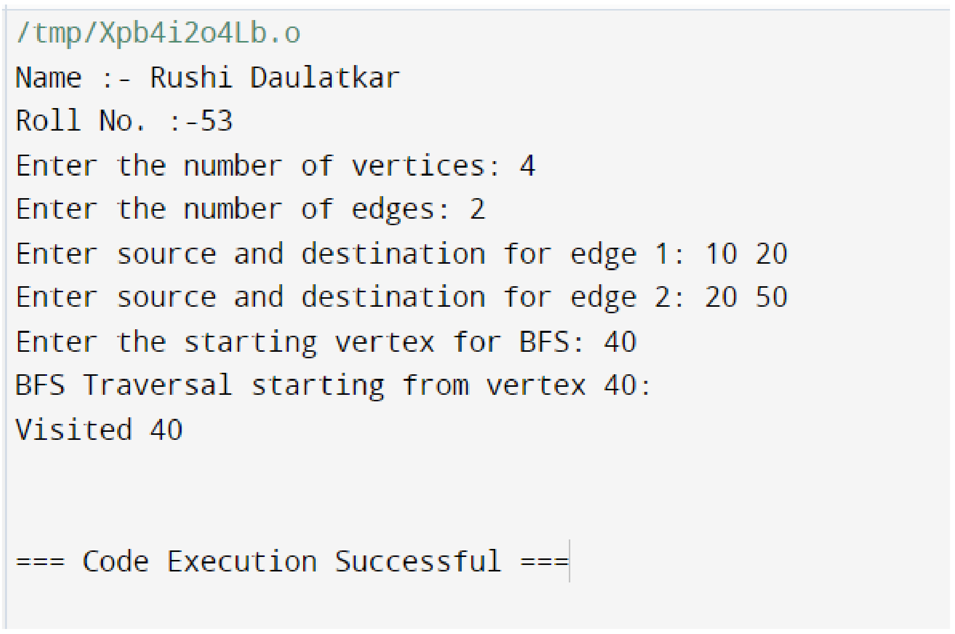
EXPERIMENT NO. 12 (B-1)

```
//The Code Is As Follows :-

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

// Define structure for a node in adjacency list
struct Node {
    int vertex;
    struct Node* next;
};

// Function declarations
void enqueue(struct Node**, int);
int isEmpty(struct Node*);
int dequeue(struct Node**);

// Define structure for graph
struct Graph {
    int numVertices;
    struct Node** adjLists;
    int* visited;
};
```

```
// Function to create a new node with given vertex
struct Node* createNode(int v) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

// Function to create a graph with given number of vertices
struct Graph* createGraph(int vertices) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numVertices = vertices;

    graph->adjLists = (struct Node**)malloc(vertices * sizeof(struct Node*));
    graph->visited = (int*)malloc(vertices * sizeof(int));

    for (int i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }

    return graph;
}

// Function to add an edge between two vertices
```

```
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to dest
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Add edge from dest to src (assuming undirected graph)
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// Function to perform BFS traversal starting from a given vertex
void bfs(struct Graph* graph, int startVertex) {
    // Initialize queue for BFS
    struct Node* queue = NULL;
    graph->visited[startVertex] = 1;
    enqueue(&queue, startVertex);

    while (!isEmpty(queue)) {
        int currentVertex = dequeue(&queue);
        printf("Visited %d\n", currentVertex);

        struct Node* temp = graph->adjLists[currentVertex];
        while (temp != NULL) {
            int adjVertex = temp->vertex;
            if (graph->visited[adjVertex] == 0) {
                graph->visited[adjVertex] = 1;
                enqueue(&queue, adjVertex);
            }
            temp = temp->next;
        }
    }
}

// Function to check if the queue is empty
int isEmpty(struct Node* queue) {
    return queue == NULL;
}

// Function to add a vertex to the queue
void enqueue(struct Node** queue, int value) {
    struct Node* newNode = createNode(value);
    if (isEmpty(*queue)) {
        *queue = newNode;
    } else {
        struct Node* temp = *queue;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

// Function to remove and return a vertex from the queue
int dequeue(struct Node** queue) {
    int nodeData = (*queue)->vertex;
    struct Node* temp = *queue;
    *queue = (*queue)->next;
    free(temp);
    return nodeData;
}

// Main function
int main() {
    printf("Name :- Rushi Daulatkar\n");
    printf("Roll No. :-53\n");
    int numVertices, numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

    // Create graph with given number of vertices
    struct Graph* graph = createGraph(numVertices);

    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    // Add edges
    for (int i = 0; i < numEdges; i++) {
        int src, dest;
        printf("Enter source and destination for edge %d: ", i + 1);
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }

    int startVertex;
    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &startVertex);

    // Perform BFS traversal
    printf("BFS Traversal starting from vertex %d:\n", startVertex);
    bfs(graph, startVertex);

    return 0;
}
```

OUTPUT:-



```
/tmp/Xpb4i2o4Lb.o
Name :- Rushi Daulatkar
Roll No. :-53
Enter the number of vertices: 4
Enter the number of edges: 2
Enter source and destination for edge 1: 10 20
Enter source and destination for edge 2: 20 50
Enter the starting vertex for BFS: 40
BFS Traversal starting from vertex 40:
Visited 40


=== Code Execution Successful ===
```