

# Code Explanation Assistance



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

Yes, I can definitely help you with explaining your code line by line, as well as provide insights into how it works and its usage. If you share your code, I will break it down and explain each part in detail.

As for providing a video, I don't have the ability to generate or provide videos directly, but I can suggest resources or platforms where you can find videos that will help you understand the concepts in your code.

Feel free to upload your code or share details about the problem you're working on, and I'll guide you through it!



```
#include<iostream>
#include<string>
#include <list>
using namespace std;

class Node
{
public :
Node *next;
int priority;
string data;
Node(string d,int prior){
    priority = prior;
    data = d;
    next = NULL;
}
};

class PriorityQueue{
public:
Node *front=NULL;

void insert(string d,int prior){
    Node *temp,*rear;
    temp = new Node(d,prior);
    if(front == NULL){
```

```

        front = temp;
    }
    else
    {
        rear = front;
        while(
            rear->next!=NULL &&
            rear->next->priority >= prior
        ){
            rear=rear->next;
        }
        temp->next = rear->next;
        rear->next = temp;
    }

}

void peek(){
    cout<<"First patient is:\t"<<front->data;
}

void pop(){
    if(front==NULL)
        return;
    front=front->next;
}

void dis()
{
    string currPrior="";
    if(front== NULL){
        cout<<endl<<"Empty queue."<<endl;
        return;
    }
    cout<<endl;
    Node *curr= front;
    while(curr!=NULL){
        //hardCode the category
        if(curr->priority!=0){
            if(curr->priority==3)
                currPrior="Serious patient";
            else if(curr->priority==2)
                currPrior="Not serious patient";
            else
                currPrior="General checkup";
        }
        cout<<curr->data<<" with priority:\t"<<currPrior<<endl;
        curr=curr->next;
    }
}

};

```

```
int main(){

    string name;
    int priority,ch;

    cout<<endl;
    PriorityQueue q;
    do{
        cout<<endl<<"--- MAIN MENU ---";
        cout<<endl<<"1 -> Add patient";
        cout<<endl<<"2 -> Remove patient";
        cout<<endl<<"3 -> Get all patients";
        cout<<endl<<"0 -> Exit";
        cout<<endl<<"Choose an option (0-3):\t";
        cin>>ch;

        switch (ch)
        {
            case 1:
            {
                cout<<"Patient name is:\t";
                cin.ignore();
                getline(cin,name,'\n');
                cout<<endl<<"Enter priority (3-High, 2-Medium, 1-General):\t";
                cin>>priority;
                q.insert(name,priority);
                break;
            }
            case 2:
            {
                q.pop();
                break;
            }
            case 3:{
                q.dis();
                break;
            }
            case 0:
                cout<<endl<<"// END OF CODE\n\n";
                exit(1);
                break;

            default:
                break;
        }
    } while(ch!=0);
}
```

This code implements a **Priority Queue** system for managing patients based on their medical priority, using a **linked list** structure. Let's break down the code and explain how it works:

## Classes:

### 1. Node Class:

- Each `Node` object represents a patient and their priority in the queue.
- **Attributes:**
  - `next`: Points to the next node in the linked list.
  - `priority`: Holds the priority value of the patient.
  - `data`: Stores the name of the patient.
- **Constructor:** Initializes `priority`, `data`, and sets `next` to `NULL` initially.

```
cpp
class Node {
public:
    Node* next;
    int priority;
    string data;
    Node(string d, int prior) {
        priority = prior;
        data = d;
        next = NULL;
    }
};
```

### 2. PriorityQueue Class:

- This class represents the priority queue which will handle patients.
- **Attributes:**
  - `front`: Points to the front of the priority queue (linked list).
- **Methods:**
  - `insert(string d, int prior)`: Inserts a new patient in the queue based on their priority.
  - `peek()`: Displays the first patient in the queue.
  - `pop()`: Removes the patient at the front of the queue.
  - `dis()`: Displays all patients with their names and priorities.

## `PriorityQueue` Methods Explained:

### 1. Insert Method:

- A new patient is created using the `Node` constructor.
- If the queue is empty, the new node becomes the front.
- If the queue is not empty, the method traverses the list to find the correct place where the patient's priority is greater than or equal to the next patient's priority.

- After finding the correct spot, the new node is inserted accordingly.

```
cpp

void insert(string d, int prior) {
    Node* temp, *rear;
    temp = new Node(d, prior);

    if (front == NULL) {
        front = temp; // If queue is empty, new node is the first element.
    } else {
        rear = front;
        while (rear->next != NULL && rear->next->priority >= prior) {
            rear = rear->next; // Traverse until correct priority place is found.
        }
        temp->next = rear->next;
        rear->next = temp; // Insert new patient in the correct place.
    }
}
```

## 2. Peek Method:

- Displays the first patient in the queue.
- This is essentially checking the "highest priority" patient at the front.

```
cpp

void peek() {
    cout << "First patient is:\t" << front->data;
}
```

## 3. Pop Method:

- Removes the first patient from the queue by adjusting the `front` pointer to the next patient in the queue.

```
cpp

void pop() {
    if (front == NULL) {
        return; // If the queue is empty, no patient to remove.
    }
    front = front->next; // Remove the front patient by updating the front pointer.
}
```

## 4. Display Method (dis):

- Prints out the details of all patients in the queue.
- For each patient, it also prints their priority category (Serious, Not serious, General checkup) based on their priority value.

```
cpp

void dis() {
    string currPrior = "";
    if (front == NULL) {
        cout << endl << "Empty queue." << endl;
        return;
    }
    cout << endl;
```

```
Node* curr = front;
while (curr != NULL) {
    if (curr->priority != 0) {
        if (curr->priority == 3)
            currPrior = "Serious patient";
        else if (curr->priority == 2)
            currPrior = "Not serious patient";
        else
            currPrior = "General checkup";
    }
    cout << curr->data << " with priority:\t" << currPrior << endl;
    curr = curr->next; // Move to next patient in the queue.
}
}
```

## `Main` Function (Driver Code):

- The `main` function is responsible for interacting with the user.
- The user can add patients, remove patients, or view the current patients and their priorities by choosing the appropriate option from the menu.
- **Switch Statement:**
  - **Case 1:** Adds a patient by asking for their name and priority.
  - **Case 2:** Removes the highest-priority patient.
  - **Case 3:** Displays all patients and their priorities.
  - **Case 0:** Exits the program.

cpp

```
int main() {
    string name;
    int priority, ch;

    PriorityQueue q;
    do {
        cout << endl << "--- MAIN MENU ---";
        cout << endl << "1 -> Add patient";
        cout << endl << "2 -> Remove patient";
        cout << endl << "3 -> Get all patients";
        cout << endl << "0 -> Exit";
        cout << endl << "Choose an option (0-3):\t";
        cin >> ch;

        switch (ch) {
            case 1: {
                cout << "Patient name is:\t";
                cin.ignore();
                getline(cin, name, '\n');
                cout << endl << "Enter priority (3-High, 2-Medium, 1-General):\t";
                cin >> priority;
                q.insert(name, priority); // Insert patient
                break;
            }
            case 2:
                q.pop(); // Remove patient
                break;
            case 3:
                q.dis(); // Display patients
                break;
            case 0:
                cout << endl << "// END OF CODE\n\n";
                exit(1); // Exit the program
                break;
            default:

```

```

        break;
    }
} while (ch != 0);
}

```

## How It Works:

- This code simulates a **priority-based queue** where patients with higher priority (e.g., priority 3) are handled first.
- It uses a linked list, where each patient is represented as a node.
- The queue maintains a **sorted order** by priority (highest priority at the front).
- The priority queue supports operations like adding a patient, removing the highest-priority patient, and displaying all patients in the queue.

## Next Steps:

- You can run the program, input patient names and priorities, and interact with the menu to see how the queue behaves.
- To visualize the priority queue's behavior, you can create test cases with different priority values and observe how patients are added and removed.

Let me know if you'd like any further clarification on any specific part!