Code Explanation and Help

i have viva and practical exam
if i provide you my code then can you help me by explaining the code line by line,also what is use of the code and how it works
and if possible can you give video to understand the concept in code

Absolutely, I can help you with that!

✅ Here's what I can do for you:

1. **Line-by-line explanation** of your code in simple terms.
2. **Explain what the code is used for**, its logic, and how it works step by step.
3. **If needed**, I can **draw diagrams** or flowcharts to explain the concept visually.
4. **Suggest relevant YouTube videos** that match the concept in your code so you can **watch and learn interactively**.

📌 Just upload or paste your code here (C++, Python, or Assembly — whichever you're working with).

Would you like to start with one specific program right now?

```
//C13
#include<iostream>
using namespace std;

class DFS {
     public:
   int top, f, r, x;
   int** adjList;
   int data[30], data1[30];
   int visit[20];
   int g[10][10];
   void create();
   void display();
   void createList();
   void displayList();
   void dfs();
   void bfs();

     DFS() {
     top = -1;
     f = r = -1;
     adjList = NULL;
```

```
    }

        int pop() {
        if(top != -1)
        {
            int y = data[top];
            top--;
            return y;
        }
        return -1;
    }

    void push(int t) {
        top++;
        data[top] = t;
    }

    void enqueue(int t) {
        if(f == -1 && r == -1)
        {
            f++;
            r++;
            data1[r] = t;
        }
        else
        {
            r++;
            data1[r] = t;
        }
    }

    int dequeue() {
        if(f == -1 && r == -1)
            return -1;
        else
        {
            int y = data1[f];
            if(f == r)
                f = r = -1;
            else
                f++;
            return y;
        }
    }
};

void DFS::create() {
    cout<<"Number of nodes:\t";
    cin>>x;
    for(int i = 0; i < x; i++)
    {
```

```
        for(int j = 0; j < x; j++)
        {
            cout<<endl<<"Enter link status of graph from node:\t";
            cin>>g[i][j];
        }
    }
}

void DFS::createList() {
    cout << "Number of nodes:\t";
    cin >> x;
    adjList = new int*[x];
    for (int i = 0; i < x; ++i)
    {
        adjList[i] = new int[x];
        for (int j = 0; j < x; ++j)
        {
            adjList[i][j] = 0;
        }
    }

    int connected, node;
    for (int i = 0; i < x; i++)
    {
        cout << "\nEnter number of nodes connected to node " << i << ": ";
        cin >> connected;
        cout << "\nEnter the nodes connected to node " << i << ": ";
        for (int j = 0; j < connected; j++)
        {
            cin >> node;
            adjList[i][node] = 1;
        }
    }
}

void DFS::displayList()
{
    for (int i = 0; i < x; i++)
    {
        cout << "\nNode " << i << " is connected to: ";
        for (int j = 0; j < x; j++)
        {
            if (adjList[i][j] == 1)
            {
                cout << j << " ";
            }
        }
    }
    cout<<"\n";
}
```

```
void DFS::display()
{
    cout<< "   ";
    for (int i = 0; i < x; i++)
    {
        cout<<" "<<i;
    }
    cout<<"\n";
    for (int i = 0; i < x; i++)
    {
        cout<<i<<" |";
        for (int j = 0; j < x; j++)
        {
            cout<<" "<< g[i][j];
        }
        cout<<"\n";
    }
}

void DFS::dfs()
{
    for(int i = 0; i < x; i++)
        visit[i] = 0;
    DFS s;
    int v1;
    cout<<"\nEnter starting node: ";
    cin>>v1;
    s.push(v1);
    cout<<"DFS traversal is: ";
    while(s.top != -1)
    {
        int v = s.pop();
        if(visit[v] == 0)
        {
            cout<<" "<<v;
            visit[v] = 1;
            for(int i = x-1; i > -1; i--)
            {
                if(g[v][i] == 1 && visit[i] == 0)
                {
                    s.push(i);
                }
            }
        }
    }
}

void DFS::bfs()
{
    for(int i = 0; i < x; i++)
        visit[i] = 0;
```

```
    DFS s;
    int v1;
    cout<<"\nEnter starting node: ";
    cin>>v1;
    s.enqueue(v1);
    cout<<"\nBFS traversal is: ";
    while(s.f != -1 && s.r != -1)
    {
        int v = s.dequeue();
        if(visit[v] == 0)
        {
            cout<<" "<<v;
            visit[v] = 1;
            for(int i = 0; i < x; i++)
            {
                if(adjList[v][i] == 1 && visit[i] == 0)
                {
                    s.enqueue(i);
                }
            }
        }
    }
    cout<<"\n";
}

int main()
{
    DFS obj;
    bool flag = true;
    int choice;
    while(flag)
    {
        cout<<"\n***YOUR CHOICES ARE****\n";
        cout<<"\n1. Create Graph (Matrix) \n2. DFS Traversal (Using Matrix) \n3. Create Graph (List) \n4.
BFS Traversal (Using List) \n5. Exit";
        cout<<"\nEnter choice: ";
        cin>>choice;
        switch(choice)
        {
        case 1:
            obj.create();
            obj.display();
            break;

        case 2:
            obj.dfs();
            break;

        case 3:
            obj.createList();
            obj.displayList();
```

```
            break;

        case 4:
            obj.bfs();
            break;

        case 5:
            flag = false;
            break;

        default:
            cout<<"\nEnter Valid Choice!";
            break;
        }
    }
    return 0;
}
```

Let's go through the code line by line and understand how it works:

Class Definition: DFS

```cpp
class DFS {
    public:
    int top, f, r, x;
    int** adjList;
    int data[30], data1[30];
    int visit[20];
    int g[10][10];
    void create();
    void display();
    void createList();
    void displayList();
    void dfs();
    void bfs();
```

- This defines a class `DFS` which encapsulates the functionality for both **Depth First Search (DFS)** and **Breadth First Search (BFS)**, as well as graph creation and display.
- **Member variables:**
  - `top`, `f`, `r` - These variables are used for stack and queue operations.
  - `adjList` - A pointer to the adjacency list (dynamic array).
  - `data`, `data1` - Arrays used for stack and queue data storage.
  - `visit` - An array that tracks visited nodes.
  - `g` - A 2D array representing the adjacency matrix of the graph.

Constructor

```cpp
DFS() {
    top = -1;
    f = r = -1;
    adjList = NULL;
}
```

- **Constructor**: Initializes `top`, `f`, `r` to `-1`, which indicates an empty stack and queue. It also sets `adjList` to `NULL` (no adjacency list).

Stack Operations: `pop()` and `push()`

```cpp
int pop() {
    if(top != -1) {
        int y = data[top];
        top--;
        return y;
    }
    return -1;
}

void push(int t) {
    top++;
    data[top] = t;
}
```

- `pop()` removes and returns the top element from the stack.
- `push(t)` adds an element `t` to the stack.

Queue Operations: `enqueue()` and `dequeue()`

```cpp
void enqueue(int t) {
    if(f == -1 && r == -1) {
        f++;
        r++;
        data1[r] = t;
    } else {
        r++;
        data1[r] = t;
    }
}

int dequeue() {
    if(f == -1 && r == -1) return -1;
    else {
        int y = data1[f];
        if(f == r) f = r = -1;
        else f++;
        return y;
    }
}
```

- `enqueue(t)` adds an element `t` to the queue.
- `dequeue()` removes and returns an element from the front of the queue.

Graph Creation: `create()`

```cpp
void create() {
    cout<<"Number of nodes:\t";
    cin>>x;
    for(int i = 0; i < x; i++) {
        for(int j = 0; j < x; j++) {
            cout<<endl<<"Enter link status of graph from node:\t";
            cin>>g[i][j];
        }
    }
}
```

- **Graph creation**: This function creates an adjacency matrix representation of the graph.
- The user is prompted to input the number of nodes and then the status of the edges between nodes (either 0 for no edge or 1 for an edge).

Adjacency List Creation: `createList()`

```cpp
void createList() {
    cout << "Number of nodes:\t";
    cin >> x;
    adjList = new int*[x];
    for (int i = 0; i < x; ++i) {
        adjList[i] = new int[x];
        for (int j = 0; j < x; ++j) {
            adjList[i][j] = 0;
        }
    }

    int connected, node;
    for (int i = 0; i < x; i++) {
        cout << "\nEnter number of nodes connected to node " << i << ": ";
        cin >> connected;
        cout << "\nEnter the nodes connected to node " << i << ": ";
        for (int j = 0; j < connected; j++) {
            cin >> node;
            adjList[i][node] = 1;
        }
    }
}
```

- **Adjacency List creation**: This function creates an adjacency list representation of the graph.
- The user is asked to input how many nodes are connected to each node and the specific nodes connected to each.

Display Adjacency List: `displayList()`

```cpp
void displayList() {
    for (int i = 0; i < x; i++) {
        cout << "\nNode " << i << " is connected to: ";
        for (int j = 0; j < x; j++) {
            if (adjList[i][j] == 1) {
                cout << j << " ";
            }
```

```
        }
    }
    cout<<"\n";
}
```

- This function displays the adjacency list, showing the nodes connected to each node.

Display Matrix: `display()`

```cpp
void display() {
    cout<< "    ";
    for (int i = 0; i < x; i++) {
        cout<<" "<<i;
    }
    cout<<"\n";
    for (int i = 0; i < x; i++) {
        cout<<i<<" |";
        for (int j = 0; j < x; j++) {
            cout<<" "<< g[i][j];
        }
        cout<<"\n";
    }
}
```

- Displays the adjacency matrix representation of the graph.

Depth First Search (DFS): `dfs()`

```cpp
void dfs() {
    for(int i = 0; i < x; i++) visit[i] = 0;
    DFS s;
    int v1;
    cout<<"\nEnter starting node: ";
    cin>>v1;
    s.push(v1);
    cout<<"DFS traversal is: ";
    while(s.top != -1) {
        int v = s.pop();
        if(visit[v] == 0) {
            cout<<" "<<v;
            visit[v] = 1;
            for(int i = x-1; i > -1; i--) {
                if(g[v][i] == 1 && visit[i] == 0) {
                    s.push(i);
                }
            }
        }
    }
}
```

- **DFS Traversal**:
    - Initializes the `visit[]` array to 0 (not visited).
    - The user provides a starting node `v1`.
    - The algorithm uses a stack to explore the graph in a depth-first manner.
    - The stack ensures that each node is explored before moving to adjacent nodes.

Breadth First Search (BFS): `bfs()`

```cpp
void bfs() {
    for(int i = 0; i < x; i++) visit[i] = 0;
    DFS s;
    int v1;
    cout<<"\nEnter starting node: ";
    cin>>v1;
    s.enqueue(v1);
    cout<<"\nBFS traversal is: ";
    while(s.f != -1 && s.r != -1) {
        int v = s.dequeue();
        if(visit[v] == 0) {
            cout<<" "<<v;
            visit[v] = 1;
            for(int i = 0; i < x; i++) {
                if(adjList[v][i] == 1 && visit[i] == 0) {
                    s.enqueue(i);
                }
            }
        }
    }
    cout<<"\n";
}
```

- **BFS Traversal**:
    - Similar to DFS, it initializes the `visit[]` array.
    - The starting node `v1` is enqueued and then the BFS explores the graph level by level using a queue.
    - Nodes are dequeued and each unvisited adjacent node is enqueued.

Main Program:

```cpp
int main() {
    DFS obj;
    bool flag = true;
    int choice;
    while(flag) {
        cout<<"\n***YOUR CHOICES ARE****\n";
        cout<<"\n1. Create Graph (Matrix) \n2. DFS Traversal (Using Matrix) \n3. Create Graph (List
        cout<<"\nEnter choice: ";
        cin>>choice;
        switch(choice) {
            case 1:
                obj.create();
                obj.display();
                break;
            case 2:
                obj.dfs();
                break;
            case 3:
                obj.createList();
                obj.displayList();
                break;
            case 4:
                obj.bfs();
                break;
            case 5:
                flag = false;
                break;
            default:
```

```
                  cout<<"\nEnter Valid Choice!";
                  break;
          }
      }
      return 0;
}
```

- The `main()` function creates an object `obj` of the `DFS` class.
- It runs a loop where the user can choose:
  - Create and display graph (Matrix/List).
  - Perform DFS/BFS traversal.
  - Exit the program.

Summary of the Code:

1. **Graph Representation**: It supports both **adjacency matrix** and **adjacency list** representations.
2. **DFS & BFS Traversals**: Implements both **Depth First Search (DFS)** using a stack and **Breadth First Search (BFS)** using a queue.
3. **Interactive Menu**: The program is interactive, allowing the user to input nodes and edges and choose traversal types.

For Video Resources:

Here are some YouTube video suggestions to understand these concepts better:

- DFS and BFS Traversal — A good explanation of DFS and BFS.
- Graph Representation: Matrix vs List — Explains the differences between matrix and list representations.

Would you like more clarification on any specific part?