**Name:** Safyan Anwar

**Roll no:** -018

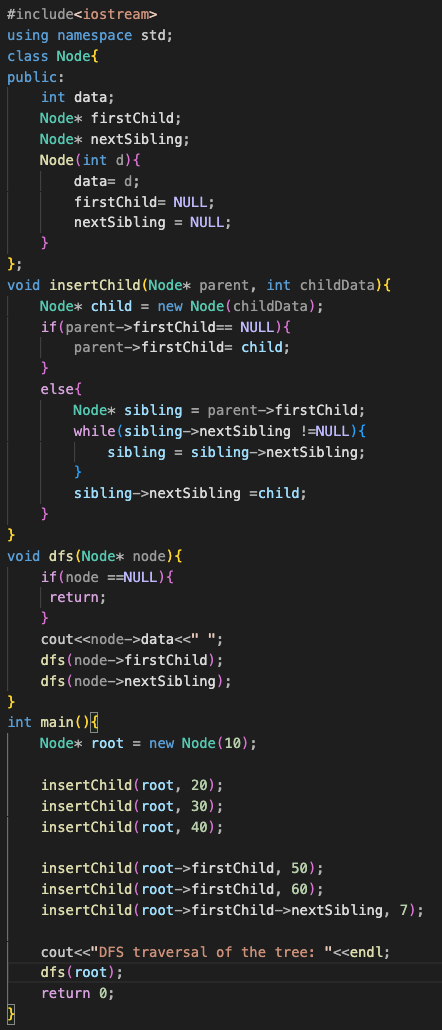
**Section:** 3A-BSSE

**Assignment no:** 13

**LAB TASK:**

**Q1: “Tree DFS”:**

**Code:**

****

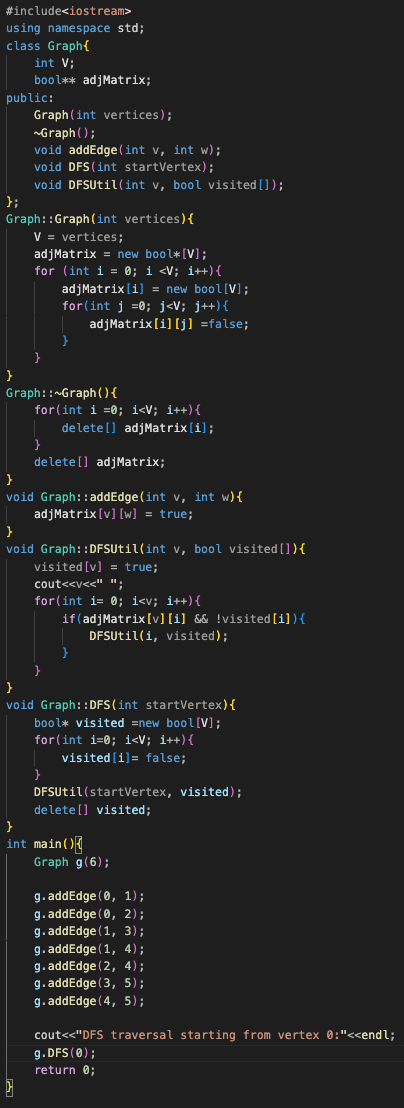
**Output:**

****

**Explanation:**

* **Node class:** Represent a tree node with data a pointer to the firstchild and a pointer to the next stibling.
* **inserthild function:** Add a new child node to a specified parent node. If the parent has no children the new child becomes the first child; otherwise, it is added as the last sibling.
* **dfs function:** Perform a depth first search traversal, visiting the current node then recursively visiting its first child and next sibling.
* **Main function:**
  + Create a root node with value 10.
  + Inserts child nodes to build the tree structure.
  + Call the dfs function to print the nodes in pre-order.

**Q2: “Graph DFS”:**

****

**Output:**

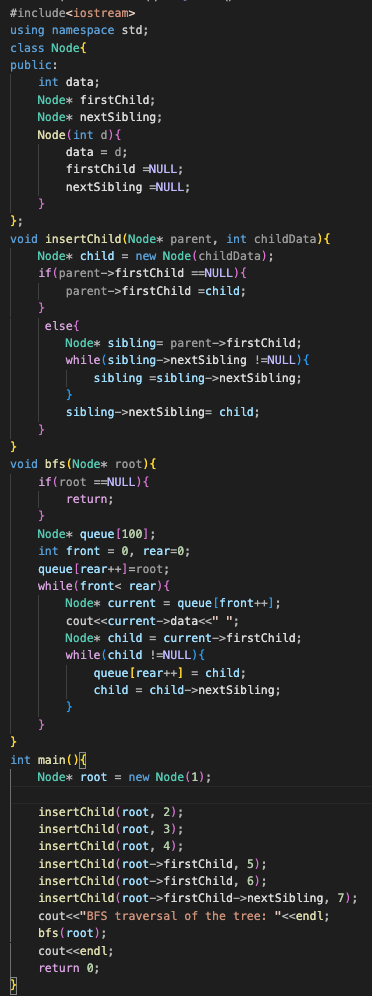
****

**Explanation:**

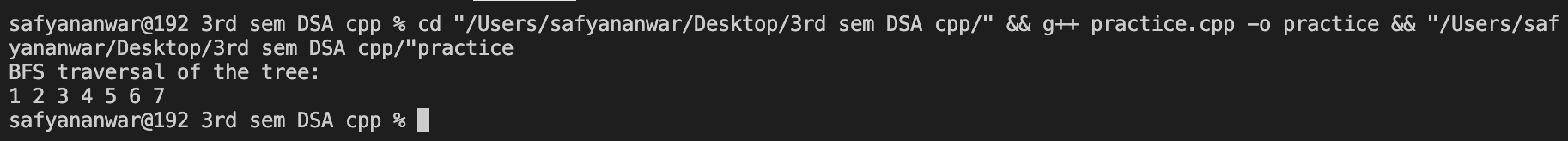
* **Graph class**: Represents a directed graph using an adjacency matrix.
* Constructor: Initializes the graph with a specified number of vertices and allocates memory for the adjacency matrix, setting all entries to false.
* **Destructor:** Frees the memory allocated for the adjacency matrix.
* addEdge Method: Adds a directed edge from vertex v to vertex w by updating the adjacency matrix.
* **DFSUtil method:** A recursive helper function that visits vertices in dfs order marking them as visited and printing their values.
* **DFS method**: Initializes a visited array and starts the DFS traversal from a specified vertex.
* **Main function:** Creates a graph with 6 vertices, adds edges, and performs DFS starting from vertex 0 printing the traversal order.

**Q3: “Tree BST”:**

**Code:**

****

**Output:**

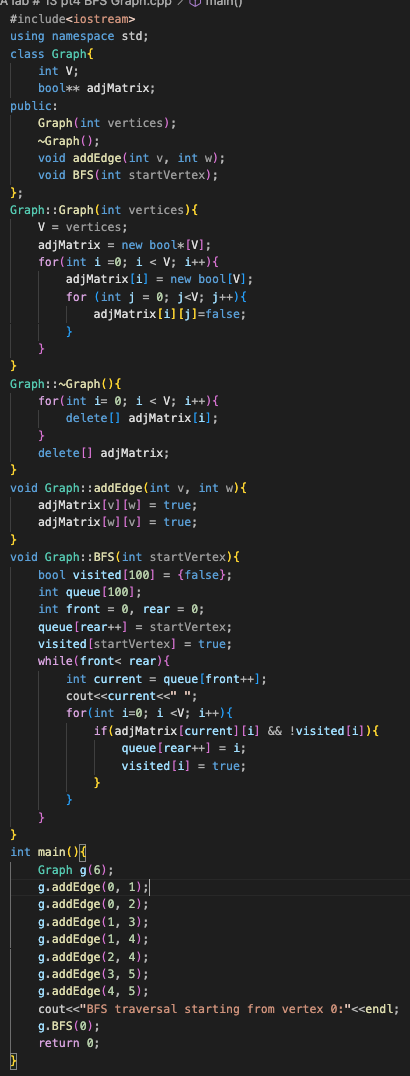
****

**Explain:**

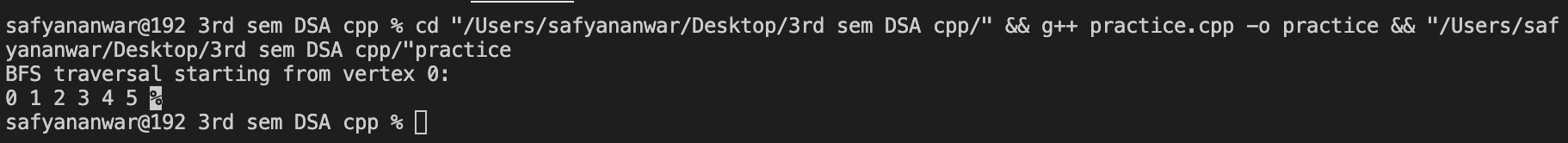
* **Class:** Represents a tree node with data, a pointer to the first child and a pointer to the next sibling.
* **Insert child:** Add a child node to a parent node. If the parent has no children, the new child becomes the first child; otherwise, it is added as the last sibling.
* **BFS function:** Perform a bfs traversal of the tree using a fixed-size array to simulate a queue. It prints the data of each node level by level.
* **Main function**: Creates a tree with a root node and several child nodes then call the BFS function to display the traversal order.

**Q4: “BFS Graph”:**

**Code:**

****

**Output:**

****

**Explanation:**

* **Graph class**: Represents a graph using an adjacency matrix to store connections between vertices.
* **Constructor:** Initializes the graph with a specified number of vertices and allocates memory for the adjacency matrix.
* **Destructor:** Frees the memory used by the adjacency matrix.
* addEdge Method: Adds an undirected edge between two vertices by updating the adjacency matrix.
* **BFS method:** Performs a breadth-first search starting from a specified vertex, using an array to simulate a queue and track visited vertices, printing the order of traversal.
* **Main function:** Creates a graph, adds edges, and calls the BFS method starting from vertex 0.
* **Note:** I used in graph classes operator ‘scope resolution DFS, BFS’. And I converted this lab # 13 in 4 parts.