The University of Azad Jammu and Kashmir

Name: Zarnab Fatima

Roll NO: 2024-SE-14

Course Title: Data Structure & Algorithm

Course Code: CS-2101

Submitted To: Engr. Sidra Rafique

Department of Software Engineering

LAB 03

Abstract Data Type

ADT (Abstract Data Type):

ADT (Abstract Data Type) is a logical description of how data is organized and what operations can be performed on it, without specifying how it's implemented in code.

Key Points:

It hides implementation details (abstraction).

Defines only what operations are available (like insert, delete, search).

Examples: Stack, Queue, List, Tree, Graph.

Example:

Stack ADT supports:

push(): Add element

pop(): Remove top element

peek(): View top element

isEmpty(): Check if stack is empty

TASK:

Design and implement an Abstract Data Type (ADT) for a real-world system or application of your choice. Explain the operation supported by your ADT and how it demonstrates the principles of abstractions and encapsulation in context of DS?

Chosen ADT: PATIENT QEUE

This ADT manages the queue of patients waiting to be seen by a doctor.

Operations Supported:

1. enqueue(patient)

Adds a new patient to the end of the queue.

2. dequeue()

Removes the patient at the front of the queue (next to be treated).

3. peek()

Returns details of the next patient without removing them.

4. isEmpty()

Checks if the queue is empty.

5. getSize()

Returns the number of patients currently in the queue.

How It Demonstrates DS Principles:

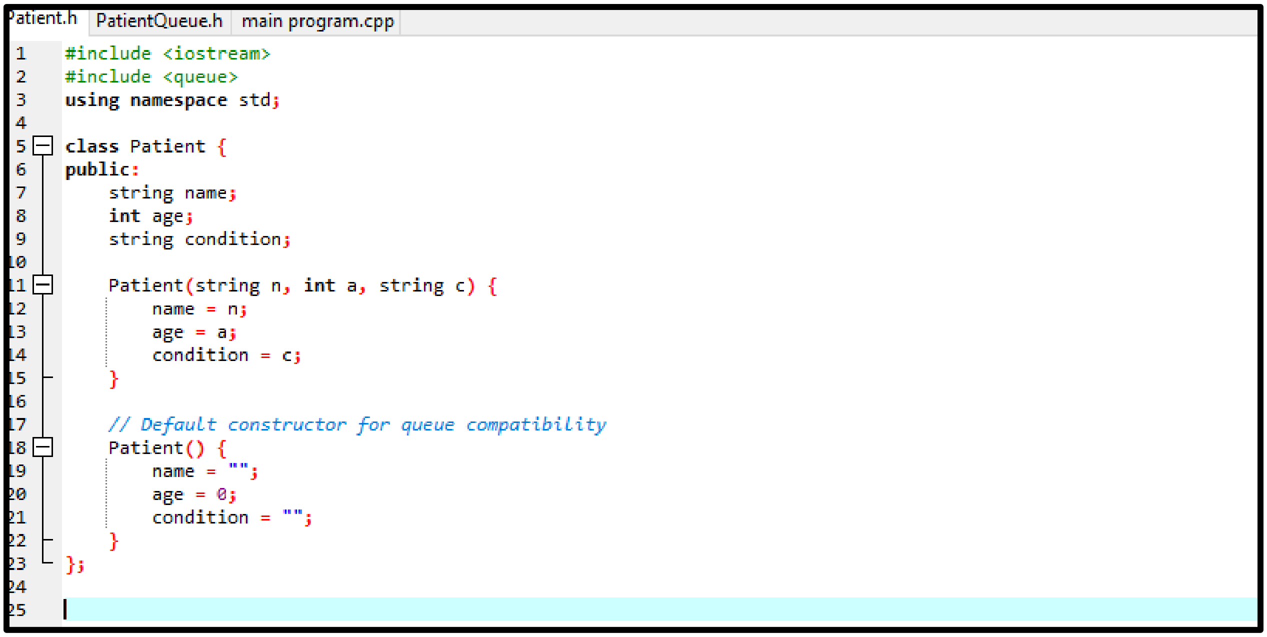
1. Abstraction:

- The internal working (e.g., how the queue is stored—array or linked list) is hidden from the user.
- The user only interacts with simple, meaningful operations like enqueue() or peek().

2. Encapsulation:

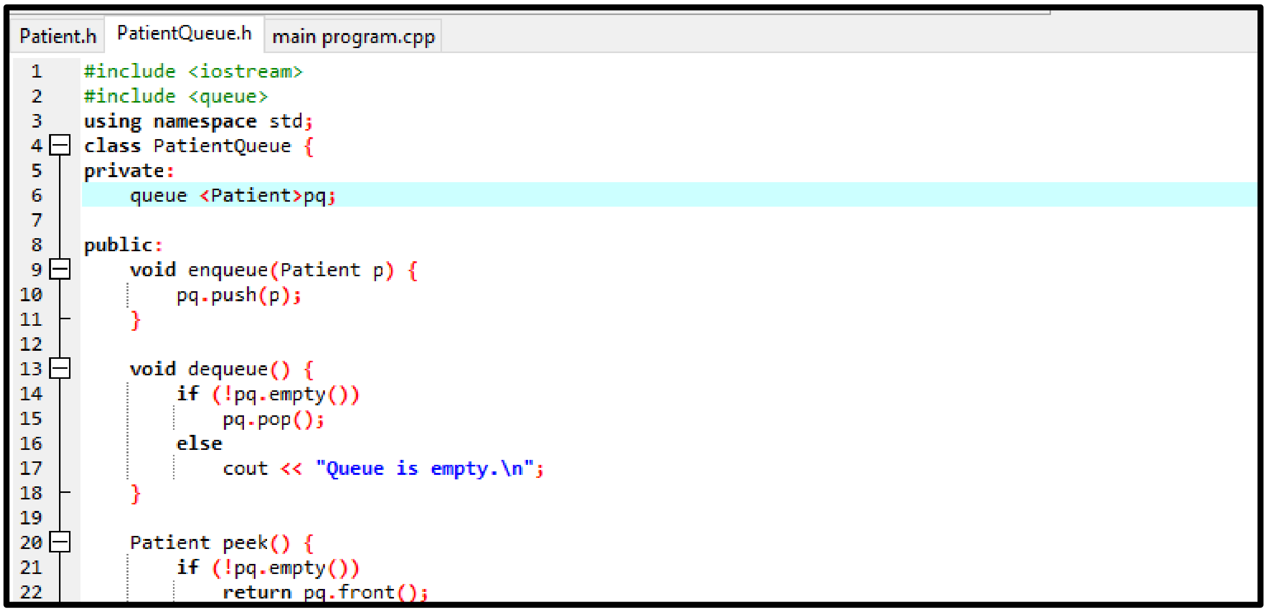
- Data (patient list) is encapsulated within the class.
- Access is only possible through public methods, preventing external code from directly modifying the queue structure.

Example in C++



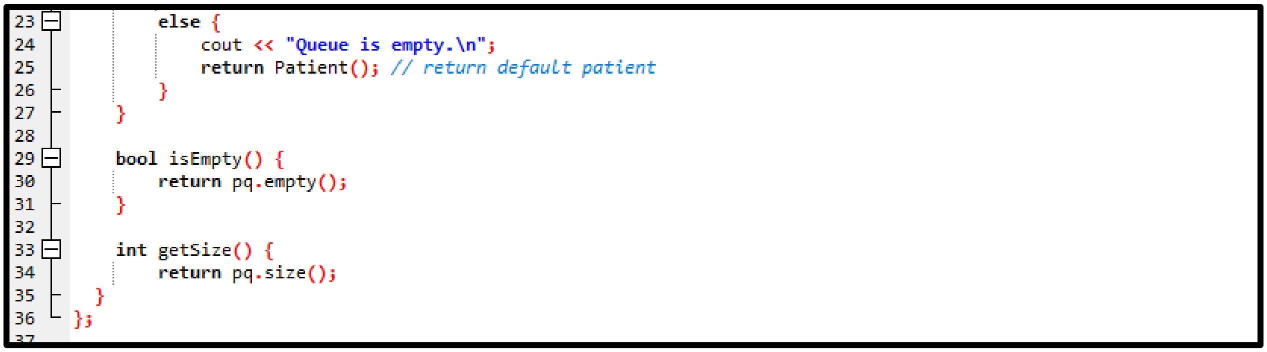
```cpp
#include <iostream>
#include <queue>
using namespace std;

class Patient {
public:
    string name;
    int age;
    string condition;

    Patient(string n, int a, string c) {
        name = n;
        age = a;
        condition = c;
    }

    // Default constructor for queue compatibility
    Patient() {
        name = "";
        age = 0;
        condition = "";
    }
};

```

Figure 1: Patient Class



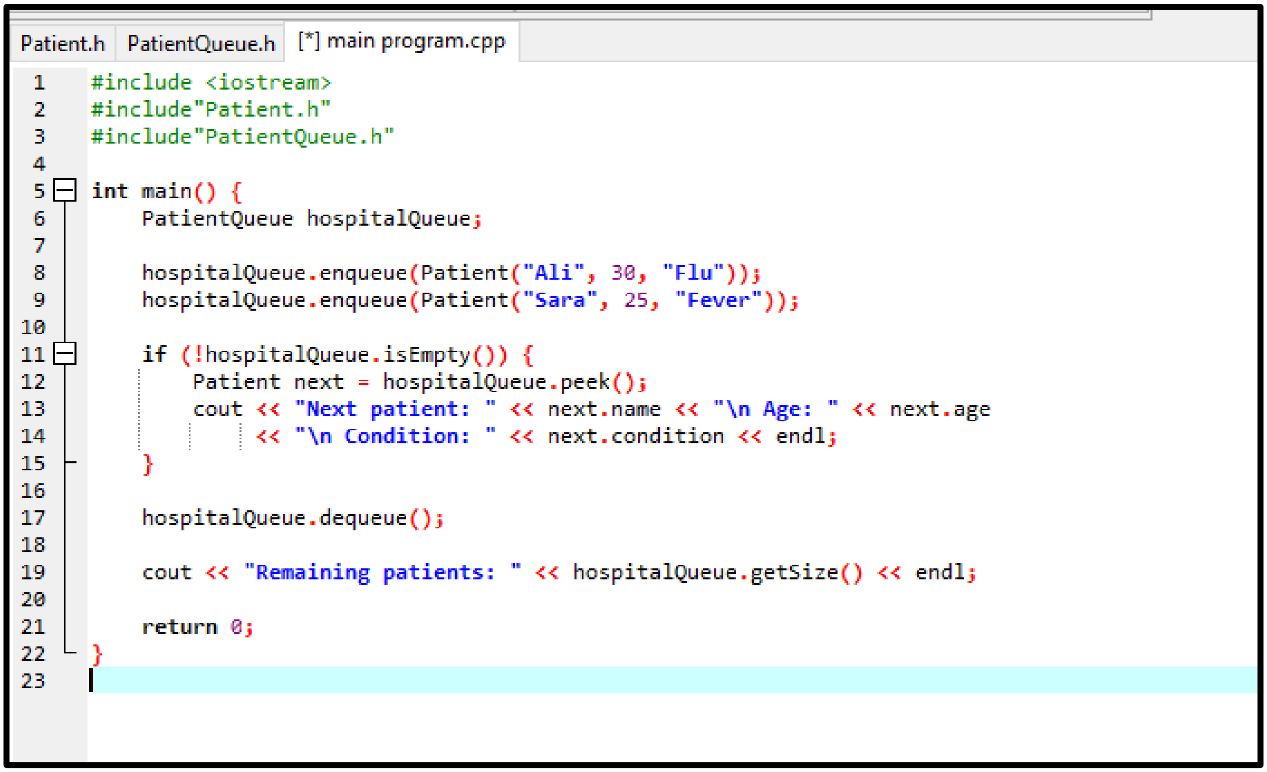
```cpp
#include <iostream>
#include <queue>
using namespace std;
class PatientQueue {
private:
    queue <Patient>pq;

public:
    void enqueue(Patient p) {
        pq.push(p);
    }

    void dequeue() {
        if (!pq.empty())
            pq.pop();
        else
            cout << "Queue is empty.\n";
    }

    Patient peek() {
        if (!pq.empty())
            return pq.front();
```

Figure 2: PatientQueue Class



```
        else {
            cout << "Queue is empty.\n";
            return Patient(); // return default patient
        }
    }

    bool isEmpty() {
        return pq.empty();
    }

    int getSize() {
        return pq.size();
  }
};
```

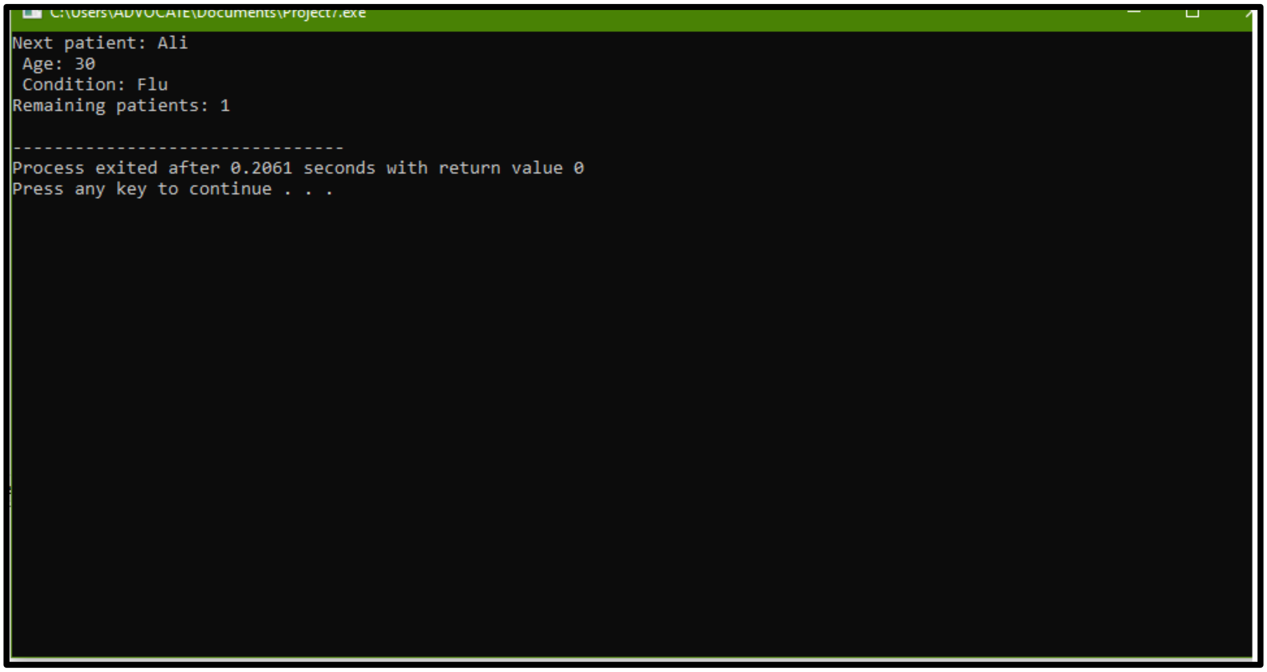
Figure 3



```
#include <iostream>
#include"Patient.h"
#include"PatientQueue.h"

int main() {
    PatientQueue hospitalQueue;

    hospitalQueue.enqueue(Patient("Ali", 30, "Flu"));
    hospitalQueue.enqueue(Patient("Sara", 25, "Fever"));

    if (!hospitalQueue.isEmpty()) {
        Patient next = hospitalQueue.peek();
        cout << "Next patient: " << next.name << "\n Age: " << next.age
             << "\n Condition: " << next.condition << endl;
    }

    hospitalQueue.dequeue();

    cout << "Remaining patients: " << hospitalQueue.getSize() << endl;

    return 0;
}
```

Figure 4: Main Program

Output:



```
Next patient: Ali
 Age: 30
 Condition: Flu
Remaining patients: 1

--------------------------------
Process exited after 0.2061 seconds with return value 0
Press any key to continue . . .
```

Figure 5:Output of Program

Explanation:

Patient Queue Management System using C++

This C++ program simulates a simple hospital queue system using Object-Oriented Programming. It defines a Patient class to store patient information (name, age, condition) and a PatientQueue class that uses a queue to manage patients in First-In-First-Out (FIFO) order.

The system allows:

Adding (enqueueing) new patients.

Viewing the next patient to be treated.

Removing (dequeueing) the treated patient.

Checking if the queue is empty.

Counting the number of patients in the queue.

The program demonstrates abstraction and encapsulation by hiding the queue logic inside the PatientQueue class, making the system modular and easy to maintain.

*****THANK YOU*****