

Semester Project: Computer Programming Lab

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Course Title:** Computer Programming  
**Faculty Name:** Mohsin Javaid Butt  
**Semester:** Spring 2025  
**Deadline:** 22 May 2024  
**Group Members:** Two

|  |  |
| --- | --- |
| Member1 | Member2 |
| Name: Binyameen | Name: Fahad Ahmed |
| Enrollment #: 01-134251-015 | Enrollment #: 01-134251-019 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Project Name: Hospital Management System

Table of Contents

[Introduction: 1](#_Toc198824822)

[Design: 2](#_Toc198824823)

[Implementation: 3](#_Toc198824824)

[Evaluation: 4](#_Toc198824825)

[Diagrams: 5](#_Toc198824826)

[Code snippets: 6](#_Toc198824827)

[Conclusion reflecting the Project Outcomes: 7](#_Toc198824828)

# Introduction:

This project is a simple Hospital Management System (HMS) implemented in C++. The purpose of the system is to provide an efficient way of managing patient information, doctors selection, and medical billing. It is designed to handle up to 100 patients and 3 doctors in each of four medical fields: Radiology, Oncology, Dermatology, and General Medicine. The system also allows users to calculate bills based on the doctor's consultation fee and any additional charges. The program allows user interaction via a text-based menu, making it simple to navigate and use.

**Key features include:**

1. **Patient Management**: Adds patient information, including their name, age, department, doctor selection, gender, and calculates their total bill.
2. **Doctor Information**: Each department has a predefined set of doctors, each with their consultation fee.
3. **Billing System**: The system calculates the total bill, including taxes and additional services if applicable.
4. **File Handling**: Patient data is saved in a text file for persistence.

# Design:

**System Architecture**

The system uses an array-based design for storing data about patients, doctors, and related information. The main components are:

1. **Patient Data**: Includes name, age, department, doctor, gender, and bill amount.
2. **Doctor Data**: Stores doctor names, fields, and charges for each department.
3. **Operations**: Functions such as adding a patient, viewing patients, calculating bills, and displaying the menu.
4. **File Storage**: The patient data is written to a file (patient\_records.txt) for persistence.

**Data Flow**

1. **Add Patient**: The user enters the patient’s information, selects a department, chooses a doctor, and calculates the bill.
2. **View Patients**: The system displays all registered patients.
3. **Calculate Bill**: The user can retrieve a specific patient's bill based on their ID.

**User Interface**

The system is a command-line interface that prompts the user for various inputs, such as patient details, department choice, doctor selection, and more.

**Use of Constants**

* maxPatient: Defines the maximum number of patients that can be stored (100).
* maxDoctor: Limits the number of doctors available in each department (3).

# Implementation:

The code consists of several functions and arrays for storing data related to patients and doctors. Below are key sections of the code:

**Data Structures for Storing Information**

// Patient information storage

string patientName[maxPatient];

int patientAge[maxPatient];

int patientId[maxPatient];

string patientDepartment[maxPatient];

string patientDoctor[maxPatient];

float patientBill[maxPatient];

string patientGender[maxPatient];

int patientCount = 0;

// Doctor information for multiple departments

string radiologistName[maxDoctoer] = {"Dr Fahad", "Dr Yameen", "Dr Hamza"};

string radiologistFeild[maxDoctoer] = {"Radiology", "Radiology", "Radiology"};

int radiologistCharges[maxDoctoer] = {250, 340, 560};

**Function to Add a Patient**

This function handles the input of a new patient's information, including their doctor and department choice, and calculates the associated costs.

void add\_patient() {

if (patientCount >= maxPatient) {

cout << "Sorry you can't enter further patients." << endl;

return;

}

cout << "Please enter patient name: ";

cin >> patientName[patientCount];

cout << "Please enter patient age: ";

cin >> patientAge[patientCount];

// Additional inputs omitted for brevity...

// Select Doctor and Calculate Charges

doctor\_Options(radiologistName, radiologistFeild, radiologistCharges, maxDoctoer);

// Charge calculation logic...

}

**Doctor Selection**

The doctor\_Options function displays available doctors based on the selected department and allows the user to choose a doctor.

void doctor\_Options(string doctorName[], string doctorField[], int doctorCharges[], int doctorCount = 3) {

cout << "Select the doctors from available options:" << endl;

for (int i = 0; i < doctorCount; i++) {

cout << i + 1 << " Name: " << doctorName[i] << " Field: " << doctorField[i] << " Charges: " << doctorCharges[i] << endl;

}

}

**Bill Calculation**

This function calculates the total bill, considering doctor charges, additional services, and tax.

float total\_charge = doctorCharge + additional\_charges;

float tax\_rate = 0.10;

float tax\_amount = total\_charge \* tax\_rate;

float total\_bill = total\_charge + tax\_amount;

**File Handling**

The patient data is saved into a file (patient\_records.txt) for persistence. This ensures that patient records are not lost after the program ends.

ofstream outFile("patient\_records.txt", ios::app);

if (outFile.is\_open()) {

outFile << "ID: " << patientId[patientCount]

<< ", Name: " << patientName[patientCount]

<< ", Age: " << patientAge[patientCount]

<< ", Gender: " << patientGender[patientCount]

<< ", Department: " << patientDepartment[patientCount]

<< ", Doctor: " << patientDoctor[patientCount]

<< ", Bill: $" << patientBill[patientCount] << "\n";

outFile.close();

}

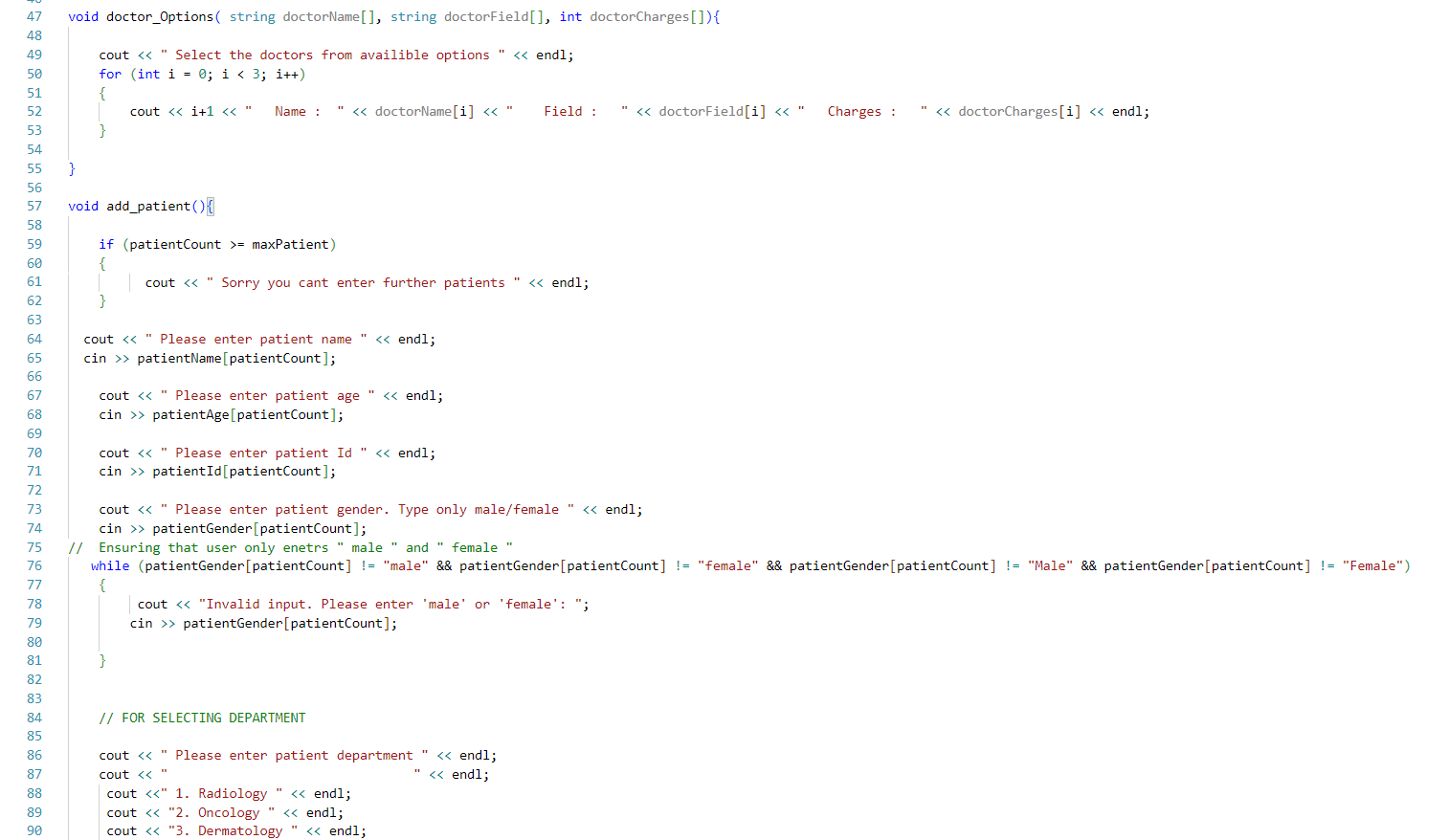
# Evaluation:

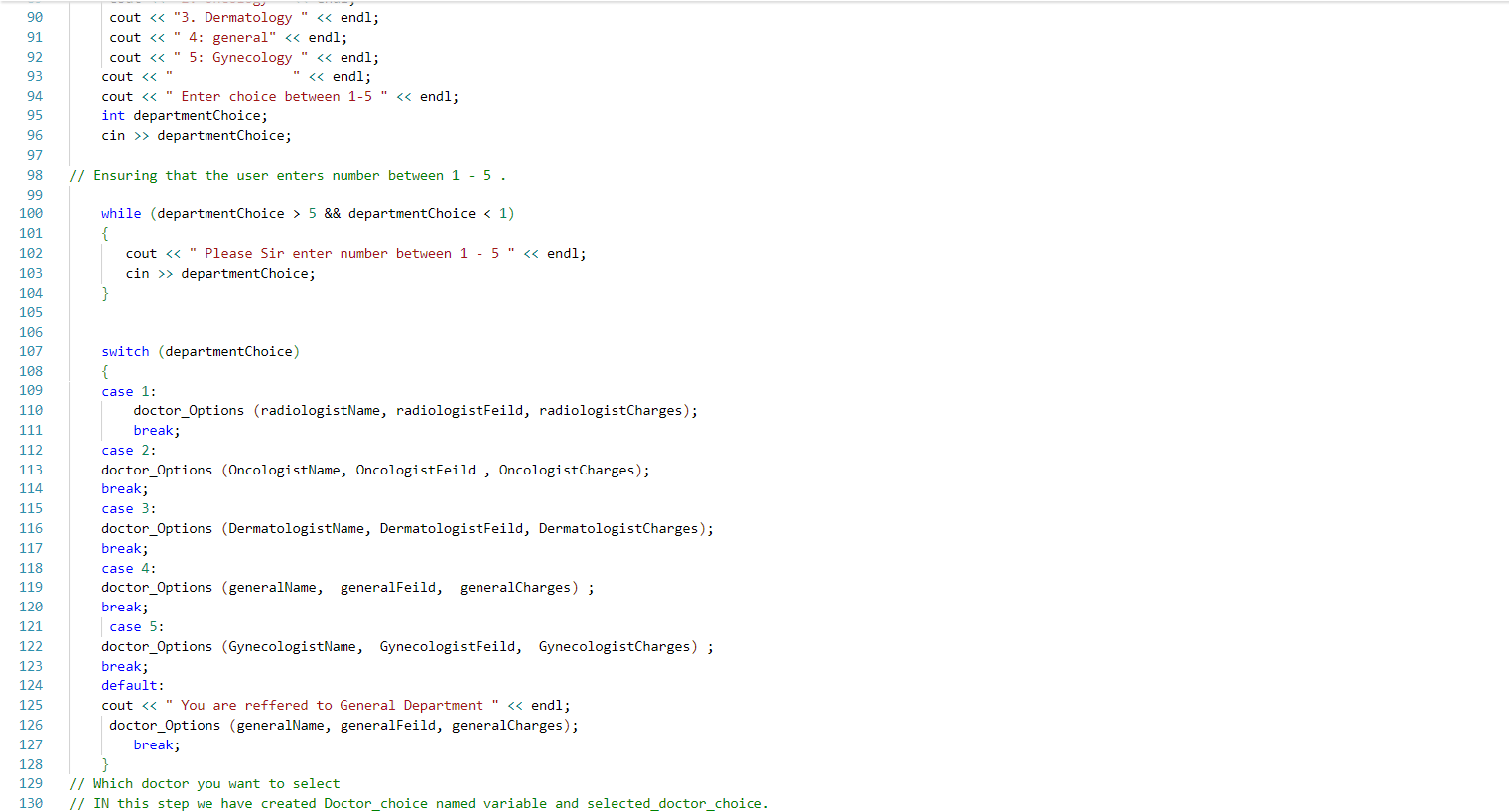
The system was evaluated in terms of functionality, usability, and reliability. The following aspects were considered:

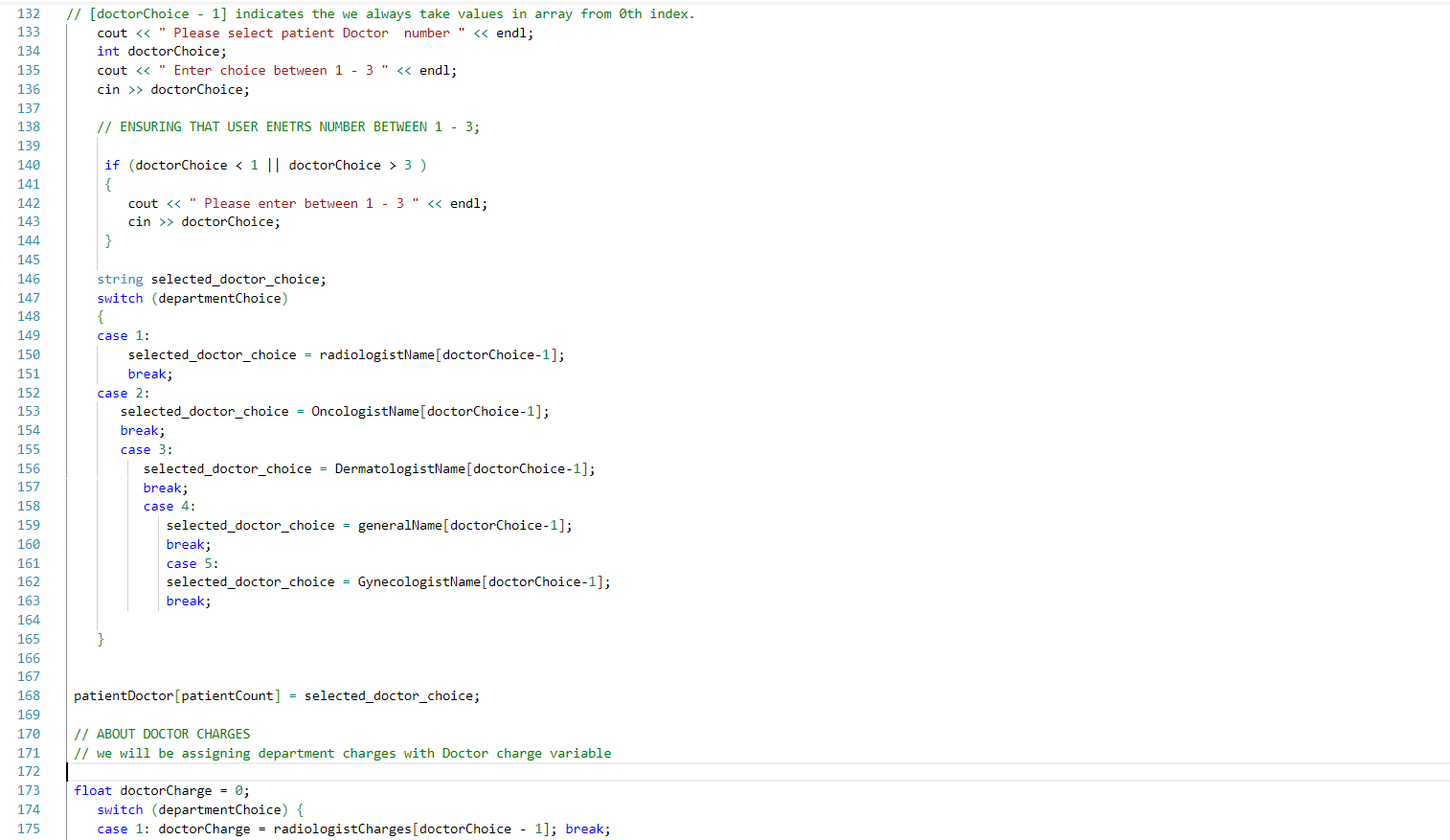
1. **Functionality**: All major functionalities, such as adding patients, viewing records, and calculating bills, work correctly. The bill includes charges for doctor consultations, additional services, and taxes.
2. **Usability**: The system provides clear prompts and choices for user input. While the system is text-based, the flow is simple to follow.
3. **Reliability**: The system is reliable in storing patient data and generating bills. The data is written to a file, ensuring persistence across multiple runs of the program.
4. **Performance**: The program performs efficiently with a maximum of 100 patients, ensuring it meets the requirements of a basic hospital management system.

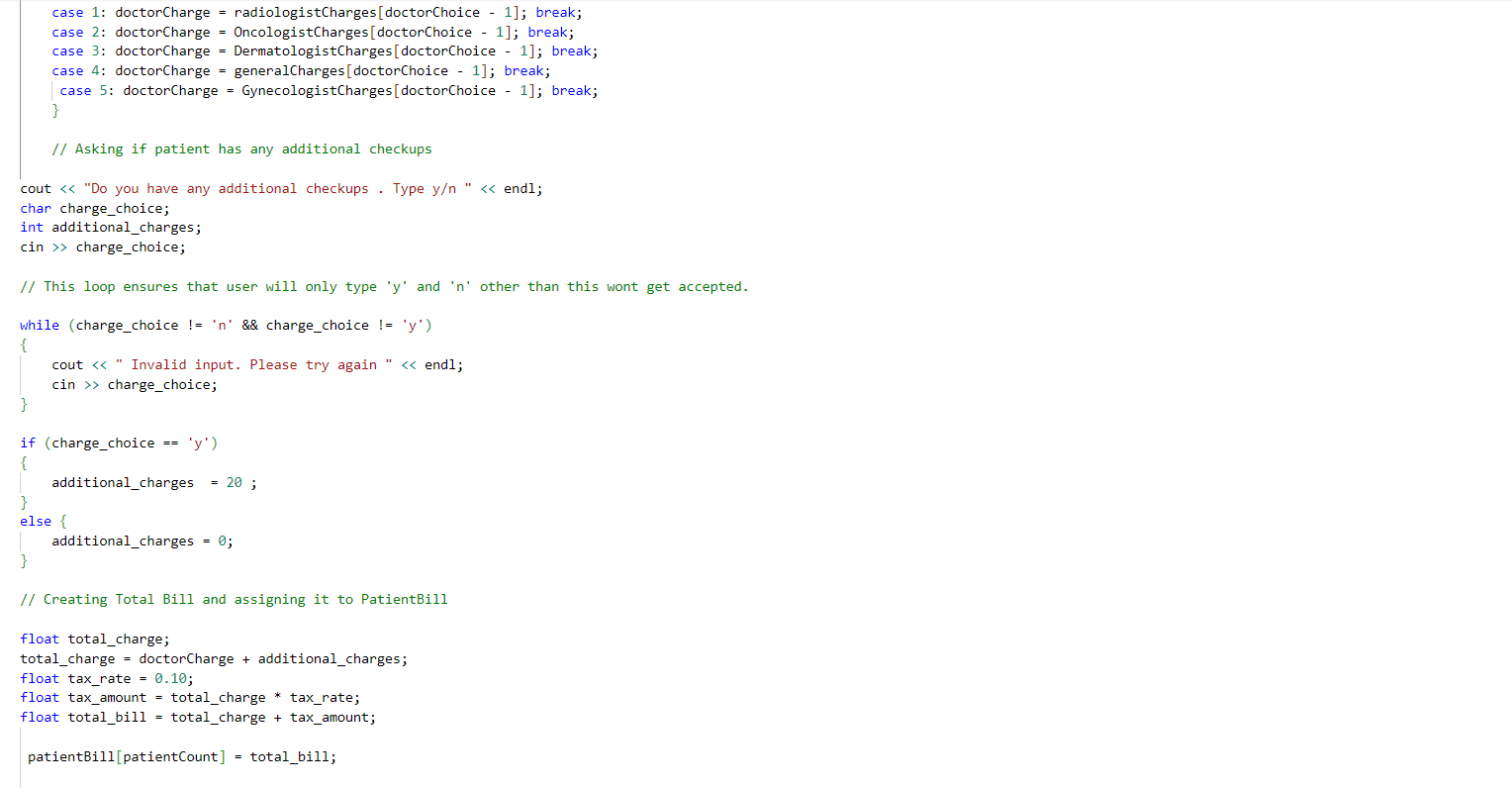
# Diagrams:

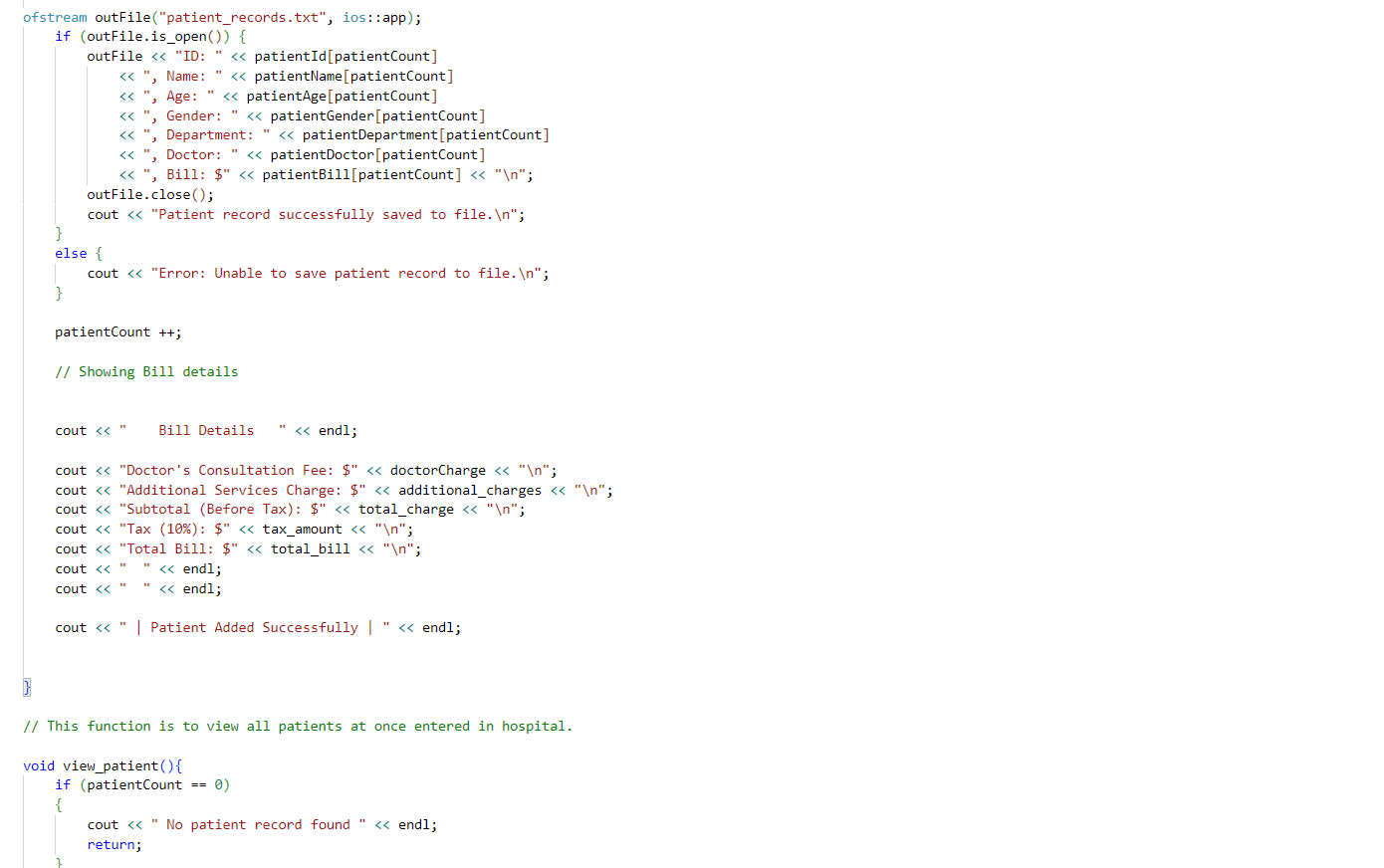
















# Code snippets:

1. **Main Menu**  
   The main menu is a loop that allows the user to interact with the system until they choose to exit.

void show\_menu() {

while (true) {

cout << "Welcome to the Hospital Management System" << endl;

cout << "1: Add patient" << endl;

cout << "2: View all patients" << endl;

cout << "3: Calculate bill" << endl;

cout << "4: Exit" << endl;

cout << "Enter your choice: ";

int my\_choice;

cin >> my\_choice;

switch (my\_choice) {

case 1:

add\_patient();

break;

case 2:

view\_patient();

break;

case 3:

calculate\_Bill();

break;

case 4:

cout << "Goodbye! Take care!" << endl;

return;

default:

cout << "Invalid choice, try again" << endl;

break;

}

}

}

# Conclusion reflecting the Project Outcomes:

This project successfully implemented a basic Hospital Management System with functionalities for patient data management, doctor selection, bill calculation, and file handling. The project achieved its goal of providing an easy-to-use interface for hospital staff to input patient details and calculate bills.

The outcomes include:

1. **Working Patient Management System**: The system allows for the addition of patients and their associated medical data.
2. **Accurate Billing**: The billing system calculates fees based on the doctor’s consultation charges and includes tax and additional services.
3. **Data Persistence**: Patient records are saved to a file, ensuring that data persists across program executions.

In the future, additional features such as more departments, advanced search options, and a graphical user interface (GUI) can be implemented to improve the system's usability and scalability.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(EnD)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_