Admin

```cpp
#include <bits/stdc++.h>
#include <string>
#include "header.h"

using namespace std;

void executeOption(int choice);


int main ()
{
    while (true)
    {
        int choice;
        cout << "1. Hostel Management" << endl << "2. Non-Hostel" << endl;
        cout << "3. Cycle Rental System" << endl << "0. Exit" << endl;
        cout << "Choose action: ";
        cin >> choice;
        if (choice == 0)
        {
            cout << "Exiting student management system" << endl;
            return 0;
        }
        executeOption(choice);
    }
    return 0;
}

void executeOption(int choice)
{
    switch (choice)
    {
    case 1:
        hostel();
        break;
    case 2:
        bus();
        break;
    case 3:
        cycle();
        break;
    default:
        cout << "Invalid choice, try again" << endl;
    }
}
//filename: admin.cpp
//to compile the files--->g++ admin.cpp DS_Cycle.cpp DS_Hostel.cpp
DS_Transport.cpp header.h -o admin
```

# Admin.cpp Documentation

## Overview

The `admin.cpp` file is the focal point of a larger system that manages a hostel, a bus system, and a cycle rental system. It provides a CLI interface for users to interact with these systems.

## Main Function

The `main()` function initializes the student management system and then enters a loop where it displays the main menu and waits for the user to enter their choice. The loop continues until the user enters `1-3` or anything else other than the given options, at which point the program exits.

## Display Main Menu

The `main()` function prints the main menu options to the console. The options are:

- `1. Hostel Management`

- `2. Non-Hostel`

- `3. Cycle Rental System`

- `0. Exit`

## Execute Option

The `executeOption(int choice)` function takes the user's choice as an argument and executes the corresponding function. If the user enters an invalid choice, it ends the program.

## Dependencies

This file includes the following dependencies:

- `DS_Hostel.cpp`: Contains functions related to hostel management.

- `DS_Transport.cpp`: Contains functions related to the bus system.

- `DS_Cycle.cpp`: Contains functions related to the cycle rental system.

## Note

This is the documentation of the Admin part of Student Management System

Cycle Management System Documentation

```cpp
// DS_Cycle.cpp
#include <iostream>
#include <string>
#include <utility>
#include "header.h"

const int MAX_CYCLES = 50;
int TOP = MAX_CYCLES - 1;

using namespace std;
void pushStudent(string id, const string& out);
void pushStudentIn(const string& id, const string& in);


struct Student
{
    string id;
    string out;
    string in;
    int secOut{};
    int secIn{};
    int total{};
    double cost{}; // Change the data type to double for accurate cost
calculation
    Student *next{};
};

Student *top = nullptr;

bool isEmpty()
{
    return TOP == -1;
}

void popCycle()
```

```
{
    TOP--;
}

void pushCycle()
{
    TOP++;
}

int pushTime(const string &t)
{
    int time = stoi(t);
    int hours = time / 100;
    int minutes = time - (hours * 100);
    return (hours * 3600) + (minutes * 60);
}

void rentCycle()
{
    string id, out;
    cout << "Student: Can I rent a cycle?" << endl;

    if (isEmpty())
    {
        cout << "No cycles available" << endl;
    }
    else
    {
        cout << "Yes, you can" << endl;
        cout << "Enter ID: ";
        cin.ignore();
        getline(cin, id);
        cout << "Out time (24-hour format, e.g., 1220 for 12:20): ";
        getline(cin, out);
        cout << "Enjoy your ride!! =)" << endl;

        pushStudent(id, out);
        popCycle();
    }
}

void returnCycle()
{
    string basicString, in, id;
    cout << "Staff: Return cycle? (yes/no)" << endl;
```

```cpp
    cout << "Student: ";
    cin >> basicString;

    if (basicString == "yes")
    {
        cout << "Enter ID: ";
        cin.ignore();
        getline(cin, id);
        cout << "In time (24-hour format, e.g., 1220 for 12:20): ";
        getline(cin, in);

        pushStudentIn(id, in);
        pushCycle();
    }
    else
    {
        cout << "Continue enjoy your ride!" << endl;
    }
}

void pushStudent(string id, const string& out)
{
    auto *newNode = new Student;
    newNode->id = std::move(id);
    newNode->out = out;
    newNode->secOut = pushTime(out);
    newNode->next = top;
    top = newNode;
}

void pushStudentIn(const string& id, const string& in)
{
    Student *temp = top;

    while (temp != nullptr)
    {
        if (temp->id == id)
        {
            temp->in = in;
            temp->secIn = pushTime(in);
            temp->total = temp->secIn - temp->secOut;
            temp->cost = (temp->total / 60.0) * 0.5; // Corrected the cost
calculation
            cout << "Total time for " << temp->id << ": " << temp->total << "
seconds" << endl;
```

```
            cout << "Cost for " << temp->id << ": tk " << temp->cost << endl;
            return;
        }
        temp = temp->next;
    }

    cout << "No entry found for student: " << id << endl;
}

void cycle()
{
    while (true)
    {
        std::cout << "Cycle Rental System Menu:" << std::endl;
        std::cout << "1. Rent a Cycle" << std::endl;
        std::cout << "2. Return a Cycle" << std::endl;
        std::cout << "3. Exit Cycle Rental System" << std::endl;

        int choice;
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice)
        {
            case 1:
                rentCycle();
                break;
            case 2:
                returnCycle();
                break;
            case 3:
                std::cout << "Exiting Cycle Rental System." << std::endl;
                return;
            default:
                std::cout << "Invalid choice. Please try again." << std::endl;
                break;
        }
    }
}
```

# Cycle Rental System: Comprehensive Documentation

This documentation provides a detailed and professional overview of the Cycle Rental System, a C++ program designed to manage cycle rental operations. The system offers a user-friendly command-line interface for easy interaction and is designed to be flexible and customizable.

## System Design

The system is designed with a focus on modularity and simplicity. It uses a singly linked list to manage the list of students who have rented cycles. Each node in the list represents a student and contains information about the student's ID, the time they rented and returned the cycle, and the total cost of the rental. The system also maintains a global variable `top` that points to the top of the student list.

## Data Structures

The system defines a single structure, `Student`. This structure is used to represent a student who rents a cycle. Each `Student` object contains the following fields:

- `id`: A string that represents the unique identifier for the student.
- `out`: A string that represents the time when the student rents a cycle.
- `in`: A string that represents the time when the student returns the cycle.
- `secOut`: An integer that represents the number of seconds when the student rents a cycle.
- `secIn`: An integer that represents the number of seconds when the student returns the cycle.
- `total`: An integer that represents the total time the cycle is rented.
- `cost`: A double that represents the cost of renting the cycle.
- `next`: A pointer to the next `Student` object in the list.

## Functions

The system defines several functions to manage the cycle rental operations:

- `isEmpty()`: This function checks if the cycle list is empty.

- `popCycle()`: This function decreases the top of the cycle list.

- `pushCycle()`: This function increases the top of the cycle list.

- `pushTime(const string &t)`: This function converts the time from a string format to seconds.

- `rentCycle()`: This function allows a student to rent a cycle. It prompts the user to enter the student id and out time. If a cycle is available, the student is assigned a cycle and the cycle is marked as occupied.

- `returnCycle()`: This function allows a staff member to return a cycle. It prompts the user to enter the student id and in time. If a cycle is returned, the student is removed from the cycle and the cycle is marked as unoccupied.

- `pushStudent(string id, const string& out)`: This function adds a student to the top of the student list.

- `pushStudentIn(const string& id, const string& in)`: This function updates the in time and total time for a student.

- `cycleMenu()`: This function provides a menu for the cycle rental operations. It allows the user to rent a cycle, return a cycle, or exit the cycle rental system.

## Usage

To use the system, you need to include the `DS_Cycle.h` header file in your program and call the `cycleMenu()` function. The system will then display a menu and prompt the user to enter their choice. The user can perform various operations such as renting a cycle, returning a cycle, or exiting the cycle rental system.

## Conclusion

The Cycle Rental System is a robust tool for managing cycle rental operations. It provides a user-friendly command-line interface that allows easy interaction with the system. The system is designed to be flexible and easy to customize, making it a great choice for managing cycle rental operations.

## Note

Please note that this system does not persist data between different runs. If you need to save the data between different runs, you will need to implement a data storage system such as a database or a file system.

Hostel Management System Documentation

```
// student reg// done
// fee// default 1000 million dollars
// room reg// done
// seat reg// done
// food reg// baad

// in out// done

#include <iostream>
#include <string>
#include "header.h"

using namespace std;

struct Room;
struct HostelStudent;

Room* createRoom(int roomNumber, int totalSeats);
Room* rentRoom(Room* head);
void displayRooms(Room* head);
void checkOut(Room* head);

Room* headRoom = nullptr;
Room* tailRoom = nullptr;
void hostel ();

// Structure for hostel student
struct HostelStudent
{
    int id;
    string checkInDate;
    string checkOutDate;
    int fee;
    HostelStudent* next;
};

// Structure for room
struct Room
{
```

```cpp
    int roomNumber;
    int totalSeats;
    HostelStudent** seatStudents; // Array of pointers
    bool* isSeatOccupied;
    Room* nextRoom;
};

// Create a room
Room* createRoom(int roomNumber, int totalSeats)
{
    Room* newRoom = new Room;
    newRoom->roomNumber = roomNumber;
    newRoom->totalSeats = totalSeats;
    newRoom->seatStudents = new HostelStudent*[totalSeats](); // Initialize to
nullptr
    newRoom->isSeatOccupied = new bool[totalSeats](); // Initialize to false
    newRoom->nextRoom = nullptr;

    // Link the new room to the list
    if (tailRoom == nullptr)
    {
        headRoom = newRoom;
        tailRoom = newRoom;
    }
    else
    {
        tailRoom->nextRoom = newRoom;
        tailRoom = newRoom;
    }

    return newRoom;
}

// Rent a room function
Room* rentRoom(Room* head)
{
    int roomId;
    cout << "Enter the room number to rent: ";
    cin >> roomId;

    Room* currentRoom = head;

    // Find the requested room
    while (currentRoom != nullptr && currentRoom->roomNumber != roomId)
    {
        currentRoom = currentRoom->nextRoom;
    }

    if (currentRoom == nullptr)
    {
        cout << "Room not found." << endl;
        return head;
    }

    // Check for available seats
    int availableSeatIndex = -1;
    for (int i = 0; i < currentRoom->totalSeats; ++i)
    {
        if (!currentRoom->isSeatOccupied[i])
        {
            availableSeatIndex = i;
            break;
        }
    }

    if (availableSeatIndex == -1)
    {
        cout << "No available seats in the room." << endl;
        return head;
    }

    // Rent the seat to a hostel student
    HostelStudent* newStudent = new HostelStudent;
    cout << "Enter student id: ";
    cin >> newStudent->id;
    cout << "Enter check-in date (dd-mm-yy): ";
    cin >> newStudent->checkInDate;

    currentRoom->seatStudents[availableSeatIndex] = newStudent;
    currentRoom->isSeatOccupied[availableSeatIndex] = true;

    cout << "Student " << newStudent->id << " rented a seat in room " <<
currentRoom->roomNumber << endl;

    return head;
}

// Display information about available rooms
void displayRooms(Room* head)
{
    Room* currentRoom = head;

    cout << "Available Rooms:" << endl;

    while (currentRoom != nullptr)
    {
        cout << "Room " << currentRoom->roomNumber << ": ";

        for (int i = 0; i < currentRoom->totalSeats; ++i)
        {
            if (!currentRoom->isSeatOccupied[i])
            {
                cout << "Seat " << (i + 1) << " (Available) ";
            }
            else
            {
                cout << "Seat " << (i + 1) << " (Occupied by Student " <<
currentRoom->seatStudents[i]->id << ") ";
            }
        }

        cout << endl;

        currentRoom = currentRoom->nextRoom;
    }
}

// Check out a student function
void checkOut(Room* head)
{
    int studentId;
    cout << "Enter student id to check out: ";
    cin >> studentId;

    Room* currentRoom = head;

    // Find the room with the student
     while (currentRoom != nullptr)
    {
        for (int i = 0; i < currentRoom->totalSeats; ++i)
        {
            if (currentRoom->isSeatOccupied[i] && currentRoom->seatStudents[i]-
>id == studentId)
            {
                // Check out the student
```

```cpp
                currentRoom->isSeatOccupied[i] = false;

                HostelStudent* studentToCheckOut = currentRoom->seatStudents[i];

                cout << "Enter check-out date (dd-mm-yy) for Student " <<
studentId << ": ";
                cin >> studentToCheckOut->checkOutDate;

                cout << "Student " << studentId << " checked out from seat in
room " << currentRoom->roomNumber << endl;
                delete studentToCheckOut; // Release memory allocated for the
student
                currentRoom->seatStudents[i] = nullptr; // Set the pointer to
nullptr
                return;
            }
        }

        currentRoom = currentRoom->nextRoom;
    }

    // Student not found
    cout << "Student not found in any room." << endl;
}


void hostel ()
{
    // Create rooms with 4 seats
    headRoom = createRoom(101, 4);
    tailRoom = createRoom(102, 4);


    while (true)
    {
        cout << "Menu:" << endl;
        cout << "1. Display available rooms" << endl;
        cout << "2. Rent a room" << endl;
        cout << "3. Check out a student" << endl;
        cout << "4. Exit" << endl;

        int choice;
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice)
        {
        case 2:
            rentRoom(headRoom);
            break;
        case 1:
            displayRooms(headRoom);
            break;
        case 3:
            checkOut(headRoom);
            break;
        case 4:
            cout << "Exiting hostel program." << endl;
            return;
        default:
            cout << "Invalid choice. Please try again." << endl;
            break;
        }
    }
}
//filename: DS_Hostel.cpp
```

# Hostel Management System Documentation

This document provides a detailed explanation of the Hostel Management System, a C++ program that manages hostel operations such as room allocation, student registration, and fee management. The system offers a user-friendly command-line interface for easy interaction.

## Structures

The system defines two main structures: `HostelStudent` and `Room`.

- `HostelStudent`: This structure represents a student in the hostel. It contains the following fields:
  - `id`: The unique identifier for the student.
  - `checkInDate`: The date when the student checked into the hostel.
  - `checkOutDate`: The date when the student checked out from the hostel.
  - `fee`: The fee paid by the student.
  - `next`: A pointer to the next student in the list.

- `Room`: This structure represents a room in the hostel. It contains the following fields:
  - `roomNumber`: The unique identifier for the room.
  - `totalSeats`: The total number of seats in the room.
  - `seatStudents`: An array of pointers to `HostelStudent` structures representing the students in each seat.
  - `isSeatOccupied`: An array of boolean values indicating whether each seat is occupied.
  - `nextRoom`: A pointer to the next room in the list.

## Functions

The system defines several functions to manage the hostel operations.

- `createRoom(int roomNumber, int totalSeats)`: This function creates a new room with the given room number and total seats. It initializes the `seatStudents` and `isSeatOccupied` arrays to the correct size and sets all seats as unoccupied. The function then adds the new room to the end of the room list.

- `void hostel()`: This function initializes the hostel by creating two rooms.

- `rentRoom(Room *head)`: This function allows a student to rent a room. It prompts the user to enter the room number and student details. If the room is found and there are available seats, the student is assigned to a seat in the room and the seat is marked as occupied.

- `displayRooms(Room *head)`: This function displays the status of all rooms in the hostel. It shows the room number and the status of each seat in the room.

- `checkOut(Room *head)`: This function allows a student to check out from the hostel. It prompts the user to enter the student id. If the student is found in any room, the student is removed from the room and the seat is marked as unoccupied.

- `hostelMenu(Room *head)`: This function provides a menu for the hostel management operations. It allows the user to display available rooms, rent a room, check out a student, or exit the hostel management.

## Usage

To use the system, you need to include the `DS_Hostel.h` header file in your program and call the `hostelMenu(Room *head)` function with the head of the room list as the argument. The system will then display a menu and prompt the user to enter their choice. The user can perform various operations such as displaying available rooms, renting a room, checking out a student, or exiting the hostel management.

## Conclusion

The Hostel Management System is a simple yet effective tool for managing hostel operations. It provides a user-friendly command-line interface that allows easy interaction with the system. The system is designed to be flexible and easy to use.

## Note

Please note that this system does not persist data between different runs. If you need to save the data between different runs, you will need to implement a data storage system such as a database or a file system.

Transport Management System Documentation

```
// DS_Transport.cpp
#include <iostream>
#include <string>
#include <cmath>
#include "header.h"

using namespace std;

const int BUS_CAPACITY = 20;
```

```cpp
// Structure to represent a regular student
struct RegularStudent
{
    int id;
    string location;
    string classStartTime;
    string classEndTime;
    RegularStudent *nextStudent; // Pointer to the next student in the list
    RegularStudent *connector;   // Pointer to the next student in the
transportation list
};

// Structure to represent transportation details
struct Transportation
{
    string location;
    int busCapacity;
    int availableSeats;
    int busDepartureTime;
    RegularStudent *passengerList;
};

// Function prototypes
RegularStudent *createRegularStudent(int id, const string &classStartTime, const
string &classEndTime, const string &location);
Transportation *createTransportation(const string &location);
void assignTransportation(RegularStudent *student, Transportation *transport);
void displayTransportationDetails(Transportation *transport);

void bus()
{
    RegularStudent *head = nullptr;
    int id;
    string classStartTime;
    string classEndTime;
    string location;
    int option;
    int countDMD = 0, countMPR = 0, countSVR = 0;

    cout << "Welcome to Student Information System\n";

    // Input loop to gather information about regular students
    while (true)
    {
        cout << "\nEnter Non-Hostel Student Details:\n";

        // Getting input from the user
        cout << "ID: ";
        cin >> id;

        cout << "Class Start Time: ";
        cin >> classStartTime;

        cout << "Class End Time: ";
        cin >> classEndTime;

        cout << "Location (DMD/MPR/SVR): ";
        cin >> location;

        // Counting students based on location
        if (location == "DMD")
            countDMD++;
        else if (location == "MPR")
            countMPR++;
        else if (location == "SVR")
            countSVR++;

        // Creating a new regular student and adding it to the linked list
        RegularStudent *newStudent = createRegularStudent(id, classStartTime,
classEndTime, location);

        newStudent->nextStudent = head;
        head = newStudent;

        // Asking the user if they want to add another regular student
        cout << "\nDo you want to add another Regular student? (1/0): ";
        cin >> option;
        if (option == 0)
        {
            break;
        }
    }

    // Displaying the summary of regular students
    cout << "\nSummary of Regular Students:\n";
    cout << "DMD Students: " << countDMD << "\n";
    cout << "MPR Students: " << countMPR << "\n";
    cout << "SVR Students: " << countSVR << "\n";
```

```cpp
    // Calculate buses required based on student count and assign buses
    int busesForDMD = ceil(static_cast<double>(countDMD) /
static_cast<double>(BUS_CAPACITY));
    int busesForMPR = ceil(static_cast<double>(countMPR) /
static_cast<double>(BUS_CAPACITY));
    int busesForSVR = ceil(static_cast<double>(countSVR) /
static_cast<double>(BUS_CAPACITY));

    // Create and assign buses
    Transportation *dmdTransport = createTransportation("DMD");
    Transportation *mprTransport = createTransportation("MPR");
    Transportation *svrTransport = createTransportation("SVR");

    for (int i = 0; i < busesForDMD; ++i)
        assignTransportation(head, dmdTransport);

    for (int i = 0; i < busesForMPR; ++i)
        assignTransportation(head, mprTransport);

    for (int i = 0; i < busesForSVR; ++i)
        assignTransportation(head, svrTransport);

    // Display transportation details
    cout << "\nTransportation Details:\n";
    displayTransportationDetails(dmdTransport);
    displayTransportationDetails(mprTransport);
    displayTransportationDetails(svrTransport);

    cout << "\nThank you for using Student Information System.\n";

}

// Function to create a new regular student
RegularStudent *createRegularStudent(int id, const string &classStartTime, const
string &classEndTime, const string &location)
{
    RegularStudent *newStudent = new RegularStudent;
    newStudent->id = id;
    newStudent->classStartTime = classStartTime;
    newStudent->classEndTime = classEndTime;
    newStudent->location = location;
    newStudent->nextStudent = nullptr;
    newStudent->connector = nullptr;
    return newStudent;
}
```

```cpp
// Function to create a new transportation
Transportation *createTransportation(const string &location)
{
    Transportation *transport = new Transportation;
    transport->location = location;
    transport->busCapacity = BUS_CAPACITY;
    transport->availableSeats = BUS_CAPACITY;
    transport->busDepartureTime = 80; // Bus departs 1 hour and 20 minutes before
the class start time
    transport->passengerList = nullptr;
    return transport;
}


// Function to assign transportation to students
void assignTransportation(RegularStudent *student, Transportation *transport)
{
    // Calculate the total number of assigned students for the location
    int assignedStudents = 0;
    RegularStudent *currentStudent = student;
    while (currentStudent != nullptr)
    {
        if (currentStudent->location == transport->location)
        {
            assignedStudents++;
        }
        currentStudent = currentStudent->nextStudent;
    }

    // Assign transportation to students based on the summary
    for (int i = 0; i < assignedStudents && transport->availableSeats > 0; ++i)
    {
        RegularStudent *newPassenger = new RegularStudent(*student); // Create a
copy of the student
        newPassenger->connector = transport->passengerList;
        transport->passengerList = newPassenger;
        transport->availableSeats--;

        // Move to the next student
        student = student->nextStudent;
    }
}

// Function to display transportation details
// Function to display transportation details
void displayTransportationDetails(Transportation *transport)
{
    cout << "Transportation for " << transport->location << ":\n";

    // Calculate the total number of assigned students for the location
    int assignedStudents = 0;
    RegularStudent *currentPassenger = transport->passengerList;
    while (currentPassenger != nullptr)
    {
        assignedStudents++;
        currentPassenger = currentPassenger->connector;
    }

    cout << "Number of Buses: " << ceil(static_cast<double>(assignedStudents) /
static_cast<double>(BUS_CAPACITY))
         << " | Bus Capacity: " << transport->busCapacity
         << " | Assigned Students: " << assignedStudents
         << " | Available Seats: " << transport->availableSeats
         << "\nBus Departure Time: " << transport->busDepartureTime << " minutes
before class start\n";
}
//filename: DS_Transport.cpp
```

# Transport Management System Documentation

This document provides a detailed explanation of the Transport Management System implemented in the provided C++ code. The TMS is part of a larger student management system, designed to manage the transportation of students based on their location.

## Overview

The Transport Management is designed to handle the transportation of students from different locations (DMD/Dhanmondi, MPR/Mirpur, SVR/Savar) to their classes. It uses a linked list to store information about each student, and another linked list to store information about each bus. The system calculates the number of buses required based on the number of students in each location and assigns the students to the buses accordingly.

## Code Structure

The code is structured around two main data structures: `RegularStudent` and `Transportation`.

### RegularStudent

The `RegularStudent` structure represents a student. It contains the following fields:

- `id`: The student's ID.
- `location`: The student's location (DMD, MPR, SVR).
- `classStartTime`: The start time of the student's class.
- `classEndTime`: The end time of the student's class.
- `nextStudent`: A pointer to the next student in the list.
- `connector`: A pointer to the next student in the transportation list.

### Transportation

The `Transportation` structure represents a bus. It contains the following fields:

- `location`: The location of the bus.
- `busCapacity`: The maximum number of students that the bus can carry.
- `availableSeats`: The number of available seats in the bus.
- `busDepartureTime`: The time at which the bus departs.
- `passengerList`: A pointer to the list of students on the bus.

## Functions

The code includes several functions to manage the transportation system:

- `createRegularStudent`: Creates a new student and adds it to the linked list.
- `createTransportation`: Creates a new bus.

- `assignTransportation`: Assigns students to buses based on their location.

- `displayTransportationDetails`: Displays the details of each bus, including the number of buses, bus capacity, number of assigned students, available seats, and bus departure time.

## Usage

The `transportMenu` function is the main entry point of the System.

- It prompts the user to enter the details of each student

- Then it calculates the number of buses required based on the number of students in each location

- After that it creates the buses, assigns the students to the buses

- And finally displays the transportation details.

## Conclusion

The TMS is a simple system for managing the transportation of students. It uses linked lists to store and manage data, and provides functions for creating students and buses, assigning students to buses, and displaying transportation details. The system is designed to be flexible and allowing for easy addition of new locations and buses.

Header

```c
#ifndef HEADER
#define HEADER

struct Room;
struct HostelStudent;
struct RegularStudent;
struct Student;
struct Transportation;

Room* createRoom(int roomNumber, int totalSeats);
Room* rentRoom(Room* head);


void hostel();
void cycle();
void bus();

#endif
```

```
//filename: header.h
```

the header file contains the function and structure declaration for the functions and structures defined in the files:

- 1. DS_Cycle.cpp

- 2. DS_Transport.cpp

- 3. DS_Hostel.cpp

And it connects the other source files to the main file admin.cpp for use.

It contains the following declarations:

struct Room

struct HostelStudent

struct RegularStudent

struct Student

struct Transportation

Room* createRoom (int roomNumber, int totalSeats)

Room* rentRoom (Room* head)

Void hostel()

Void cycle()

Void bus()