BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY

Committed to Academic Excellence

Course Code	:	CSE100
Course Title	:	Software Development Project
Course Teacher	:	Ashfiya Jannat Keya Lecturer, Dept. of CSE

Project Report

Group Name	:	CtrlShiftHack
Project Name	:	Hotel Management System

Team Members:

Abdullah Bin Shawkat . (22235103204)
Jayanta Das .(22235103221)
Ayesha Siddika Mohona .(22235103206)
Mohammad Mahmudul Hasan Rodra .(22235103211)
Md Nazmul Hasan Siam . (22235103223)

Abstract

A **Hotel Management System** (HMS) is a software application that helps hotels manage their operations. It can help improve efficiency, reduce costs, and improve guest satisfaction. **HMSs** are becoming increasingly sophisticated and are likely to play an even more important role in the future of the hotel industry.

Acknowledgements

We would like to express our deepest gratitude to Allah and Ashfiya Jannat Keya, our project supervisor and esteemed Lecturer in the Department of Computer Science and Engineering. Her unwavering support, insightful guidance, and continuous encouragement have been invaluable throughout the development of the Hotel Management System project. Under her mentorship, we have not only honed our technical skills but also gained a profound understanding of software development principles.

Special thanks go to our classmates and friends for their collaborative spirit, constructive discussions, and shared enthusiasm, which played a crucial role in shaping the project and making it a success.

Thank you all for your contributions and support.

Declaration

We hereby declare that the Project on **Hotel Management System** in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering of Bangladesh University of Business and Technology (BUBT) is our own work and that it contains no material which has been accepted for the award to the candidate(s) of any other degree or diploma. To the best of our knowledge, it contains no materials previously published or written by any other person except where due reference is made in the project.

Contents

1. Introduction

1.1 Introduction

The hospitality industry is witnessing rapid technological advancements aimed at
enhancing operational efficiency and guest satisfaction. In line with this trend, the
Hotel Managernent System Software is proposed as a comprehensive and user-friendly application designed to automate and streamline the daily or Erations of a
hotel. This software aims to mitigate the challenges associated with manual paperwork and inefficiencies in managing room reservations, guest information, check-ins, check-outs, room services, billing, and reporting. By leveraging the power of C++ and adhering to Object-Oriented Programming (OOP) principles, the
system ensures a macular, reusable, and maintainable codebase.

1.2 Objective

The primary objectives of the **Hotel Management System** project are as follows:

1.**Operational Efficiency:**

- Simplify hotel management
- manual paperwork for increased efficiency.

2. **User-Friendly interface:**
   - Provide an intuitive interface for hotel staff.
   - Facilitate easy management of room bookings and guest information.
3. **Automation of Processes:**

   Automate room availability checks.

   Streamline reservation confirmations and guest check-ins/check-outs.

Reporting and Analytics:

Generate detailed reports for decision-making.

Optimize hotel operations through analytics.

Data Security:

Implement secure data handling.

Ensure the protection of sensitive guest information and maintain data integrity.

1.3 Project Scope

The project encompasses the following key features:

The scope of the Hotel Management System Software project encompasses the development of a robust and user-friendly application to automate and streamline various hotel operations. The project will focus on the following key areas:

Room Management:

Add, modify, and delete rooms with detailed specifications.

Display real-time availability status.

Implement room allocation during reservations.

Guest Management:

Capture and store guest information comprehensively.

Support smooth guest check-in and check-out processes.

Enable staff to modify guest details and handle specific requests.

Reservation Management:

Create, modify, and cancel room reservations.

Implement automatic room availability checks during reservations.

Generate reservation confirmations and reminders.

Billing:

Automatically calculate and generate bills based on reservations and additional services.

Provide a transparent and accurate billing system.

Reports and Analytics:

Generate various reports, including occupancy rates, revenue summaries, and guest feedback analysis.

Present data using graphical representations for easy interpretation.

Allow filtering and customization of reports based on user preferences.

Implementation Details:

Develop the software using C++ and adhere to Object-Oriented Programming (OOP) principles.

Implement key classes such as Room, Guest, Reservation, Billing, and Report Generator to ensure a modular and organized codebase.

User Interface:

Design a user-friendly command-line interface (CLI) for easy interaction with the system.

Provide clear prompts and menus to guide users through various tasks and operations.

Testing and Quality Assurance:

Conduct rigorous testing, including unit testing and integration testing, to ensure the correctness and reliability of functionalities.

Evaluate the software using real-world hotel scenarios and test cases to handle various use cases and potential errors effectively.

Tools:

Utilize a C++ compiler and Code::Blocks for successful project completion.

1.4 Our Contributions

In the development of the **Hotel Management System**, our team has played a pivotal role in the following aspects:
Identify and fix bugs to maintain high code quality.

UI/UX Design:

Design a user-friendly interface for hotel staff and guests.

Documentation:

Create user guides and developer documentation.

Database Management:

Contribute to the design and management of the database.

Security Implementation:

Implement security measures for data protection.

Performance Optimization:

Optimize system performance under heavy loads.

Feedback and Improvement:

Collect and act on user feedback for iterative improvements.

Collaboration and Communication:

Collaborate effectively and communicate progress regularly.

Adherence to Coding Standards:

Ensure contributions align with coding standards and best practices.

This project report aims to provide a comprehensive overview of the **Hotel Management System**, shedding light on its development, functionality, and the significant role it plays in optimizing blood bank operations.

2. Existing Literature

2.1 Introduction

While developing the Hotel Management System Software, a review of existing literature has been conducted to identify key trends, technologies, and best practices in hotel management systems. Several studies have emphasized the importance of automation in enhancing operational efficiency in the hospitality sector. The adoption of Object-Oriented Programming (OOP) principles in system design has proven beneficial for creating modular and maintainable software solutions.

2.2 Necessity of Hotel Management System

Operational Inefficiencies:

Challenge: Manual processes lead to errors and inefficiencies.

Necessity: The project streamlines operations, reducing paperwork and enhancing efficiency.

Data Security Concerns:

Challenge: Manual data handling poses security risks**.**

Necessity: The project prioritizes secure data management to protect guest information.

Need for User-Friendly Interfaces:

Challenge: Complex or unintuitive interfaces may lead to staff inefficiency and resistance to system adoption.

Necessity: The software addresses this by providing a user-friendly command-line interface (CLI) for seamless interaction.

Scalability and Future Expansion:

Challenge: Lack of scalability hinders future expansion.

Necessity: The project's OOP principles ensure a scalable and maintainable system.

2.3 Conclusion

The review of existing literature reveals a compelling need for the proposed Hotel Management System Software in the hospitality industry. The challenges outlined, including operational inefficiencies, security concerns, and the lack of analytical insights, underscore the significance of automating hotel management processes.

3. Proposed Model

3.1 Introduction

The proposed Hotel Management System Software is a strategic response to the evolving needs of the hospitality industry. Designed to streamline daily operations, the application focuses on automating tasks related to room reservations, guest management, billing, and reporting. Developed using C++ and following Object-Oriented Programming (OOP) principles, the system ensures a modular and maintainable codebase. With a user-friendly command-line interface (CLI), the software aims to enhance operational efficiency, provide a seamless guest experience, and contribute to data security. The model's core objectives include simplifying processes, fostering automation, and delivering insightful analytics for informed decision-making in hotel management.

3.2 Hardware and Software

● Programming Language: C++

Software:

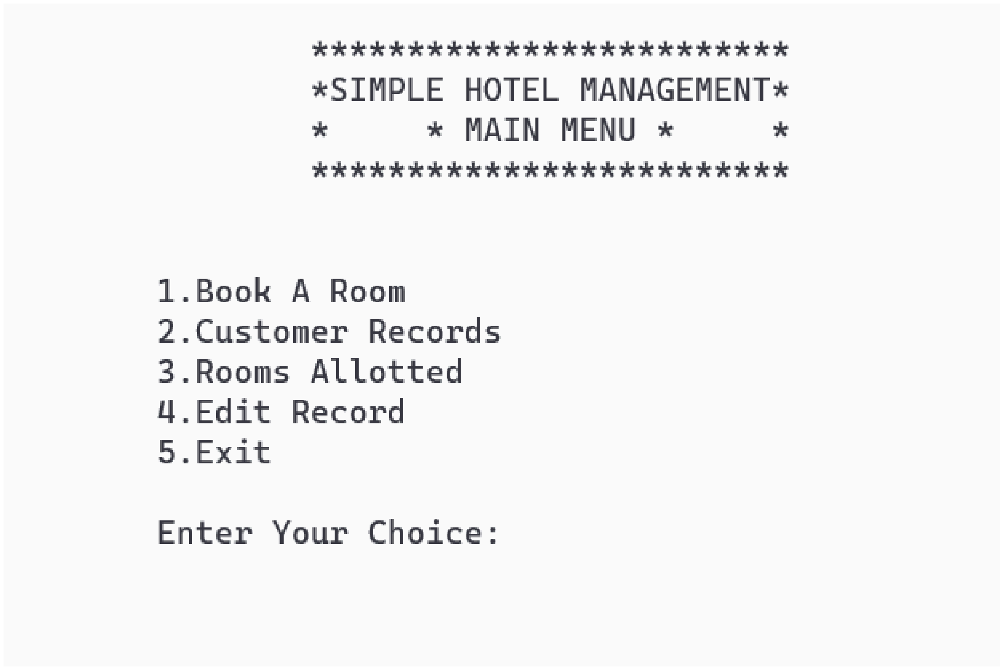
● Development Environment: Any suitable IDE (e.g., Visual Studio, Code::Blocks)

Hardware :

- 1GHz or High processor
- 512 MB RAM (minimum)
- 500 MB Hard Disk(minimum)

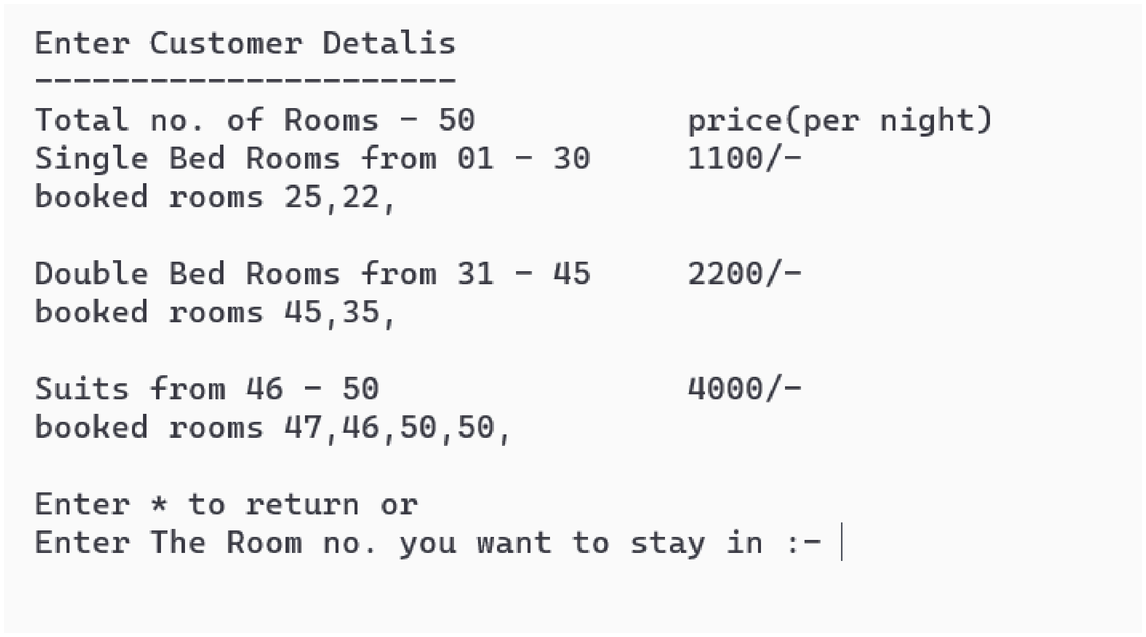
3.3 Screenshots of Encountered Interfaces

The Screenshots of the interfaces we are going to face inside the Program are shown below:



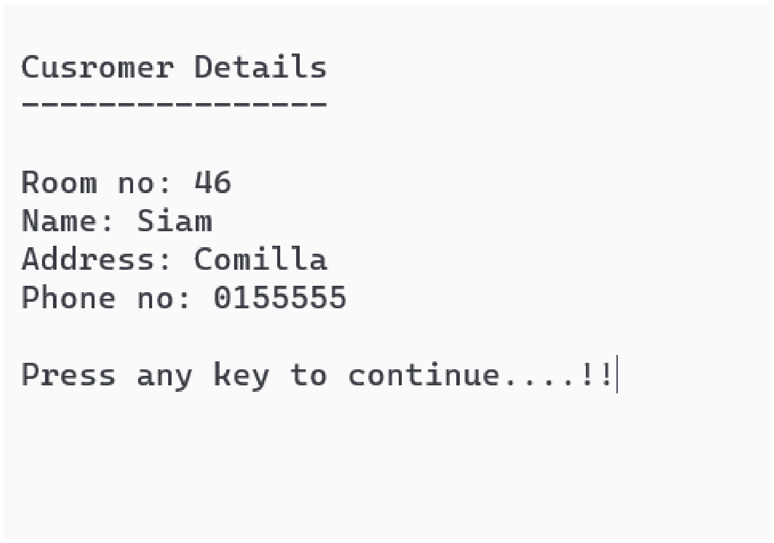
```
                *************************
                *SIMPLE HOTEL MANAGEMENT*
                *     * MAIN MENU *     *
                *************************


        1.Book A Room
        2.Customer Records
        3.Rooms Allotted
        4.Edit Record
        5.Exit

        Enter Your Choice:
```

Figure 3.1 : Main Menu Interface



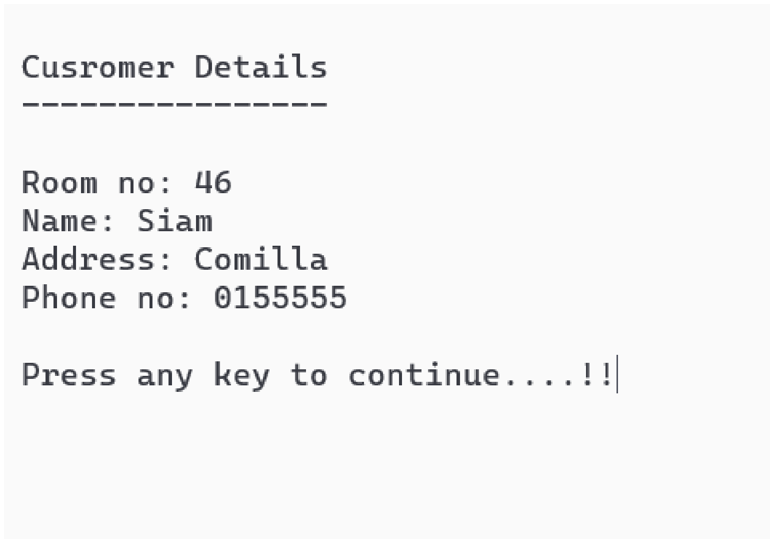
```
Enter Customer Detalis
----------------------
Total no. of Rooms - 50            price(per night)
Single Bed Rooms from 01 - 30     1100/-
booked rooms 25,22,

Double Bed Rooms from 31 - 45     2200/-
booked rooms 45,35,

Suits from 46 - 50                4000/-
booked rooms 47,46,50,50,

Enter * to return or
Enter The Room no. you want to stay in :-
```

Figure 3.2: Book Room Interface



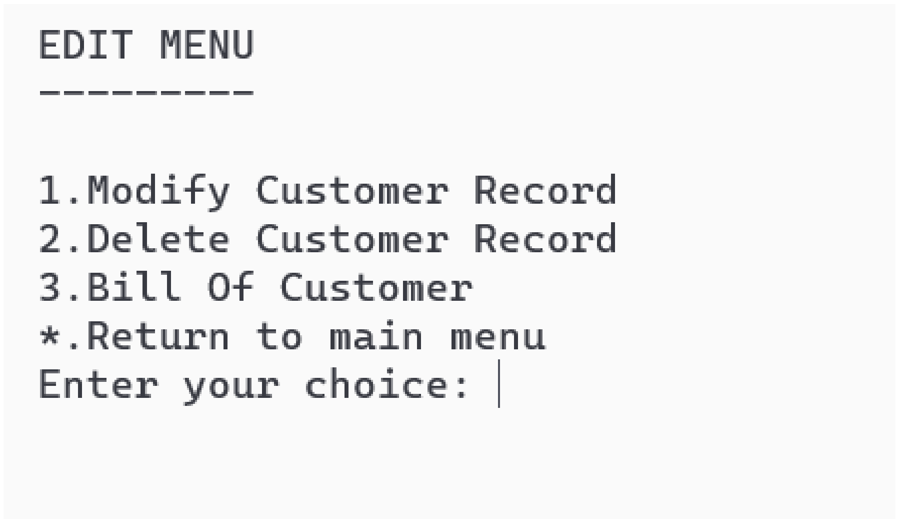
```
Cusromer Details
----------------

Room no: 46
Name: Siam
Address: Comilla
Phone no: 0155555

Press any key to continue....!!
```

Figure 3.3: Customer Detail Interface



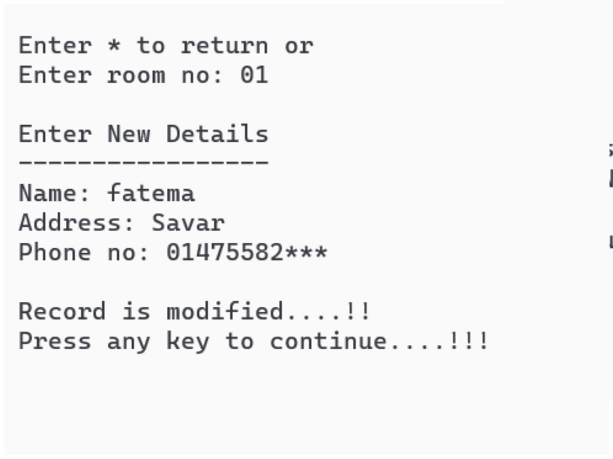
```
Cusromer Details
----------------

Room no: 46
Name: Siam
Address: Comilla
Phone no: 0155555

Press any key to continue....!!
```

Figure 3.4 : List of All Room Interface

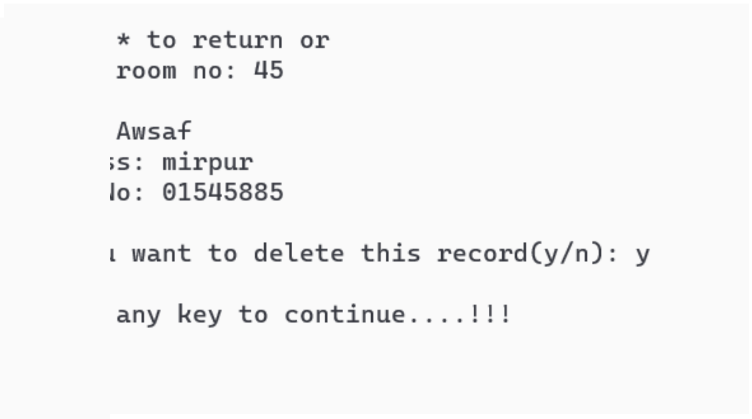


```
EDIT MENU
---------

1.Modify Customer Record
2.Delete Customer Record
3.Bill Of Customer
*.Return to main menu
Enter your choice: 
```

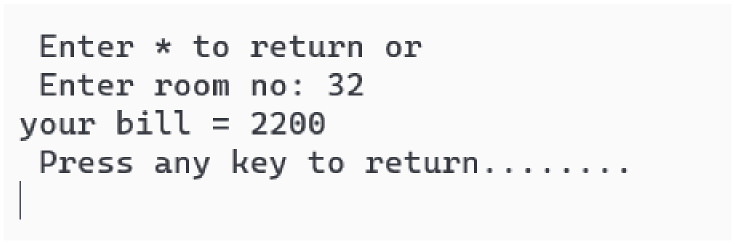
Figure 3.5 : Edit Menu Interface



```
Enter * to return or
Enter room no: 01

Enter New Details
-----------------
Name: fatema
Address: Savar
Phone no: 01475582***

Record is modified....!!
Press any key to continue....!!!
```



```
* to return or
room no: 45

Awsaf
ss: mirpur
lo: 01545885

ι want to delete this record(y/n): y

any key to continue....!!!
```



```
 Enter * to return or
 Enter room no: 32
your bill = 2200
 Press any key to return........
```

Figure 3.6 : Modify, Delete & Bill Interface

4. Implementation of Our System

4.1 Introduction

Explanation of Key functions :

Class Definition (hotel):

Defines a class representing hotel details such as room number, guest name, address, and phone number.

Contains member functions to handle various operations related to booking, displaying records, editing, and billing.

Main Menu (main_menu):

Displays a menu with options to book a room, view customer records, display allotted rooms, edit records, or exit the program.

Uses a switch-case structure to execute the selected option.

Booking a Room (add):

Takes customer details and allows booking of a room.

Checks for room availability and writes the booking details to a file ("Record.dat").

Displaying Customer Records (display):

Displays details of a specific customer based on the entered room number.

Reads customer records from the "Record.dat" file.

Displays a list of all allotted rooms along with customer details.

Reads records from the "Record.dat" file.

Editing and Deleting Records (edit, modify, delete_rec):

Allows modification and deletion of customer records based on room number.

Uses file handling to read and write records to a temporary file ("temp.dat").

Checking Room Availability (check):

Checks if a given room number is already booked.

Helps prevent double booking.

Generating Customer Bill (bill):

Calculates and displays the bill based on the type of room booked.

Reads records from the "Record.dat" file.

Main Function (main):

Instantiates the hotel class and initiates the main menu.

5. A Code Walkthrough

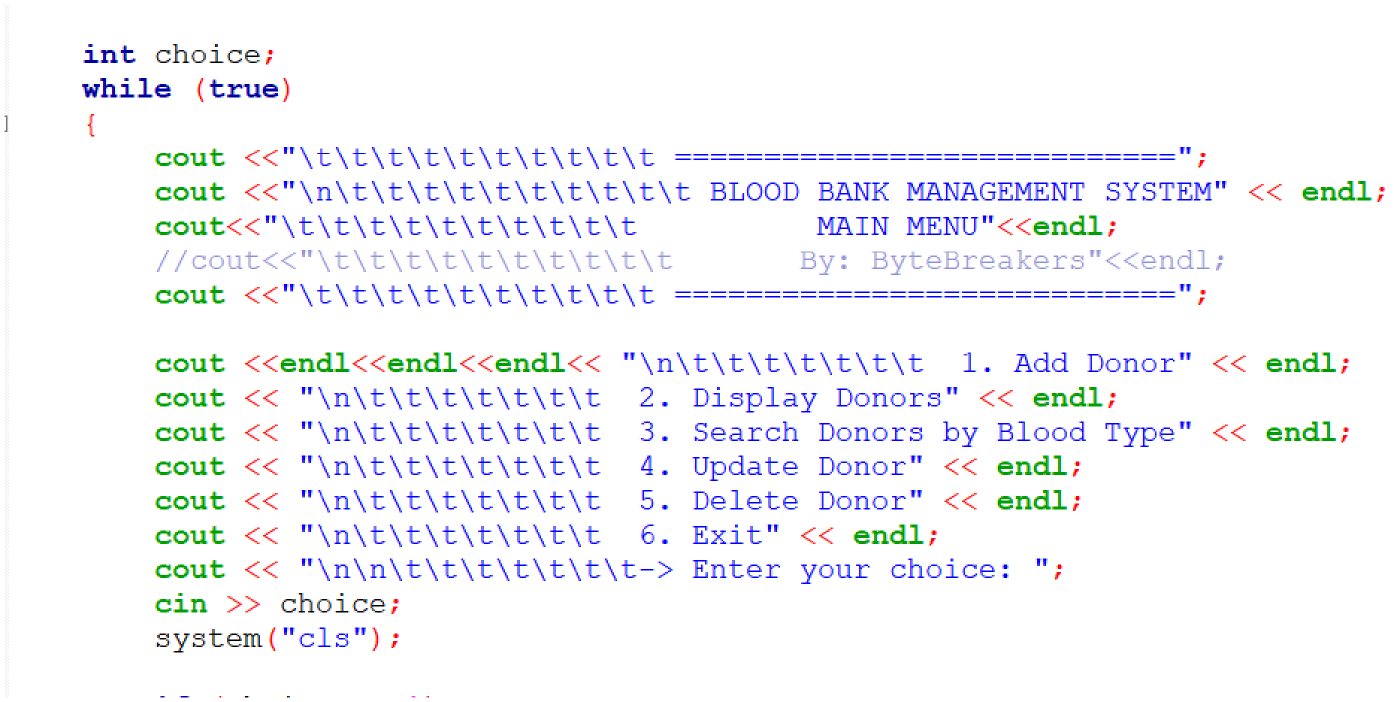
5.1 Introduction

Here we have analyzed all the results regarding to the project and also we have tried to show the source code of our program.

5.2 Result Analysis

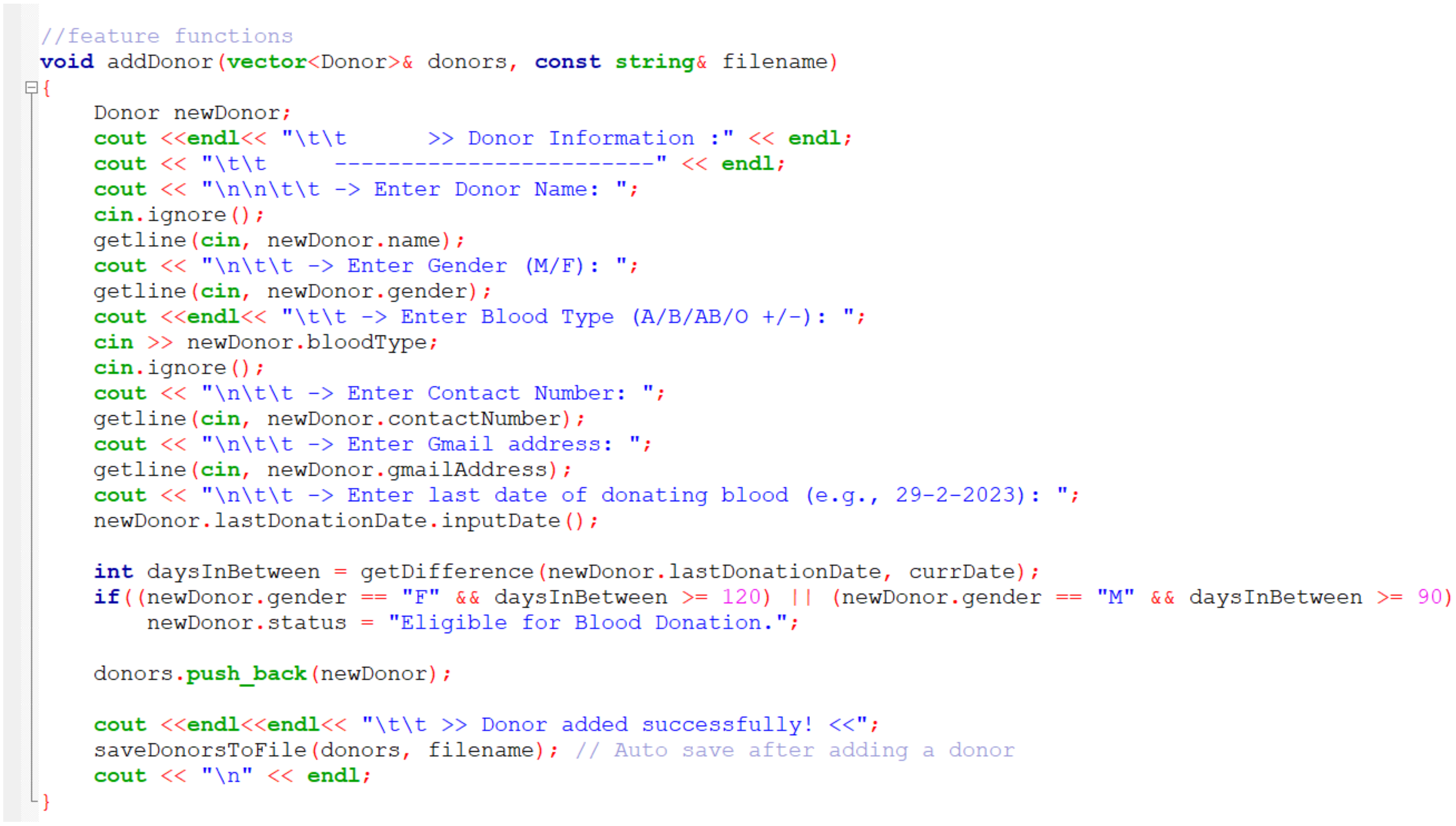
Some screenshots of our program source code are shown below in order.

Starting With the Main Menu



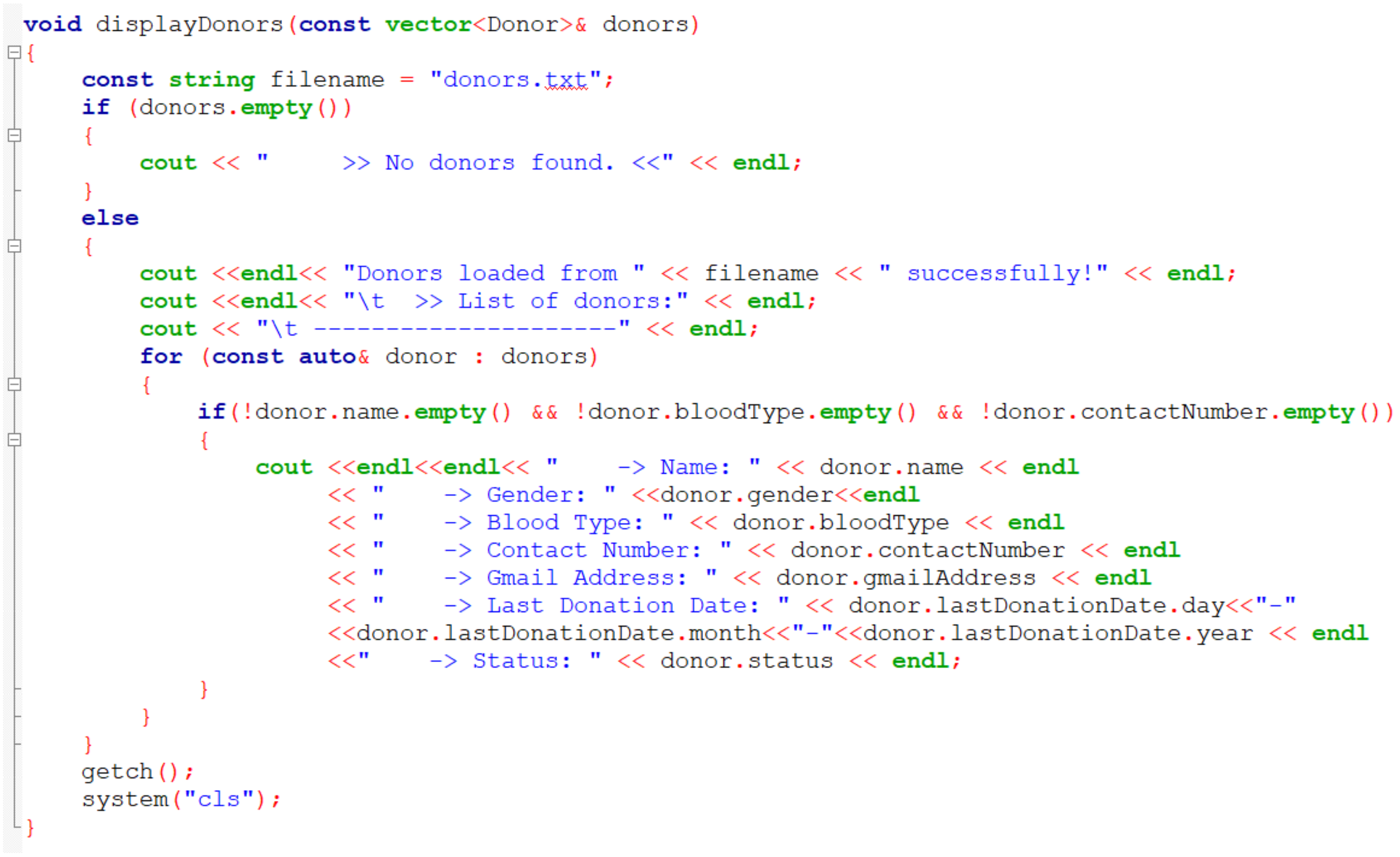
```
int choice;
while (true)
{
    cout <<"\t\t\t\t\t\t\t\t\t\t\t ============================";
    cout <<"\n\t\t\t\t\t\t\t\t\t\t\t BLOOD BANK MANAGEMENT SYSTEM" << endl;
    cout<<"\t\t\t\t\t\t\t\t\t\t\t          MAIN MENU"<<endl;
    //cout<<"\t\t\t\t\t\t\t\t\t\t\t       By: ByteBreakers"<<endl;
    cout <<"\t\t\t\t\t\t\t\t\t\t\t ============================";

    cout <<endl<<endl<<endl<< "\n\t\t\t\t\t\t\t  1. Add Donor" << endl;
    cout << "\n\t\t\t\t\t\t\t\t  2. Display Donors" << endl;
    cout << "\n\t\t\t\t\t\t\t\t  3. Search Donors by Blood Type" << endl;
    cout << "\n\t\t\t\t\t\t\t\t  4. Update Donor" << endl;
    cout << "\n\t\t\t\t\t\t\t\t  5. Delete Donor" << endl;
    cout << "\n\t\t\t\t\t\t\t\t  6. Exit" << endl;
    cout << "\n\n\t\t\t\t\t\t\t\t-> Enter your choice: ";
    cin >> choice;
    system("cls");
```

Figure 5.1 : SS of Main Menu Code



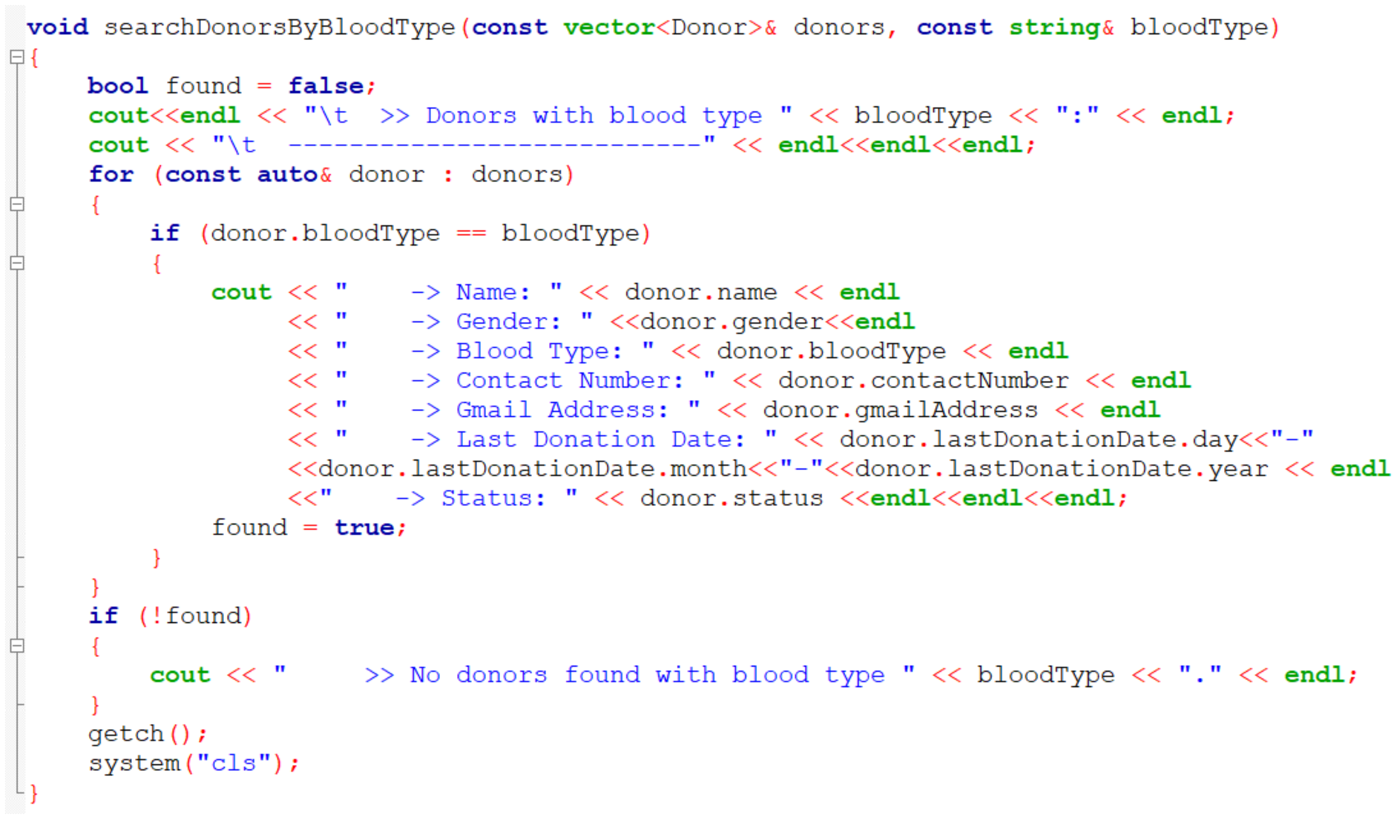
```
//feature functions
void addDonor(vector<Donor>& donors, const string& filename)
{
    Donor newDonor;
    cout <<endl<< "\t\t      >> Donor Information :" << endl;
    cout << "\t\t     ------------------------" << endl;
    cout << "\n\n\t\t -> Enter Donor Name: ";
    cin.ignore();
    getline(cin, newDonor.name);
    cout << "\n\t\t -> Enter Gender (M/F): ";
    getline(cin, newDonor.gender);
    cout <<endl<< "\t\t -> Enter Blood Type (A/B/AB/O +/-): ";
    cin >> newDonor.bloodType;
    cin.ignore();
    cout << "\n\t\t -> Enter Contact Number: ";
    getline(cin, newDonor.contactNumber);
    cout << "\n\t\t -> Enter Gmail address: ";
    getline(cin, newDonor.gmailAddress);
    cout << "\n\t\t -> Enter last date of donating blood (e.g., 29-2-2023): ";
    newDonor.lastDonationDate.inputDate();

    int daysInBetween = getDifference(newDonor.lastDonationDate, currDate);
    if((newDonor.gender == "F" && daysInBetween >= 120) || (newDonor.gender == "M" && daysInBetween >= 90)
        newDonor.status = "Eligible for Blood Donation.";

    donors.push_back(newDonor);

    cout <<endl<<endl<< "\t\t >> Donor added successfully! <<";
    saveDonorsToFile(donors, filename); // Auto save after adding a donor
    cout << "\n" << endl;
}
```

Figure 5.2 : SS of Add Donor Feature Code



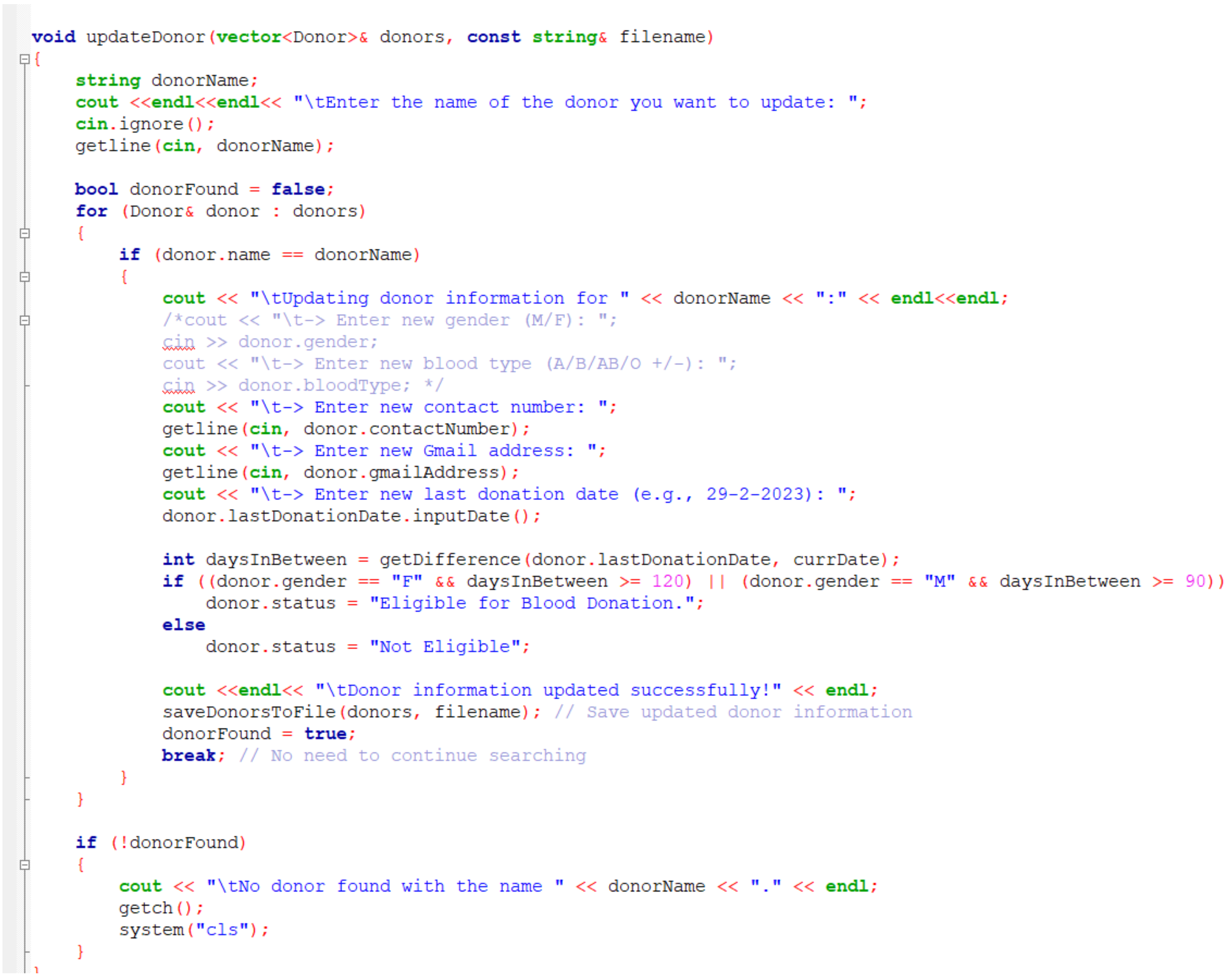
```
void displayDonors(const vector<Donor>& donors)
{
    const string filename = "donors.txt";
    if (donors.empty())
    {
        cout << "     >> No donors found. <<" << endl;
    }
    else
    {
        cout <<endl<< "Donors loaded from " << filename << " successfully!" << endl;
        cout <<endl<< "\t  >> List of donors:" << endl;
        cout << "\t ---------------------" << endl;
        for (const auto& donor : donors)
        {
            if(!donor.name.empty() && !donor.bloodType.empty() && !donor.contactNumber.empty())
            {
                cout <<endl<<endl<< "    -> Name: " << donor.name << endl
                     << "    -> Gender: " <<donor.gender<<endl
                     << "    -> Blood Type: " << donor.bloodType << endl
                     << "    -> Contact Number: " << donor.contactNumber << endl
                     << "    -> Gmail Address: " << donor.gmailAddress << endl
                     << "    -> Last Donation Date: " << donor.lastDonationDate.day<<"-"
                     <<donor.lastDonationDate.month<<"-"<<donor.lastDonationDate.year << endl
                     <<"    -> Status: " << donor.status << endl;
            }
        }
    }
    getch();
    system("cls");
}
```

Figure 5.3 : SS of Display Donors Feature Code



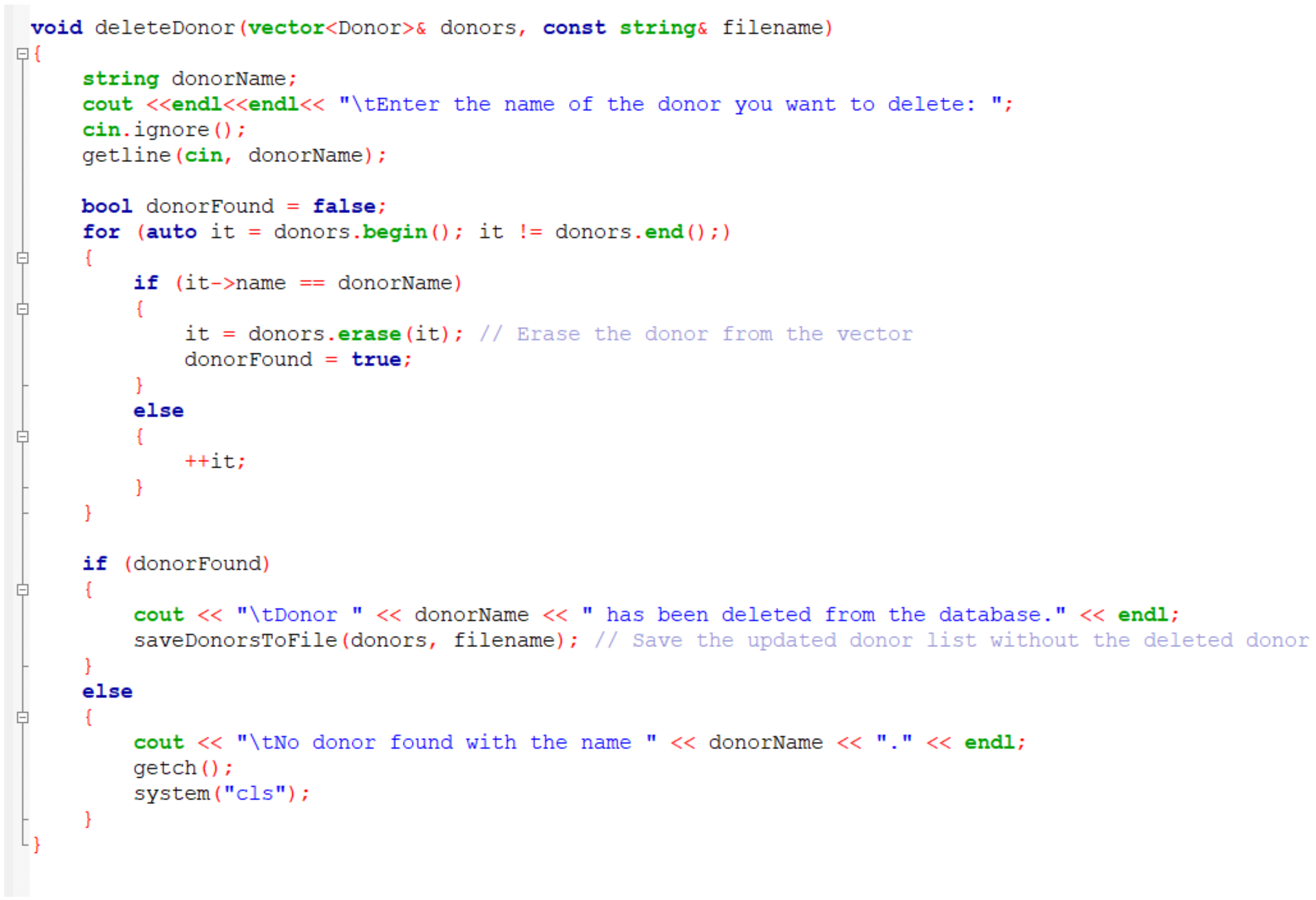
```cpp
void searchDonorsByBloodType(const vector<Donor>& donors, const string& bloodType)
{
    bool found = false;
    cout<<endl << "\t  >> Donors with blood type " << bloodType << ":" << endl;
    cout << "\t  ---------------------------" << endl<<endl<<endl;
    for (const auto& donor : donors)
    {
        if (donor.bloodType == bloodType)
        {
            cout << "    -> Name: " << donor.name << endl
                 << "    -> Gender: " <<donor.gender<<endl
                 << "    -> Blood Type: " << donor.bloodType << endl
                 << "    -> Contact Number: " << donor.contactNumber << endl
                 << "    -> Gmail Address: " << donor.gmailAddress << endl
                 << "    -> Last Donation Date: " << donor.lastDonationDate.day<<"-"
                 <<donor.lastDonationDate.month<<"-"<<donor.lastDonationDate.year << endl
                 <<"    -> Status: " << donor.status <<endl<<endl<<endl;
            found = true;
        }
    }
    if (!found)
    {
        cout << "      >> No donors found with blood type " << bloodType << "." << endl;
    }
    getch();
    system("cls");
}
```

Figure 5.4 : SS of Search by Blood Type Feature Code



```
void updateDonor(vector<Donor>& donors, const string& filename)
{
    string donorName;
    cout <<endl<<endl<< "\tEnter the name of the donor you want to update: ";
    cin.ignore();
    getline(cin, donorName);

    bool donorFound = false;
    for (Donor& donor : donors)
    {
        if (donor.name == donorName)
        {
            cout << "\tUpdating donor information for " << donorName << ":" << endl<<endl;
            /*cout << "\t-> Enter new gender (M/F): ";
            cin >> donor.gender;
            cout << "\t-> Enter new blood type (A/B/AB/O +/-): ";
            cin >> donor.bloodType; */
            cout << "\t-> Enter new contact number: ";
            getline(cin, donor.contactNumber);
            cout << "\t-> Enter new Gmail address: ";
            getline(cin, donor.gmailAddress);
            cout << "\t-> Enter new last donation date (e.g., 29-2-2023): ";
            donor.lastDonationDate.inputDate();

            int daysInBetween = getDifference(donor.lastDonationDate, currDate);
            if ((donor.gender == "F" && daysInBetween >= 120) || (donor.gender == "M" && daysInBetween >= 90))
                donor.status = "Eligible for Blood Donation.";
            else
                donor.status = "Not Eligible";

            cout <<endl<< "\tDonor information updated successfully!" << endl;
            saveDonorsToFile(donors, filename); // Save updated donor information
            donorFound = true;
            break; // No need to continue searching
        }
    }

    if (!donorFound)
    {
        cout << "\tNo donor found with the name " << donorName << "." << endl;
        getch();
        system("cls");
    }
}
```

Figure 5.5 : SS of Update Donor Feature Code



```cpp
void deleteDonor(vector<Donor>& donors, const string& filename)
{
    string donorName;
    cout <<endl<<endl<< "\tEnter the name of the donor you want to delete: ";
    cin.ignore();
    getline(cin, donorName);

    bool donorFound = false;
    for (auto it = donors.begin(); it != donors.end();)
    {
        if (it->name == donorName)
        {
            it = donors.erase(it); // Erase the donor from the vector
            donorFound = true;
        }
        else
        {
            ++it;
        }
    }

    if (donorFound)
    {
        cout << "\tDonor " << donorName << " has been deleted from the database." << endl;
        saveDonorsToFile(donors, filename); // Save the updated donor list without the deleted donor
    }
    else
    {
        cout << "\tNo donor found with the name " << donorName << "." << endl;
        getch();
        system("cls");
    }
}
```

Figure 5.6 : Delete Donor Feature Code

6. Conclusion

The Hotel Management System implemented in C++ demonstrates a foundational structure for automating and managing key operations in a hotel. The code successfully captures essential functionalities such as room booking, customer record management, billing, and reporting through a command-line interface (CLI).

The project's strengths lie in its simplicity and effectiveness in handling routine hotel management tasks. It employs file handling for persistent data storage, allowing the system to maintain customer records across sessions. The inclusion of features like room availability checks, billing calculations, and record editing adds practical value to the system.

However, there are areas for improvement and further development. The code lacks robust error handling, which could enhance the system's resilience to incorrect inputs. Additionally, the CLI might be limiting for users who prefer more intuitive graphical interfaces. The project could benefit from incorporating modern programming practices, such as exception handling and enhanced user interactions.

In conclusion, while the current implementation provides a functional Hotel Management System, there is room for refinement and expansion to meet the evolving needs of the hospitality industry. Future iterations could focus on user experience enhancements, security measures, and the incorporation of advanced features to make the system more comprehensive and adaptable to diverse hotel management scenarios.