```
 #include <iostream>
 #include <cstdlib>

 using namespace std;

 const int MAX = 100;
 void fnBreadthFirstSearchReach(int vertex, int g[MAX][MAX], int
v[MAX], int n);

 class Queue

 {

    private:

        int cQueue[MAX];
        int front, rear;

    public:

        Queue();
        ~Queue();
        int enqueue(int data);
        int dequeue();
        int empty() { return front == -1 ? 1 : 0;  };
 };

/
**********************************************************************
**********
*Function   : main
*Input parameters: no parameters
*RETURNS    :   0 on success
**********************************************************************
**********/
int main(void)
{
    int i,j;
    int graph[MAX][MAX];
    int visited[MAX];
    int numVert;
    int startVert;

    cout << "Enter the number of vertices : ";
    cin >> numVert;

    cout << "Enter the adjacency matrix :\n";
    for (i=0; i < numVert; i++)
        visited[i] = 0;

    for (i=0; i<numVert; i++)
        for (j=0; j<numVert; j++)
            cin >> graph[i][j];
```

```
    cout << "Enter the starting vertex : ";
    cin >> startVert;

    fnBreadthFirstSearchReach(startVert-1,graph,visited,numVert);

    cout << "Vertices which can be reached from vertex " <<
startVert << " are :-" << endl;
    for (i=0; i<numVert; i++)
        if (visited[i])
            cout << i+1 << ", ";
    cout << endl;
    return 0;
}

/*Constructor*/
Queue::Queue()
{
    front = rear = -1;
}

/*Destructor*/
Queue::~Queue()
{
}


/
**********************************************************************
**********
*Function   : enqueue
*Description    : Function to insert an element at the rear of a
Queue
*Input parameters:
*   int data    - element to be inserted into the queue
*RETURNS        : returns 1 on success and 0 if queue is full
**********************************************************************
**********/

int Queue::enqueue(int data)

{
    if (front == (rear+1)%MAX)
        return 0;

    if (rear == -1)
        front = rear = 0;
    else
        rear = (rear+1)%MAX;

    cQueue[rear] = data;
    return 1;
}
/
**********************************************************************
```

```
**********
*Function   : dequeue
*Description    : Function to delete an element from the front of a
Queue
*Input parameters   : no parameters
*RETURNS        : returns element deleted on success and -1 if queue
is empty
**************************************************************************
**********/

int Queue::dequeue()

{
    int data;

    if (front == -1)
        return -1;

    data = cQueue[front];

    if (front == rear)
        front = rear = -1;
    else
        front = (front+1)%MAX;

    return data;
}
/
**************************************************************************
**********
*Function   : fnBreadthFirstSearchReach
*Description    : Function to perform BFS traversal and mark visited
vertices
*Input parameters:
*   int vertex  - source vertex
*   int g[][]   - adjacency matrix of the graph
*   int v[]     - vector to store visited information
    int n       - no of vertices
RETURNS     : void
**************************************************************************
**********/

void fnBreadthFirstSearchReach(int vertex, int g[MAX][MAX], int
v[MAX], int n)
{
    Queue verticesVisited;
    int frontVertex;
    int i;

    v[vertex] = 1;
    verticesVisited.enqueue(vertex);

    while (!verticesVisited.empty())
    {
```

```
        frontVertex = verticesVisited.dequeue();
        for (i=0; i<n; i++)
        {
            if (g[frontVertex][i] && !v[i])
            {
                v[i] = 1;
                verticesVisited.enqueue(i);
            }
        }
    }
}
```

OUTPUT

SAMPLE 1

```
Enter the number of vertices : 4

Enter the adjacency matrix :

0 1 1 0

1 0 0 1

1 0 0 1

0 1 1 0

Enter the starting vertex : 1

Vertices which can be reached from vertex 1 are :-

1, 2, 3, 4,
```

SAMPLE 2

```
Enter the number of vertices : 4

Enter the adjacency matrix :

0 1 0 0

1 0 0 0

0 0 0 1

0 0 1 0

Enter the starting vertex : 1

Vertices which can be reached from vertex 1 are :-

1, 2,
```

SAMPLE 3

Enter the number of vertices : 4

Enter the adjacency matrix :

0 1 0 0

0 0 1 0

0 0 0 1

0 0 0 0

Enter the starting vertex : 2

Vertices which can be reached from vertex 2 are :-

2, 3, 4,

SAMPLE 4

Enter the number of vertices : 4

Enter the adjacency matrix :

0 1 0 0

0 0 1 0

0 0 0 1

1 0 0 0

Enter the starting vertex : 2

Vertices which can be reached from vertex 2 are :-

1, 2, 3, 4,