BUS MANAGEMENT SYSTEM

Introduction to Bus Management System:

The Bus Management System is a comprehensive software solution designed to streamline and automate various aspects of bus reservation, passenger record keeping, and complaint management. This project is implemented in C++ and leverages data structures such as Binary Search Trees (BST) and Graphs to efficiently manage and organize information.

Project Overview:

In today's fast-paced world, efficient transportation systems are essential for the smooth functioning of cities and regions. The Bus Management System serves as a centralized platform for managing bus-related activities, providing a user-friendly interface for both passengers and administrators.

Key Features:

1. Bus Reservation:

- The system allows the reservation of buses, enabling administrators to register information such as bus serial number, bus number, driver details, and fare.

- Binary Search Trees (BST) are employed to efficiently organize and retrieve bus information.

2. Seat Booking and Cancellation:

- Passengers can book and cancel seats in registered buses, ensuring efficient utilization of available seats.

- The BST structure is utilized to manage seat information, allowing quick retrieval and modification of seat status.

3. Passenger Record Management:

- The system maintains detailed passenger records, including name, phone number, CNIC, source, destination, seat number, and charges.

- Binary Search Trees are employed to organize and search for passenger records efficiently.

4. Complaint Management:

- Passengers can register complaints related to bus services.

- A Priority Queue data structure is implemented to manage and prioritize complaints, ensuring efficient handling by administrators.

5. Graph Representation for Distance Calculation:

- The system incorporates a graph data structure to represent the distance between different cities, facilitating route planning and optimization.

- Graph algorithms are utilized to find the maximum and minimum distances within the network of cities.

Data Structures Selection:

- Binary Search Trees (BST):

- BSTs are employed for efficient storage and retrieval of bus information, ensuring quick access to details such as bus serial number, bus number, and driver information.

- Priority Queue:

- A Priority Queue is used to manage and prioritize passenger complaints, allowing administrators to address critical issues promptly.

- Graphs:

- Graphs are implemented to represent the network of cities and the distances between them, enabling route optimization and distance-related calculations.

Conclusion:

The Bus Management System presented here demonstrates the effective use of various data structures to create a robust and organized solution for bus-related operations. The chosen data structures contribute to the efficiency and responsiveness of the system, ensuring a seamless experience for both administrators and passengers in managing bus services.

BASIC DECLARATION:

```
#include<iostream>

using namespace std;

#define INF 99999
#define V 7

int index = 0;
int path[V];
int main();

string city[V] = {"Islamabad", "Multan", "Faislabad", "Lahore", "Sadiq Abad", "Quetta", "Karachi"};
```

It includes the <iostream> header for input and output operations and uses the namespace std. The defined constants include INF with a value of 99999 and V with a value of 7. It initializes an index variable and an array path with a size of V. Additionally, it declares an array city containing names of cities.

RECORD STRUCTURE:

```
// Passenger record
struct record{
    string name;
    string phone_no;
    string cnic;
    string source;
    string destination;
    int seat_no;
    int charges;
    record* next = NULL;
};
```

This code snippet defines a structure named record, representing a passenger record in a bus management system. The structure includes member variables such as name, phone_no, cnic (CNIC number), source, and destination, storing information about the passenger. Additionally, it includes seat_no for the seat number and charges for the fare. The structure also has a self-referential pointer next, initialized to NULL, indicating a linked list structure.

SOME ABSTRACT METHODS:

```
27  void Deletion(string cnic);
28
29  void insert(string name, string phone_no, string cnic,
30  string destination, string source, int seat_no, int charges, string bus_serial_number);
31
32  void display();
33
34  void enter_Complain(string complain);
35
36  void delete_Complain();
37
38  void display_complains();
```

PRI AND BUS TREE STRUCTURE:

```
40  //complain
41  struct pri {
42      string complain;
43      pri* next;
44  };
45
46
47
48  struct bus_tree{
49      int serial_no;
50      string bus_number;
51      string driver_name;
52      string cnic;
53      string phone_no;
54      int bus_fair;
55      int height;
56
57      bus_tree* left = NULL;
58      bus_tree* right = NULL;
59      bool seats[30];
60      string names[30];
61
62      record *record_start = NULL;
63      record *record_end = NULL;
64
65          pri *complains_front = NULL;
66      pri *complains_rear = NULL;
67  };
```

The pri structure represents a node in a linked list for complaints. It contains a string variable complain to store the complaint information and a self-referential pointer next pointing to the next node in the linked list.

The bus_tree structure represents a node in a binary search tree (BST) for bus information. It includes member variables to store details about a bus such as serial_no, bus_number, driver_name, cnic, phone_no, bus_fair, and height. The structure also has pointers for left and right children in the BST, an array seats to represent seat availability, and an array names to store passenger names for each seat. Additionally, it includes pointers for linked lists (record_start, record_end) to manage passenger records and (complains_front, complains_rear) for the linked list of complaints.

SOME MORE ABSTRACT METHODS FOR AVL TREES IMPLEMENTATION:

```
74   int getbf(bus_tree* p);
75   int gethight(bus_tree* p);
76   bus_tree* leftR(bus_tree* p);
77   bus_tree* rightR(bus_tree* p);
78   bus_tree* insert(bus_tree* r, string ser_no, string bus_no, string name, string cnic, string phone_no ,int fair);
79
80   bus_tree* creat(int key);
81
82   void pre_order(bus_tree* p);
83
84   void deletion(int x);
```

CLASS GRAPH:

-PRIVATE:

```
87  class Graph {
88  private:
89      int G[V][V];
90      string city[V] = {"Islamabad", "Multan", "Faislabad", "Lahore", "Sadiq Abad", "Quetta", "Karachi"};
91
92
93      int minDistance(int dist[], bool sptSet[]) {
94          int min = INF, min_index;
95
96          for (int v = 0; v < V; v++)
97              if (!sptSet[v] && dist[v] <= min)
98                  min = dist[v], min_index = v;
99
100         return min_index;
101     }
```

The Graph class has a 2D array G representing the adjacency matrix of the graph. It also includes an array city with the names of cities. The private member function minDistance is a helper function used in graph algorithms to find the vertex with the minimum distance value among the vertices not yet included in the shortest path tree.

-PUBLIC:

```
103 public:
104     Graph() {
105         // Initialize the graph matrix with the provided values
106         int matrix[V][V] = {
107             {0, 9, 0, 0, 2, 0, 1},
108             {9, 0, 5, 1, 2, 4, 6},
109             {7, 5, 0, 1, 2, 5, 0},
110             {0, 1, 1, 0, 1, 3, 0},
111             {2, 0, 2, 5, 0, 3, 6},
112             {0, 4, 5, 3, 3, 0, 0},
113             {1, 6, 0, 0, 6, 0, 0}
114         };
115 
116         for (int i = 0; i < V; i++)
117             for (int j = 0; j < V; j++)
118                 G[i][j] = matrix[i][j];
119     }
```

The public constructor of the Graph class initializes the graph matrix (G) with hardcoded values. It represents a weighted, undirected graph with 7 vertices (cities) and the edges between them. The matrix is populated with the provided values. This sets up the initial state of the graph for further graph algorithm implementations.

-PUBLIC METHODS FOR GRAPHS:

1. DISPLAY:

```
121     void displayGraph() {
122         cout << "City Graph:" << endl;
123         for (int i = 0; i < V; i++) {
124             for (int j = 0; j < V; j++) {
125                 cout << city[i] << " - " << city[j] << " : ";
126                 if (G[i][j] != INF) {
127                     cout << G[i][j] << " km";
128                 } else {
129                     cout << "Not Connected";
130                 }
131                 cout << endl;
132             }
133         }
```

The displayGraph function in the Graph class prints the details of the city graph. For each pair of cities (vertices) in the graph, it prints the distance between them if there is a direct connection; otherwise, it indicates that they are not connected.

2. MIN-MAX DISTANCE:

```
136     int findMaxDistance() {
137         int maxDistance = INT_MIN;
138 
139         for (int i = 0; i < V; i++)
140             for (int j = 0; j < V; j++)
141                 if (G[i][j] != INF)
142                     maxDistance = max(maxDistance, G[i][j]);
143 
144         return maxDistance;
145     }
146 
147     int findMinDistance() {
148         int minDistance = INF;
149 
150         for (int i = 0; i < V; i++)
151             for (int j = 0; j < V; j++)
152                 if (G[i][j] != INF)
153                     minDistance = min(minDistance, G[i][j]);
154 
155         return minDistance;
156     }
```

The findMaxDistance function in the Graph class returns the maximum distance between any two cities in the graph. It iterates through all pairs of cities and updates the maxDistance variable if it encounters a greater distance.

The findMinDistance function, on the other hand, returns the minimum distance between any two cities in the graph. It iterates through all pairs of cities and updates the minDistance variable if it encounters a smaller distance.

CLASS BST:

-INSERTION:

```
159 class BST{
160     public:
161         bus_tree* insert(bus_tree* r, int ser_no, string bus_no, string name, string cnic, string phone_no ,int fair){
162             //pre_order(r);
163 
164             if( r == NULL){
165                 return creat(ser_no, bus_no, name, cnic, phone_no, fair);
166             }
167             else if(ser_no < r->serial_no){
168                 r->left = insert(r->left, ser_no, bus_no, name, cnic, phone_no, fair);
169             }
170             else if(ser_no > r->serial_no){
171                 r->right = insert(r->right, ser_no, bus_no, name, cnic, phone_no, fair);
172             }
173 
174             r->height = 1 + max(getheight(r->left), getheight(r->right));
175             int bf = getbf(r);
176 
177             if(bf > 1 && ser_no < r->left->serial_no){
178                 return rightR(r);
179             }
180 
181             if(bf < -1 && ser_no > r->right->serial_no){
182                 return leftR(r);
183             }
184 
185             if(bf > 1 && ser_no > r->left->serial_no){
186                 r->left = leftR(r->left);
187                 return rightR(r);
188             }
189 
190             if(bf < -1 && ser_no < r->right->serial_no){
191                 r->right = rightR(r->right);
192                 return leftR(r);
193             }
194 
195             return r;
196         }
```

Inserts a new node into the binary search tree (BST) with the given parameters. Performs rotations to maintain the balance of the BST.

-PRE-ORDER, IN-ORDER:

```
200         void pre_order(bus_tree* p){
201             if(p!=NULL){
202                 cout<<"  Serial No:  "<<p->serial_no<< endl;
203                 cout<<"  Bus Number:  "<<p->bus_number<< endl;
204                 cout<<"  Driver Name:  "<<p->driver_name<< endl;
205                 cout<<"  CNIC:  "<<p->cnic<< endl;
206                 cout<<"  Phone No:  "<<p->phone_no<< endl;
207                 cout<<"  Bus Fair:  "<<p->bus_fair << endl << endl;
208                 pre_order(p->left);
209                 pre_order(p->right);
210             }
211         }
212 
213         void in_order(bus_tree* p){
214             if(p!=NULL){
215                 in_order(p->left);
216                 cout<<p->serial_no<<"   ";
217                 in_order(p->right);
218             }
219         }
220 
```

Traverses the BST in pre-order (root, left, right) and prints information about each node. Traverses the BST in in-order (left, root, right) and prints information about each node.

-GET HEIGHT AND BALANCE FACTOR:

```
222     int getheight(bus_tree* p){
223         int h = 0;
224         if (p != NULL) {
225             int l_height = getheight(p->left);
226             int r_height = getheight(p->right);
227             int max_height = max(l_height, r_height);
228             h = max_height + 1;
229         }
230         return h;
231     }
232
233     int getbf(bus_tree* p){
234         int l_height = getheight(p->left);
235         int r_height = getheight(p->right);
236         int b_factor = l_height - r_height;
237         return b_factor;
238     }
239
```

Returns the height of a given node in the BST. Returns the balance factor of a given node in the BST.

-RIGHT R AND LEFT R:

```
242     bus_tree* rightR(bus_tree* p){
243         bus_tree* x = p->left;
244         bus_tree* k = x->right;
245
246         x->right = p;
247         p->left = k;
248
249
250         p->height = 1 + max(getheight(p->left), getheight(p->right));
251         x->height = 1 + max(getheight(p->left), getheight(p->right));
252         //pre_order(x);
253         return x;
254     }
255
256
257
258     bus_tree* leftR(bus_tree* p){
259         bus_tree* x = p->right;
260         bus_tree* k = x->left;
261
262         x->left = p;
263         p->right = k;
264
265
266         p->height = 1 + max(getheight(p->left), getheight(p->right));
267         x->height = 1 + max(getheight(p->left), getheight(p->right));
268
269         return x;
270
271     }
```

Performs a right rotation on the given node to balance the BST. Performs a left rotation on the given node to balance the BST

-CREATE:

```
bus_tree* creat(int ser_no, string bus_no, string name, string cnic, string phone_no ,int fair){
    bus_tree* ptr = new bus_tree();
    ptr->serial_no = ser_no;
    ptr->bus_fair = fair;
    ptr->bus_number = bus_no;
    ptr->driver_name = name;
    ptr->phone_no = phone_no;
    ptr->cnic = cnic;


    ptr->left = NULL;
    ptr->right = NULL;
    ptr->height = 1;

    return ptr;
}
```

Creates a new node with the provided information.

-DELETION:

```
//      Deletion
void deletion(int x) {
    bus_tree *p = root;
    bus_tree *k;
    while (p != NULL && p->serial_no != x) {
        k = p;
        if (x > p->serial_no) {
            p = p->right;
        } else p = p->left;
    }
    // Leaf Node
    if (p->left == NULL && p->right == NULL) {
        if (p == k->right) {
            k->right = NULL;
        } else k->left = NULL;
        delete p;
    } else if (p->right == NULL || p->left == NULL) { //Single Branches
        if (p->right == NULL) {
            if (p == k->left) {
                k->left = p->left;
            }else{
                k->right=p->left;
            }delete p;
        }
        else
        {
            if(p==k->right){
                k->right=p->right;
            }else{
                k->left=p->right;
            }
            delete p;
        }
    }
```

Deletes a node with the specified serial number from the BST.

Handles cases for leaf nodes, nodes with a single child, and nodes with two children.

```
            else{
                bus_tree* p2 = p->right;
                bus_tree* p3 = p2;
                while(p2->left != NULL){
                   p3 = p2;
                   p2 = p2->left;
                }
                if(p2->left == NULL && p2->right == NULL){

                   p->serial_no = p2->serial_no;
                   p->bus_number = p2->bus_number;
                   p->driver_name = p2->driver_name;
                   p->cnic = p2->cnic;
                   p->phone_no = p2->phone_no;
                   p->bus_fair = p2->bus_fair;

                   if(p3 == p2)
                       p->right = NULL;
                   else p3->left = NULL;
                }
                else{
                   p->serial_no = p2->serial_no;
                   if(p2->right != NULL)
                       p3->left = p2->right;
                    else
                       p3->left = NULL;
                }
            }
            cout << "Record deleted" << endl;
        }
```

-SEARCH:

```
    bus_tree* search(int bus_serial_number){
        bus_tree *p = root;

        while (p->serial_no != bus_serial_number && p!=NULL){
            if (bus_serial_number > p->serial_no){
                p = p->right;
            }
            else{
                p = p->left;
            }
        }

        return p;
    }
```

Searches for a node with the given bus serial number in the BST and returns a pointer to it.

-**BOOK SEATS:**

```
370     void bookseats(int seat, int bus_serial_number){
371         try{
372
373
374         bus_tree *bus = search(bus_serial_number);
375
376         if(bus == NULL){
377             cout<<"Bus with above mentioned serial number is not regestered"<<endl;
378             return;
379         }
380         else{
381
382             if(bus->seats[seat-1]){
383                 cout<<"  Invalid Seat booked"<<endl;
384             }
385
386             else if(seat<30){
387                 string name,source, destination;
388                 cout<<"Enter your name"<<endl;
389                 cin>>name;
390                 bus->seats[seat-1] = true;
391                 bus->names[seat-1] = name;
392
393                 for(int i = 0; i<V; i++){
394                     cout<<i+1<<"  "<<city[i]<<endl;
395                 }
396                 cout<<"From where you have started your journey??"<<endl;
397                 cin>>source;
398                 cout<<"Where you want to go??"<<endl;
399                 for(int i = 0; i<V; i++){
400                     cout<<i+1<<"  "<<city[i]<<endl;
401                 }
402                 cin>>destination;
403
404             }
405
406             else{
407                 cout<<"Seat not available"<<endl;
408             }
409         }
410     }catch (const exception& e) {
411     cerr << "Exception in bookseats function: " << e.what() << endl;
412     }
413     }
```

Books a seat for a passenger on a specified bus. Takes the seat number, bus serial number, and passenger information. Checks seat availability and handles invalid seat bookings.

-**CANCEL BOOKING:**

```
416         void cancel_booking(int seat, int bus_serial_number){
417             try{
418             bus_tree *bus = search(bus_serial_number);
419             if(bus == NULL){
420                 cout<<"Bus with above mentioned serial number is not registered"<<endl;
421             }
422             else{
423                 cout<<"Enter your name"<<endl;
424                 string name;
425                 cin>>name;
426                 if(seat<30 && bus->seats[seat-1] == true && bus->names[seat-1]==name){
427                     bus->seats[seat-1] = false;
428                     bus->names[seat-1] = "";
429                 }
430
431                 if(bus->names[seat-1]!=name){
432                     cout<<"Seat is not booked on the entered name"<<endl;
433                 }
434                 if(seat>30){
435                     cout<<"Seat not available"<<endl;
436                 }
437                 if(!bus->seats[seat-1]){
438                     cout<<"Seat is not booked"<<endl;
439
440
441                 }
442             }
443             }catch (const exception& e) {
444             cerr << "Exception in cancel_booking function: " << e.what() << endl;
445             }
```

Cancels a booking for a specified seat on a given bus. Takes the seat number, bus serial number, and passenger name. Handles cases where the seat is not booked or the provided name doesn't match the booked name.

-**SHOW BOOKING:**

```
450         void show_booking(string password, int bus_serial_number){
451             try{
452
453             if(password == "1234"){
454                 bus_tree *bus = search(bus_serial_number);
455
456                 if(bus == NULL){
457                     cout<<"No bus of above mentioned serial number is registered"<<endl;
458                 }
459
460                 else{
461                     for(int i =0; i<30; i++){
462                         if(bus->seats[i]){
463                             cout<<"Seat no: "<<i+1<<"\nName: "<<bus->names[i]<<endl<<endl;
464                         }
465                     }
466                 }
467
468             }
469             else{
470                 cout << "Invalid password" << endl;
471             }
472         }catch (const exception& e) {
473         cerr << "Exception in show_booking function: " << e.what() << endl;
474         }
475
476         }
```

Shows the booked seats for a specified bus, requires a password for access. Prints seat numbers and corresponding passenger names. Password protection for security.

CLASS PRI:

```
482 class Pri {
483 public:
484     Pri() {
485     }
486     BST t;
```

Initializes an instance of the Pri class. The Pri class contains an instance of the BST class named t for managing buses and their associated complaints. The linked list (pri) is used to store complaints for each bus in a FIFO manner.

-ENTER COMPLAIN:

```
488     void enter_Complain(int bus_serial_number) {
489         bus_tree* bus = t.search(bus_serial_number);
490         if (bus == NULL) {
491             cout << "Bus not registered" << endl;
492         } else {
493             string complain;
494             cout << "Enter complain: ";
495             getline(cin >> ws, complain);
496 
497             pri* newComplain = new pri;
498             newComplain->complain = complain;
499             newComplain->next = NULL;
500 
501             if (bus->complains_front == NULL) {
502                 bus->complains_front = bus->complains_rear = newComplain;
503             } else {
504 
505                 pri* node = bus->complains_front;
506                 pri* prev = NULL;
507 
508                 while (node != NULL) {
509                     prev = node;
510                     node = node->next;
511                 }
512 
513                 if (prev == NULL) {
514 
515                     newComplain->next = bus->complains_front;
516                     bus->complains_front = newComplain;
517                 } else {
518 
519                     prev->next = newComplain;
520                     bus->complains_rear = newComplain;
521                 }
522             }
523         }
524 
525         cout << "Complaint Submitted" << endl;
526     }
```

Allows users to enter a complaint for a specific bus identified by its serial number. Creates a new node for the complaint and adds it to the linked list of complaints associated with the bus.

-DELETE COMPLAIN:

```
528     void delete_Complain(int bus_serial_number) {
529         bus_tree* bus = t.search(bus_serial_number);
530         if (bus == NULL) {
531             cout << "Bus not registered" << endl;
532         } else {
533             pri* p = bus->complains_front;
534 
535             if (p == NULL) {
536                 cout << "No complaints registered" << endl;
537             } else {
538                 bus->complains_front = bus->complains_front->next;
539                 cout << "Complaint: " << p->complain << endl;
540                 delete p;
541             }
542         }
543     }
```

Deletes the first complaint in the linked list for a specified bus. Prints the deleted complaint.

-DISPLAY COMPLAIN:

```
545     void display_complains(int bus_serial_number) {
546         bus_tree* bus = t.search(bus_serial_number);
547 
548         if (bus == NULL) {
549             cout << "Bus not registered" << endl;
550         } else {
551             cout << "Complaints for Bus Serial Number " << bus_serial_number << ":" << endl;
552             pri* p = bus->complains_front;
553 
554             if (p == NULL) {
555                 cout << "No complaints registered" << endl;
556             } else {
557                 while (p != NULL) {
558                     cout << "Complaint: " << p->complain << endl;
559                     p = p->next;
560                 }
561             }
562         }
563     }
564 };
```

Displays all complaints associated with a specific bus identified by its serial number. Iterates through the linked list of complaints and prints each complaint.

CLASS RECORD:

The Record class contains an instance of the BST class named t for managing buses and their associated passenger records. The linked list (record) is used to store passenger records for each bus.

-INSERT:

```
566 class Record{
567
568     public:
569         BST t;
570         void insert(string name, string phone_no, string cnic, string destination, string source, int seat_no, int charges, int bus_serial_number){
571             try{
572
573             bus_tree *bus = t.search(bus_serial_number);
574
575             if (bus == NULL){
576                 cout<<"Bus not registered"<<endl;
577             }
578
579             else{
580                 record* p = new record();
581                 p->name = name;
582                 p->phone_no = phone_no;
583                 p->cnic = cnic;
584                 p->destination = destination;
585                 p->source = source;
586                 p->seat_no = seat_no;
587                 p->charges = charges;
588
589                 if(bus->record_start == NULL){
590                     bus->record_start = bus->record_end = p;
591                 }
592                 else{
593                     bus->record_end->next = p;
594                     bus->record_end = p;
595                 }
596             }
597
598             cout << "Record added" << endl;
599         }catch (const exception& e) {
600         cerr << "Exception in insert function: " << e.what() << endl;
601         }
602         }
603
```

Inserts a new passenger record into the linked list associated with a specific bus identified by its serial number. Creates a new record node and adds it to the end of the linked list.

-DELETION:

```
605         void deletion(int bus_serial_number){
606             bus_tree *bus = t.search(bus_serial_number);
607
608             if(bus == NULL){
609                 cout<<"Bus not regestered"<<endl;
610             }
611
612             else{
613                 string cnic;
614                 cout<<"Enter your cnic"<<endl;
615                 cin>>cnic;
616                 record* p = bus->record_start;
617                 record* k;
618
619                 while(p != NULL && p->cnic != cnic){
620                     k = p;
621                     p = p->next;
622                 }
623
624                 if(p == NULL){
625                     cout << "Record not found" << endl;
626                 }
627                 else{
628                     if(p == bus->record_start){
629                         bus->record_start = bus->record_start->next;
630                     }
631                     else if(p == bus->record_end){
632                         bus->record_end = k;
633                     }
634                     else{
635                         k->next = p->next;
636                     }
637                 }
638             }
639
640             cout << "Record deleted" << endl;
641         }
642
```

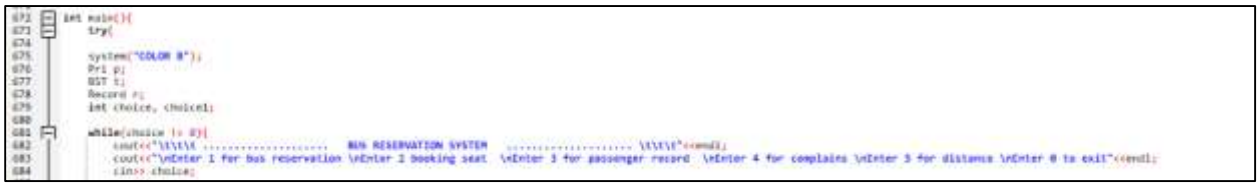
Deletes a passenger record from the linked list associated with a specific bus based on the passenger's CNIC. Finds the record using the CNIC, adjusts pointers, and deallocates memory.

-DISPLAY:

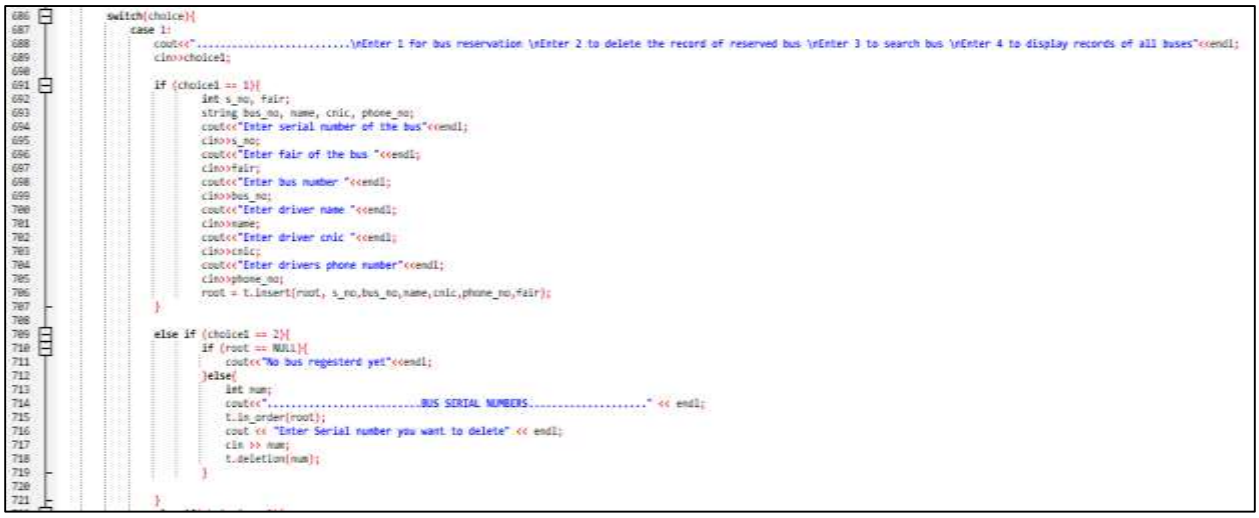
```
643
644         void display(int bus_serial_number){
645             bus_tree *bus = t.search(bus_serial_number);
646
647             if(bus == NULL){
648                 cout<<"Bus not regestered"<<endl;
649             }
650             else{
651                 record* p = bus->record_start;
652
653                 while(p != NULL){
654                     cout << " Name: "<< p->name<< endl;
655                     cout << " Phone No: "<< p->phone_no<< endl;
656                     cout << " CNIC: "<< p->cnic<< endl;
657                     cout << " Source: "<< p->source<< endl;
658                     cout << " Destination: "<< p->destination<< endl;
659                     cout << " Seat No: "<< p->seat_no<< endl;
660                     cout << " Charges: " << p->charges<< endl<< endl;
661
662                     cout << endl;
663
664                     p = p->next;
665                     }
666                 }
667             }
668
669 };
```

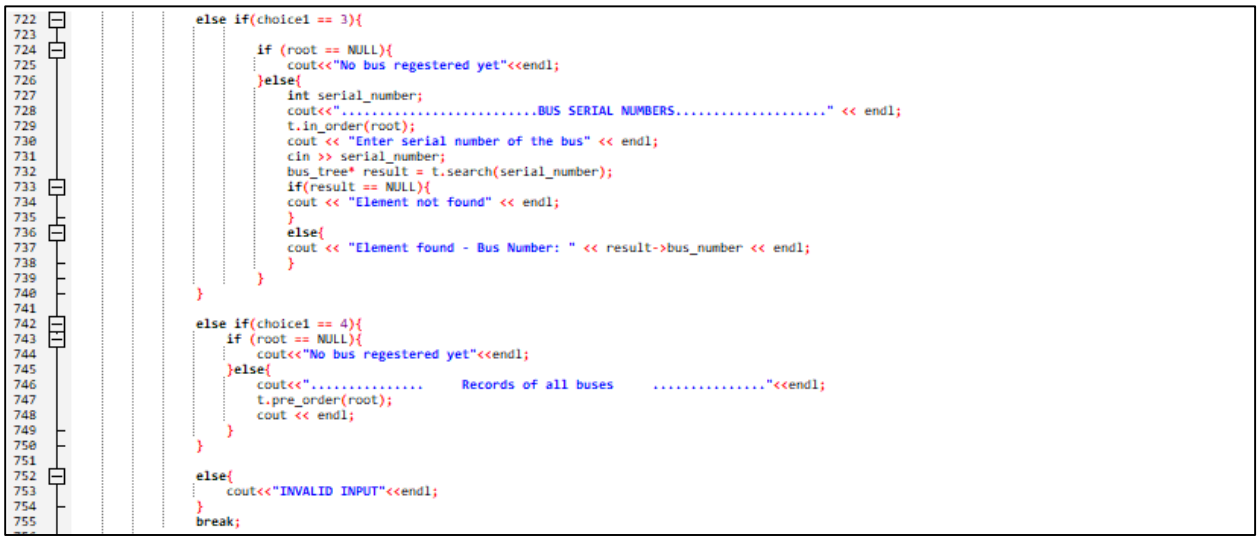
Displays all passenger records associated with a specific bus identified by its serial number. Iterates through the linked list of records and prints each passenger's details.

MAIN FUNCTION:



-CASE 1:





Sub-menu to perform bus reservation-related tasks.

Options include adding a new bus, deleting a bus record, searching for a bus, and displaying records of all buses.

-CASE 2:

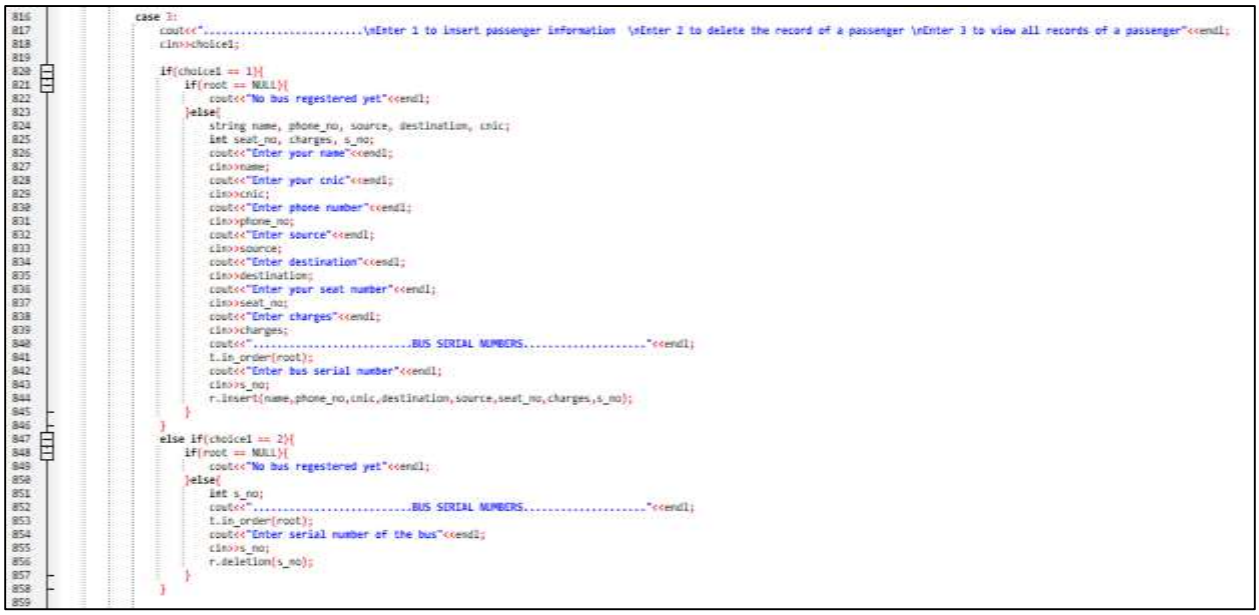
```
757             case 2:
758                 cout<<".....\nEnter 1 to book seat \nEnter 2 to cancel booking \nEnter 3 to display all bookings"<<endl;
759                 cin>>choice1;
760 
761                 if(choice1 == 1){
762                     if(root == NULL){
763                         cout<<"No bus regestered yet"<<endl;
764                     }else{
765                         int s_no, seat;
766                         cout<<".....BUS SERIAL NUMBERS....." << endl;
767                         t.in_order(root);
768                         cout << endl;
769                         cout<<"Enter serial number of the bus"<<endl;
770                         cin>>s_no;
771                         cout<<"Enter seat number"<<endl;
772                         cin>>seat;
773                         t.bookseats(seat, s_no);
774                     }
775                 }
776 
777                 else if(choice1 == 2){
778                     if(root == NULL){
779                         cout<<"No bus regestered yet"<<endl;
780                     }else{
781                         int s_no, seat;
782                         cout<<".....BUS SERIAL NUMBERS....."<<endl;
783                         t.in_order(root);
784                         cout << endl;
785                         cout<<"Enter serial number of the bus"<<endl;
786                         cin>>s_no;
787                         cout<<"Enter seat number"<<endl;
788                         cin>>seat;
789                         t.cancel_booking(seat,s_no);
790                     }
791                 }
```

```
793                 else if(choice1 == 3){
794                     if(root == NULL){
795                         cout<<"No bus regestered yet"<<endl;
796                     }else{
797                         int s_no;
798                         string password;
799                         cout<<".....BUS SERIAL NUMBERS....."<<endl;
800                         t.in_order(root);
801                         cout << endl;
802                         cout<<"Enter serial number of the bus"<<endl;
803                         cin>>s_no;
804                         cout<<"Enter password"<<endl;
805                         cin>>password;
806                         t.show_booking(password, s_no);
807                     }
808                 }
809                 else{
810                     cout<<"INVALID INPUT"<<endl;
811                 }
812                 break;
813 
```

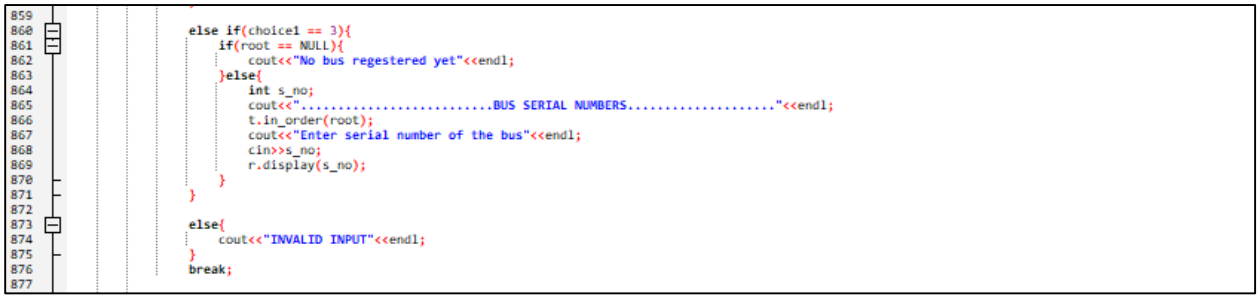
Sub-menu for booking seats, canceling bookings, and displaying all bookings.

Options include booking a seat, canceling a booking, and displaying all bookings for a specific bus

-CASE 3:



```
case 3:
    cout<<".....\nEnter 1 to insert passenger information  \nEnter 2 to delete the record of a passenger \nEnter 3 to view all records of a passenger"<<endl;
    cin>>choice1;

    if(choice1 == 1){
        if(root == NULL){
            cout<<"No bus regestered yet"<<endl;
        }else{
            string name, phone_no, source, destination, cnic;
            int seat_no, charges, s_no;
            cout<<"Enter your name"<<endl;
            cin>>name;
            cout<<"Enter your cnic"<<endl;
            cin>>cnic;
            cout<<"Enter phone number"<<endl;
            cin>>phone_no;
            cout<<"Enter source"<<endl;
            cin>>source;
            cout<<"Enter destination"<<endl;
            cin>>destination;
            cout<<"Enter your seat number"<<endl;
            cin>>seat_no;
            cout<<"Enter charges"<<endl;
            cin>>charges;
            cout<<".....BUS SERIAL NUMBERS....."<<endl;
            t.in_order(root);
            cout<<"Enter bus serial number"<<endl;
            cin>>s_no;
            r.insert(name,phone_no,cnic,destination,source,seat_no,charges,s_no);
        }
    }
    else if(choice1 == 2){
        if(root == NULL){
            cout<<"No bus regestered yet"<<endl;
        }else{
            int s_no;
            cout<<".....BUS SERIAL NUMBERS....."<<endl;
            t.in_order(root);
            cout<<"Enter serial number of the bus"<<endl;
            cin>>s_no;
            r.deletion(s_no);
        }
    }
```



```
    else if(choice1 == 3){
        if(root == NULL){
            cout<<"No bus regestered yet"<<endl;
        }else{
            int s_no;
            cout<<".....BUS SERIAL NUMBERS....."<<endl;
            t.in_order(root);
            cout<<"Enter serial number of the bus"<<endl;
            cin>>s_no;
            r.display(s_no);
        }
    }

    else{
        cout<<"INVALID INPUT"<<endl;
    }
    break;
```

Sub-menu for managing passenger records.

Options include inserting passenger information, deleting a passenger's record, and viewing all records of a passenger.

-CASE 4:

```
879             case 4:
880                 cout<<".....\nEnter 1 to regester complain \nEnter 2 to delete 1st complain \nEnter 3 to view all regestered complains "<<endl;
881                 cin>>choice1;
882
883                 if(choice1 == 1){
884                     if(root == NULL){
885                         cout<<"No bus regestered yet"<<endl;
886                     }else{
887                         int s_no, age;
888                         cout<<".....BUS SERIAL NUMBERS....."<<endl;
889                         t.in_order(root);
890                         cout << endl;
891                         cout<<"Enter serial number of the bus"<<endl;
892                         cin>>s_no;
893                         p.enter_Complain(s_no);
894                     }
895                 }
896
897                 else if(choice1 == 2){
898                     if(root == NULL){
899                         cout<<"No bus regestered yet"<<endl;
900                     }else{
901                         int s_no;
902                         cout<<".....BUS SERIAL NUMBERS....." << endl;
903                         t.in_order(root);
904                         cout << endl;
905                         cout<<"Enter serial number of the bus"<<endl;
906                         cin>>s_no;
907                         cout<<"COMPLAIN:\n....."<<endl;
908                         p.delete_Complain(s_no);
909                     }
910                 }
```

```
912                 else if(choice1 == 3){
913                     if(root == NULL){
914                         cout<<"No bus regestered yet"<<endl;
915                     }else{
916                         int s_no;
917                         cout<<".....BUS SERIAL NUMBERS....."<<endl;
918                         t.in_order(root);
919                         cout << endl;
920                         cout<<"Enter serial number of the bus"<<endl;
921                         cin>>s_no;
922                         cout<<"COMPLAINS\n....."<<endl;
923                         p.display_complains(s_no);
924                     }
925                 }
926
927                 else{
928                     cout<<"INVALID INPUT"<<endl;
929                 }
930
931                 break;
```

Sub-menu for registering and managing complaints.

Options include registering a complaint, deleting the first complaint, and viewing all registered complaints for a specific bus.

-CASE 5:

```
932                 case 5:
933                 {
934                     cout<<".....................\nEnter 1 to display all distances \nEnter 2 to display max distance \nEnter 3 to display min distance "<<endl;
935                     cin>>choice1;
936                     Graph graph;
937
938                     if(choice1==1){
939
940
941                     graph.displayGraph();
942                     }
943                     else if(choice1==2){
944
945
946                     int maxDistance = graph.findMaxDistance();
947                     cout << "Maximum Distance in the Graph: " << maxDistance << " km" << endl;
948                     }
949
950                     else if(choice1==3){
951
952
953                     int minDistance = graph.findMinDistance();
954                     cout << "Minimum Distance in the Graph: " << minDistance << " km" << endl;
955                     }
956                     else{
957                     cout<<"INVALID INPUT"<<endl;
958                     }
959                 }
960                 break;
961
962             default:
963                 break;
964         }
965     }
966     }catch (const exception& e) {
967         cerr << "Exception in main function: " << e.what() << endl;
968     }
969     return 0;
970 }
```

Sub-menu for displaying distances between cities in a graph.

Options include displaying all distances, finding the maximum distance, and finding the minimum distance in the graph.

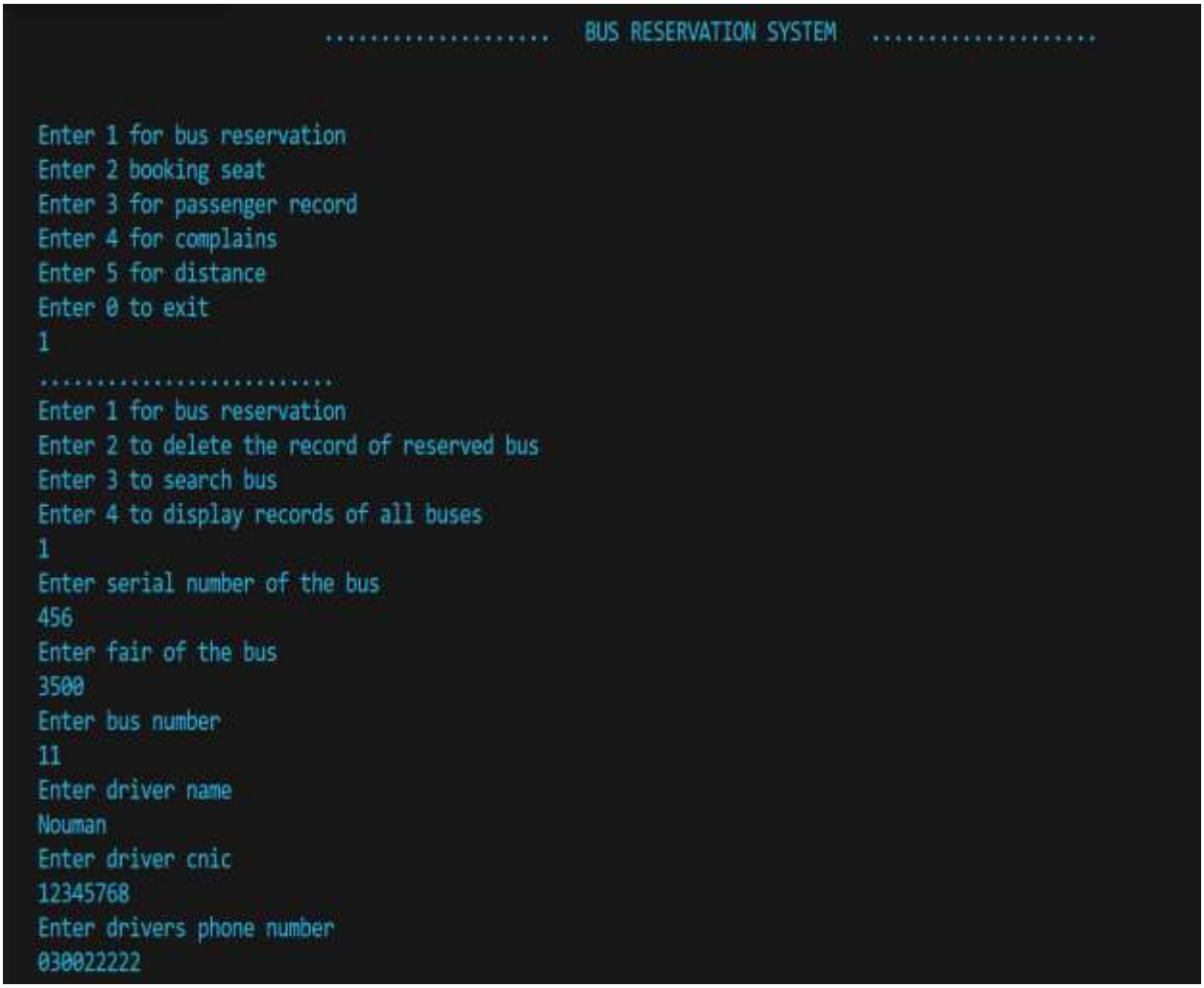
-DEFAULT CASE:

Handles invalid inputs by doing nothing.

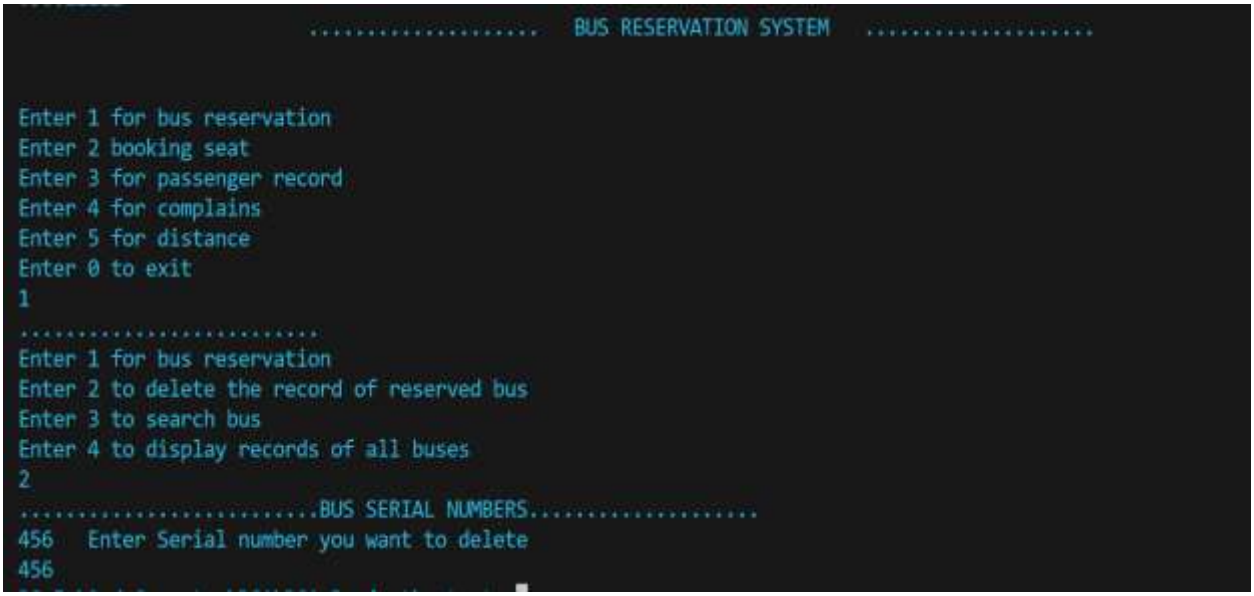
Each case corresponds to different functionalities of our bus reservation system, providing a structured and modular approach to interact with the system.

OUTPUTS

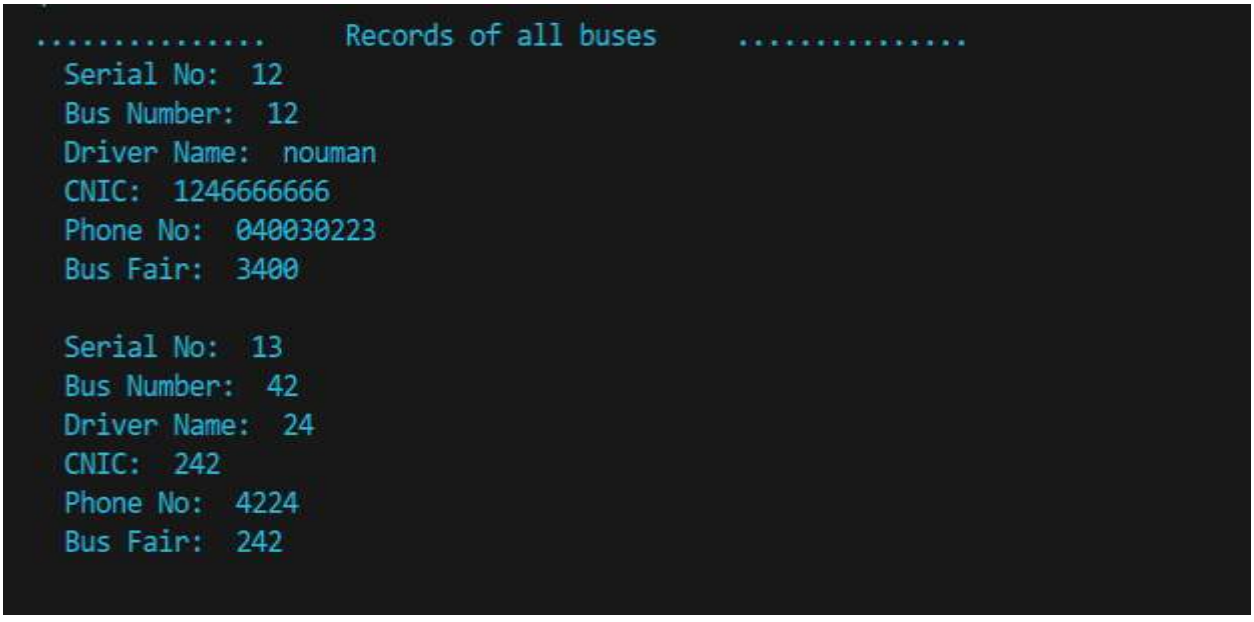
FOR 1



Here we deleted the bus that I registered before with the help of its serial number.



Here we displayed all buses record that I registered.



Here we deleted the bus from Record.

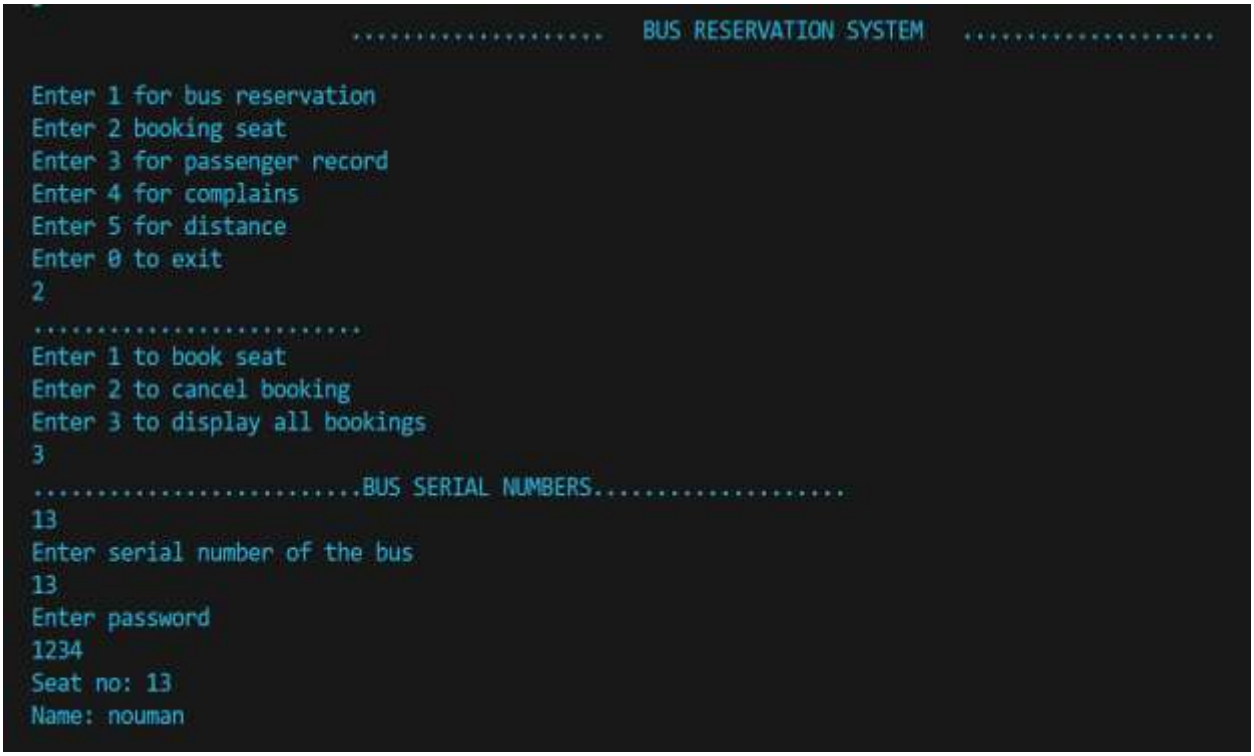
```
.........................BUS SERIAL NUMBERS.....................
11   112   Enter Serial number you want to delete
112
```

For 2

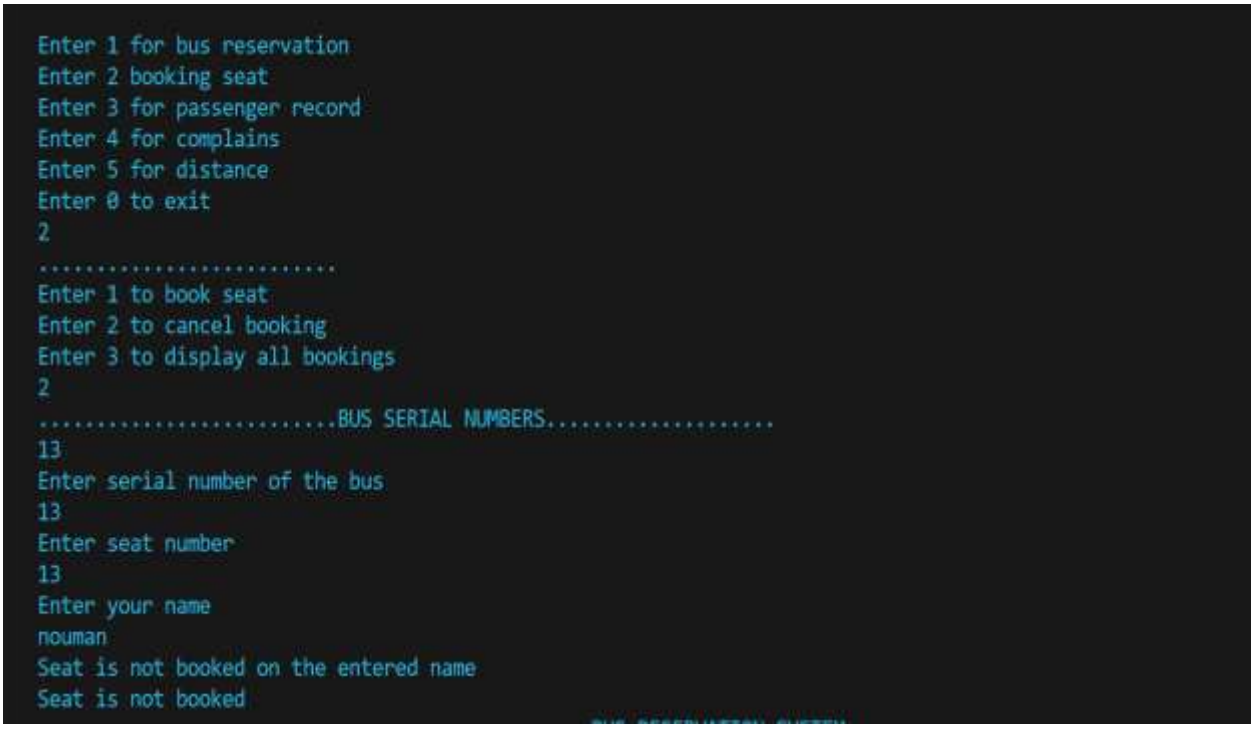
Here we booked a seat.

```
.........................
Enter 1 to book seat
Enter 2 to cancel booking
Enter 3 to display all bookings
1
.........................BUS SERIAL NUMBERS.....................
13
Enter serial number of the bus
13
Enter seat number
13
Enter your name
nouman
1  Islamabad
2  Multan
3  Faislabad
4  Lahore
5  Sadiq Abad
6  Quetta
7  Karachi
From where you have started your journey??
1
Where you want to go??
1  Islamabad
2  Multan
3  Faislabad
4  Lahore
5  Sadiq Abad
6  Quetta
```

Here we displayed booking.



Here we deleted booking but it didn't delete as I entered wrong name of passenger.



For 3

Here we added the record of passenger.

```
.........................
Enter 1 to insert passenger information
Enter 2 to delete the record of a passenger
Enter 3 to view all records of a passenger
1
Enter your name
nuoman
Enter your cnic
1234444
Enter phone number
3463566
Enter source
sadiqbad
Enter destination
islamad
Enter your seat number
13
Enter charges
4500
.........................BUS SERIAL NUMBERS.....................
13   Enter bus serial number
13
Record added
```

Here we displayed all the records of our passengers.

```
13   Enter serial number of the bus
13
 Name: nouman
 Phone No: 1
 CNIC: khalid
 Source: 1
 Destination: 1
 Seat No: 1
 Charges: 1


 Name: nuoman
 Phone No: 3463566
 CNIC: 1234444
 Source: sadiqbad
 Destination: islamad
 Seat No: 13
 Charges: 4500
```

For 4

Here we added a complaint and displayed it.

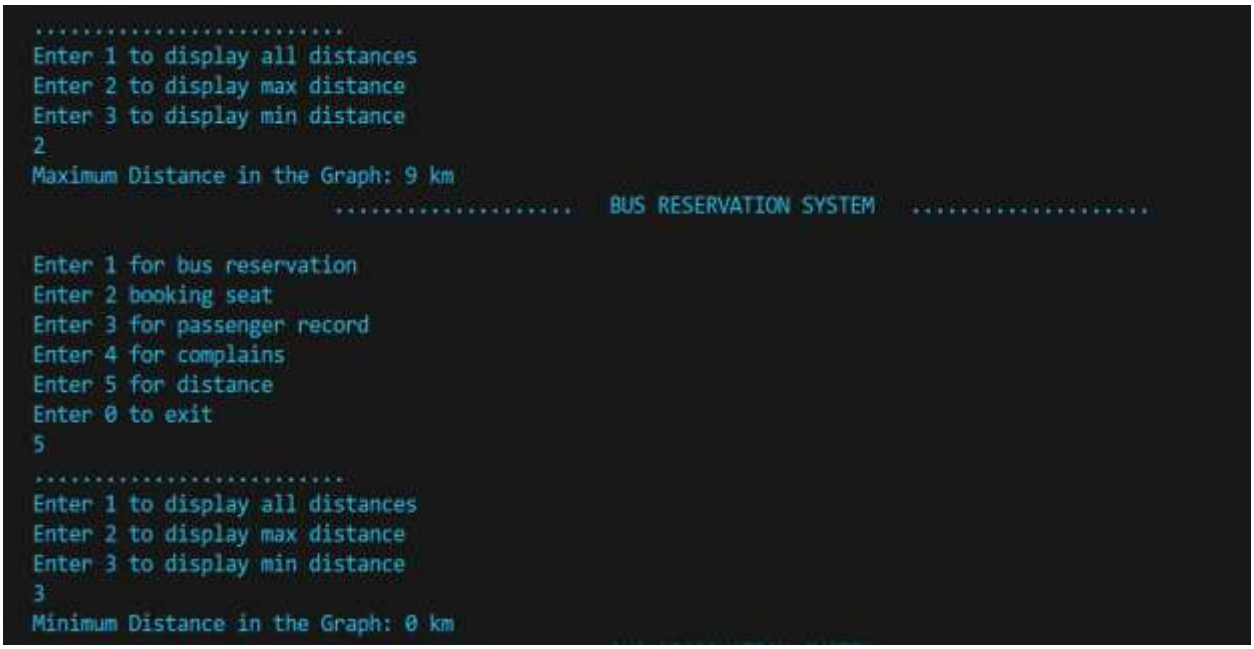
```
.....................BUS SERIAL NUMBERS.....................
13
Enter serial number of the bus
13
Enter complain: luggage missing
Complaint Submitted
                    .....................  BUS RESERVATION SYSTEM  .....................

Enter 1 for bus reservation
Enter 2 booking seat
Enter 3 for passenger record
Enter 4 for complains
Enter 5 for distance
Enter 0 to exit
4
.........................
Enter 1 to regester complain
Enter 2 to delete 1st complain
Enter 3 to view all regestered complains
3
.....................BUS SERIAL NUMBERS.....................
13
Enter serial number of the bus
13
COMPLAINS
...................
Complaints for Bus Serial Number 13:
Complaint: luggage missing
```

For 5

Here we displayed distances of all cities in which we provide services

```
Enter 0 to exit
5
.........................
Enter 1 to display all distances
Enter 2 to display max distance
Enter 3 to display min distance
1
City Graph:
Islamabad - Islamabad : 0 km
Islamabad - Multan : 9 km
Islamabad - Faislabad : 0 km
Islamabad - Lahore : 0 km
Islamabad - Sadiq Abad : 2 km
Islamabad - Quetta : 0 km
Islamabad - Karachi : 1 km
Multan - Islamabad : 9 km
Multan - Multan : 0 km
Multan - Faislabad : 5 km
Multan - Lahore : 1 km
Multan - Sadiq Abad : 2 km
Multan - Quetta : 4 km
Multan - Karachi : 6 km
Faislabad - Islamabad : 7 km
Faislabad - Multan : 5 km
Faislabad - Faislabad : 0 km
Faislabad - Lahore : 1 km
Faislabad - Sadiq Abad : 2 km
Faislabad - Quetta : 5 km
Faislabad - Karachi : 0 km
```

Here we displayed maximum and minimum distance.



This Bus Reservation System enhances efficiency and organization in managing bus-related operations, offering a comprehensive solution for both administrators and passengers.