<u>SIDDAGANGA INSTITUTE OF TECHNOLOGY</u>

TUMKUR,B.H ROAD – 572103,INDIA

<u>Hostel Management System</u>

AMAN KUSHWAHA

<u>1SI20CS009</u>

Computer Science and Engineering

Introduction:

This C++ program is a Hostel Management System designed to facilitate the efficient management of hostel facilities. It allows hostel administrators to perform various tasks such as adding residents, adding rooms, deallocating rooms, and displaying the status of each room. The program utilizes data structures like vectors to store information about residents and rooms.

The main menu provides options for performing different tasks, including adding a new resident, adding a new room, deallocating a room, and displaying the current status of each room.

Additionally, the program incorporates file I/O functionality to load data from files upon startup and save data to files before exiting. This ensures that information about residents and rooms persists across program sessions.

Problem Definition:

Hostel facilities often face challenges in efficiently managing resident accommodations, room allocations, and administrative tasks. Manual methods of managing hostel operations are prone to errors, time-consuming, and lack scalability. There is a need for a digital solution that streamlines hostel management processes, enhances administrative efficiency, and improves the overall resident experience.

Key Issues:

Manual Management: Current hostel management practices rely heavily on manual record-keeping and administrative tasks, leading to inefficiencies and inaccuracies in data management.

Room Allocation: Allocating rooms to residents based on preferences, availability, and occupancy can be a complex and time-consuming process for hostel administrators.

Resource Optimization: Hostel facilities need to optimize room occupancy and resource utilization to maximize revenue generation and operational efficiency.

Resident Experience: Hostel residents expect a seamless and user-friendly experience for tasks such as room allocation, registration, and communication with hostel administrators.

Data Management: Managing resident and room data efficiently, maintaining records accurately, and generating insights for decision-making are challenging tasks without a centralized system.

Objective:

The objective of the Hostel Management System project is to develop a digital solution that automates and streamlines hostel management processes. The system aims to address the key issues mentioned above by providing functionalities for efficient room allocation, resident registration, resource optimization, and data management. The goal is to enhance administrative efficiency, improve resident experience, and optimize hostel operations through digitization and automation.

Working:

1. Initialization:
   - The program starts by initializing data structures to store information about residents and rooms. It also defines file names for storing data persistently.

2. Data Loading:
   - Upon startup, the program attempts to load resident and room data from files. If successful, existing data is populated into the program's data structures.

3. Main Menu:
   - The program presents a main menu with options for different tasks: adding a new resident, adding a new room, deallocating a room, displaying room status, and exiting the program.

4. Task Execution:
   - Depending on the user's choice, the program executes the corresponding task:
     - Add Resident: Prompts the user to enter details about a new resident (name, USN, age). The resident is then added to the system.
     - Add Room: Prompts the user to enter the number of a new room. The room is then added to the system.
     - Deallocate Room: Allows the user to deallocate a room by entering its number. If the room is occupied, it is marked as vacant.
     - Display Room Status: Displays the status (filled or vacant) of each room in the hostel.
     - Exit: Exits the program.

5. Data Persistence:
   - Before exiting, the program saves the current resident and room data to files, ensuring that any changes made during the session are persisted for future use.

6. Error Handling:
   - Throughout the execution, the program performs error handling to ensure smooth operation. It notifies users of any errors encountered during file operations or invalid user inputs.

7. Efficiency and User Experience:

- The program is designed to streamline hostel management tasks, providing administrators with a user-friendly interface to add residents, allocate rooms, deallocate rooms, and view room status. By leveraging data structures and file I/O, it offers efficient data management and persistence.

Code:

```cpp
#include<bits/stdc++.h>
#include <iostream>
#include <fstream>
#include <string>
#include <vector>

using namespace std;

// Define data structures
struct Resident {
    string name;
    string usn; // University Serial Number
    int age;
    // Add more details as needed
};

struct Room {
    int number;
    bool occupied;
    Resident* resident;

    // Initialize room data in the constructor
    Room(int number) : number(number), occupied(false), resident(nullptr) {}
};

// Function prototypes
void addResident(vector<Resident>& residents, vector<Room>& rooms);
void addRoom(vector<Room>& rooms);
void deallocateRoom(vector<Room>& rooms);
void displayRoomStatus(const vector<Room>& rooms);

// Functions to read and write data to files
bool loadResidentsFromFile(vector<Resident>& residents, const string& filename);
bool saveResidentsToFile(const vector<Resident>& residents, const string&
filename);
bool loadRoomsFromFile(vector<Room>& rooms, const string& filename);
bool saveRoomsToFile(const vector<Room>& rooms, const string& filename);

int main() {
    // File names for residents and rooms data
    const string residentsFilename = "residents.data";
    const string roomsFilename = "rooms.data";

    // Load data from files
    vector<Resident> residents;
    vector<Room> rooms;
    if (!loadResidentsFromFile(residents, residentsFilename) ||
!loadRoomsFromFile(rooms, roomsFilename)) {
        cout << "Error loading data from files." << endl;
    }

    int choice;
    do {
        // Main menu
        cout << "Hostel Management System" << endl;
        cout << "1. Add Resident" << endl;
        cout << "2. Add Room" << endl;
        cout << "3. Deallocate Room" << endl;
        cout << "4. Display Room Status" << endl;
        cout << "0. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                addResident(residents, rooms);
                break;
            case 2:
                addRoom(rooms);
                break;
            case 3:
                deallocateRoom(rooms);
                break;
            case 4:
                displayRoomStatus(rooms);
                break;
            case 0:
                cout << "Exiting..." << endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    } while (choice != 0);

    // Save data to files before exiting
    if (!saveResidentsToFile(residents, residentsFilename) ||
!saveRoomsToFile(rooms, roomsFilename)) {
        cout << "Error saving data to files." << endl;
    }

    return 0;
}
```

```
// Function to add a new resident
void addResident(vector<Resident>& residents, vector<Room>& rooms) {
    Resident newResident;
    cout << "Enter resident name: ";
    cin >> newResident.name;
    cout << "Enter resident USN: ";
    cin >> newResident.usn;
    cout << "Enter resident age: ";
    cin >> newResident.age;
    // Add more details as needed
    residents.push_back(newResident);

    // Allocate rooms based on some logic
    // For example, you can implement alphabetical allocation here
}

// Function to add a new room
void addRoom(vector<Room>& rooms) {
    int roomNumber;
    cout << "Enter room number: ";
    cin >> roomNumber;
    rooms.push_back(Room(roomNumber));
}

// Function to deallocate a room
void deallocateRoom(vector<Room>& rooms) {
    int roomNumber;
    cout << "Enter the room number to deallocate: ";
    cin >> roomNumber;

    // Find the room by its number
    for (Room& room : rooms) {
        if (room.number == roomNumber) {
            if (room.occupied) {
                // Free the room
                room.occupied = false;
                room.resident = nullptr;
                cout << "Room " << roomNumber << " deallocated successfully." <<
endl;
                return;
            } else {
                cout << "Room " << roomNumber << " is already vacant." << endl;
                return;
            }
        }
    }
```

```cpp
    cout << "Room " << roomNumber << " not found." << endl;
}

// Function to display the status of each room (filled or vacant)
void displayRoomStatus(const vector<Room>& rooms) {
    cout << "Room Status:" << endl;
    for (const Room& room : rooms) {
        cout << "Room " << room.number << ": ";
        if (room.occupied) {
            cout << "Filled" << endl;
        } else {
            cout << "Vacant" << endl;
        }
    }
}

// Function to read residents from a file
bool loadResidentsFromFile(vector<Resident>& residents, const string& filename) {
    ifstream file(filename);
    if (!file.is_open()) {
        cerr << "Error opening " << filename << " for reading." << endl;
        return false;
    }

    string name, usn;
    int age;
    while (file >> name >> usn >> age) {
        residents.push_back({name, usn, age});
    }

    file.close();
    return true;
}

// Function to save residents to a file
bool saveResidentsToFile(const vector<Resident>& residents, const string&
filename) {
    ofstream file(filename);
    if (!file.is_open()) {
        cerr << "Error opening " << filename << " for writing." << endl;
        return false;
    }

    for (const Resident& resident : residents) {
        file << resident.name << " " << resident.usn << " " << resident.age <<
endl;
    }
    file.close();
    return true;
}

// Function to read rooms from a file (similar to loadResidentsFromFile)
bool loadRoomsFromFile(vector<Room>& rooms, const string& filename) {
    ifstream file(filename);
    if (!file.is_open()) {
        cerr << "Error opening " << filename << " for reading." << endl;
        return false;
    }

    int number;
    bool occupied;
    while (file >> number >> occupied) {
        rooms.push_back({number, occupied, nullptr});
    }

    file.close();
    return true;
}

// Function to save rooms to a file (similar to saveResidentsToFile)
bool saveRoomsToFile(const vector<Room>& rooms, const string& filename) {
    ofstream file(filename);
    if (!file.is_open()) {
        cerr << "Error opening " << filename << " for writing." << endl;
        return false;
    }

    for (const Room& room : rooms) {
        file << room.number << " " << room.occupied << endl;
    }

    file.close();
    return true;
}
```

Conclusion:

The Hostel Management System project aims to address the challenges faced by hostel facilities in efficiently managing resident accommodations and administrative tasks. By providing a digital solution that automates processes such as room allocation, resident registration, and data management, the system seeks to enhance administrative efficiency, improve resource utilization, and optimize the overall resident experience.

Through the development of this system, hostel administrators can streamline operations, minimize manual errors, and gain insights into occupancy trends and resource utilization. Residents benefit from a user-friendly interface for room allocation, registration, and communication with administrators, leading to improved satisfaction and experience.