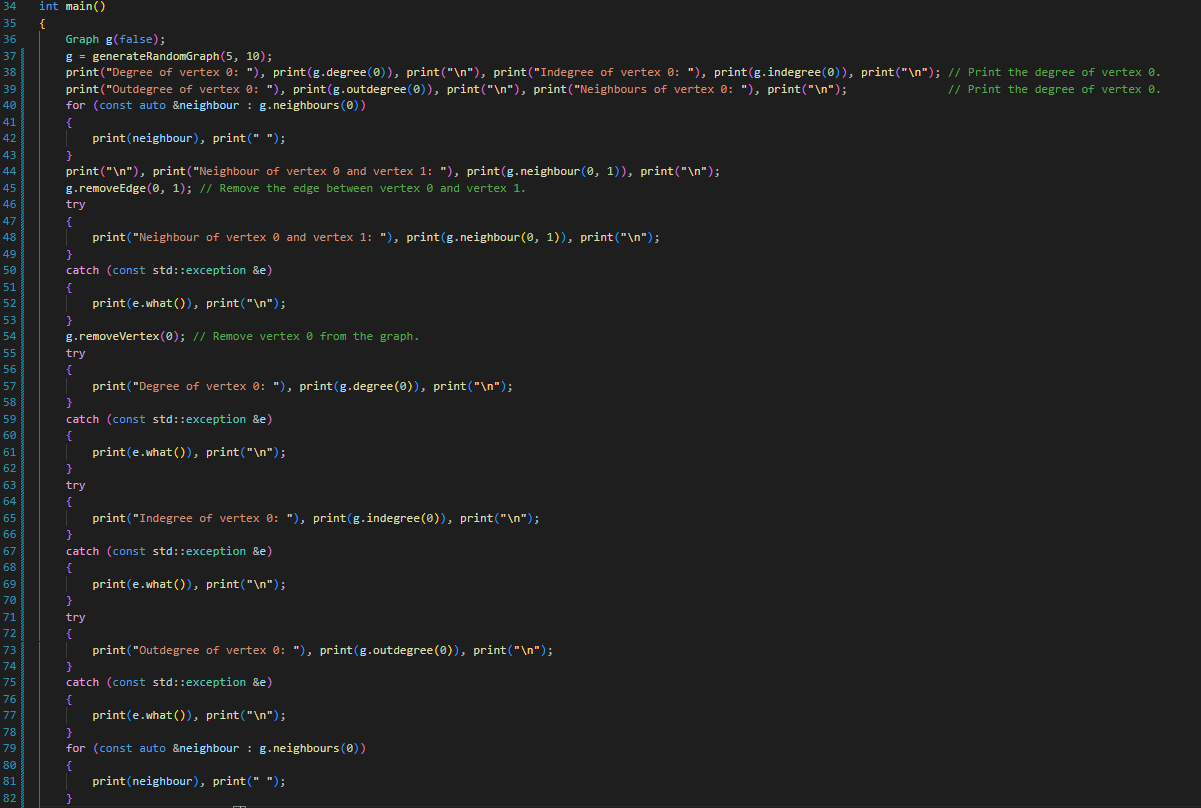
1. neighbour(vertex1, vertex2):

* Returns true if vertex2 is a neighbour of vertex1.

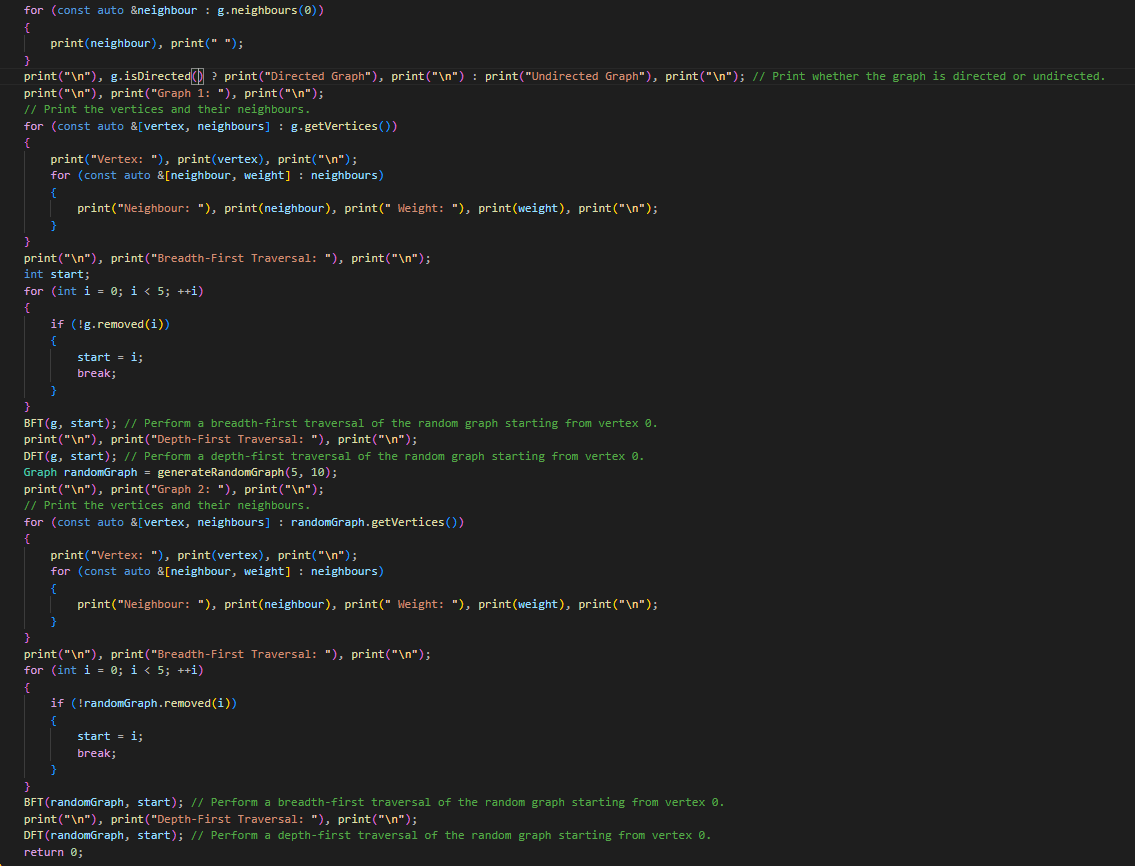
****

**# Main.cpp**

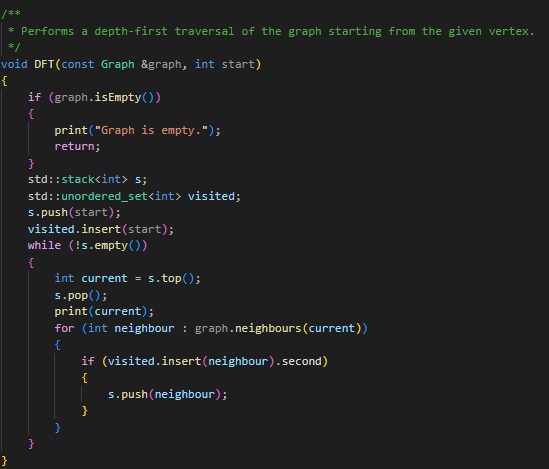
**Part 1: ( Please zoom )**

****

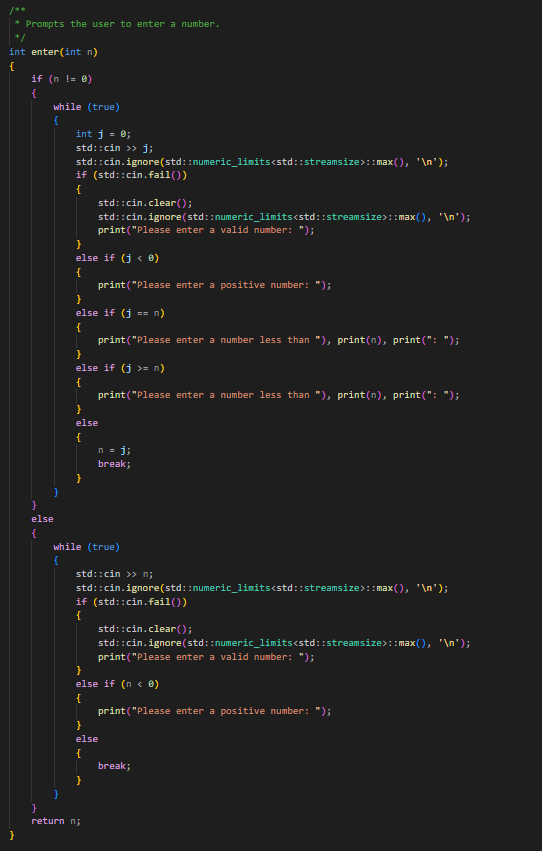
**Part 2: ( Please zoom )**

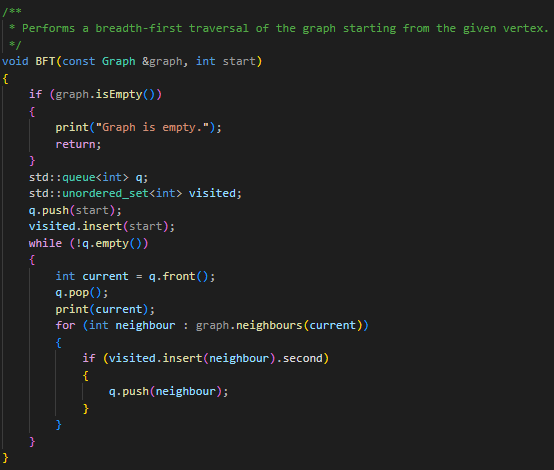
****

**# Helper functions:**

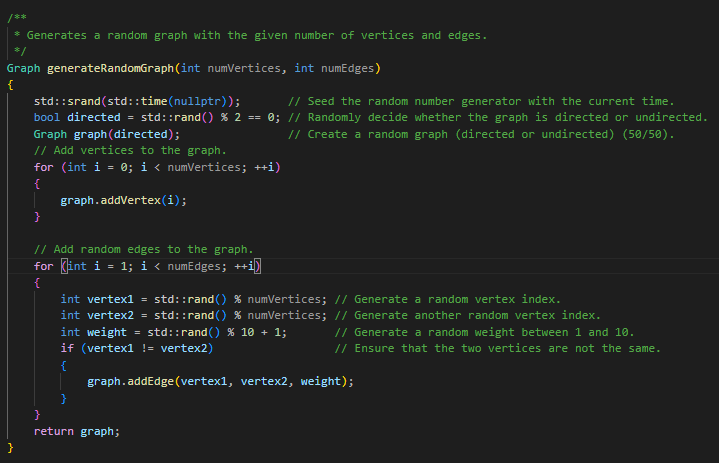


**Input function which takes input from the user:**



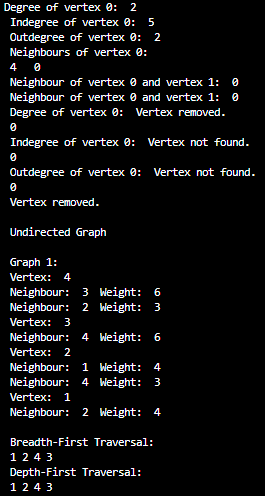


**Bonus: Write a program to generate a random graph.**

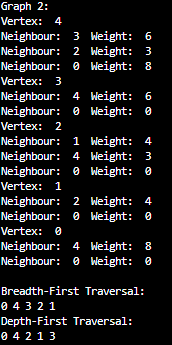


**# Output:**

Graph 1:

****

Graph 2:



**Summary:**

This lab focused on implementing a Graph data structure and its associated operations. The key components included:

1. A Graph class with methods for basic graph operations.

2. Implementation of various graph functions in Graph.cpp, including:

- Checking if the graph is empty or directed

- Adding and removing vertices and edges

- Counting vertices and edges

- Calculating indegree, outdegree, and degree of vertices

- Finding neighbors of vertices

- Traversing algorithms

3. A Main.cpp file demonstrating the usage of the Graph class, including:

- Creating and manipulating two different graphs

- Performing various operations on these graphs

4. Helper functions for input handling and graph manipulation.

5. A bonus program to generate a random graph.

The lab provided practical experience in working with graph data structures, implementing common graph algorithms, and understanding the relationships between vertices and edges in both directed and undirected graphs. The output screenshots demonstrate the successful implementation and testing of the graph operations.