

***SSN College of Engineering, Kalavakkam***  
***Department of Computer Science and Engineering***

***III Semester - CSE 'B'***

***UCS 1312 Data Structures Lab Laboratory***

***Academic Year: 2019-2020***

***Batch: 2018-2022***

***Mini Project – Hotel Management System***

=====

***Mahesh Bharadwaj - 185001089***

***Niel Parekh - 185001100***

***Sai Charan B – 185001131***

***Abstract:***

In this project we aim at creating a user friendly and efficient hotel management system.

We have used concepts in C programming like file handling, AVL trees and Linked Lists to achieve this.

The system gives a user 2 options of logging in as an administrator or a customer. The administrator is required to enter a unique username and password to gain access to some parts of the software.

There are various functions being simulated like checking in, checking out, generating a bill, getting the history of a room to get all occupants on a given date and also to get details of all past occupants of a given room. These functions are for the administrator's perusal.

There is also a mechanism to verify the dates that are being entered by a customer in order to ensure validity.

Overall this is a very easy to use and friendly system.

***Methodology:***

As mentioned above, concepts like **file-handling**, **AVL trees** and **Priority Queue** have been used to create this system.

The details of all rooms are stored in 'Rooms.dat'.

History of rooms ie the past occupants are stored in RoomHistory.dat.

Customer details are stored in Customer.dat

Login details are stored in login.bin.

Priority Queue is used to handle reservations when all rooms are occupied. Higher priority is given to user with more previous visits even though they might not have requested for a reservation first.

We have implemented an AVL tree to store the rooms in order to make the process of traversing it to find the records of a room much quicker and more efficient.

All the data bring entered into the system is being stored in binary files making it secure and easy to manipulate. The login details of the administrator are also stored in a binary file.

The process is as follows:

Initially, all rooms are read into AVL Tree.

User enters his details -> The date checking mechanism checks if the dates entered make sense -> The details are stored in a file -> The room number is booked in the AVL tree -> The admin logs in -> The bill is generated -> The customer is checked out and the room is made available in the tree -> The details of the customer and room are stored in RoomHistory.dat.

Finally, all room records are written from AVL Tree into file to save current status.

We have also incorporated the concept of ADT in this project.

## ***Functions:***

**The Login Menu :** The user is given the option to login as admin or as customer. Each of which have different options available to them. The admin login requires a username and password. This was implemented using binary file, "login.bin".

**Check in :** If any rooms are vacant, the user is given the option to occupy the room. The user is prompted for their info and the information of their stay. This is implemented using AVL Tree. The room is accessed using AVL Tree.

```
void CheckIn(PriorityQueue P, AVLTree t)
```

**Check out :** It informs the customer to contact the admin to check out.

**Check available rooms/List of Available Rooms :** It lists the rooms that are vacant currently to the customer.

```
void DisplayAvailableRooms(int * arr)
```

**Bill Generator :** This is used by the admin to check out any customer and generate the bill amount that the customer owes. This will vacate the room and make it available for occupation. The information after checking out is stored in a binary file. The checking out process is done using AVL Tree.

```
float CheckOut(AVLTree t)
```

**History Of A Room :** Displays all the previous tenants who stayed in the room specified. This information is stored in a binary file, "RoomHistory.dat". This file is searched and relevant details displayed.

```
void Room_History()
```

**Rooms to Vacate Today :** Displays the rooms and the customers who are expected to check out today. This information is stored in a binary file.

```
void getCheckOut()
```

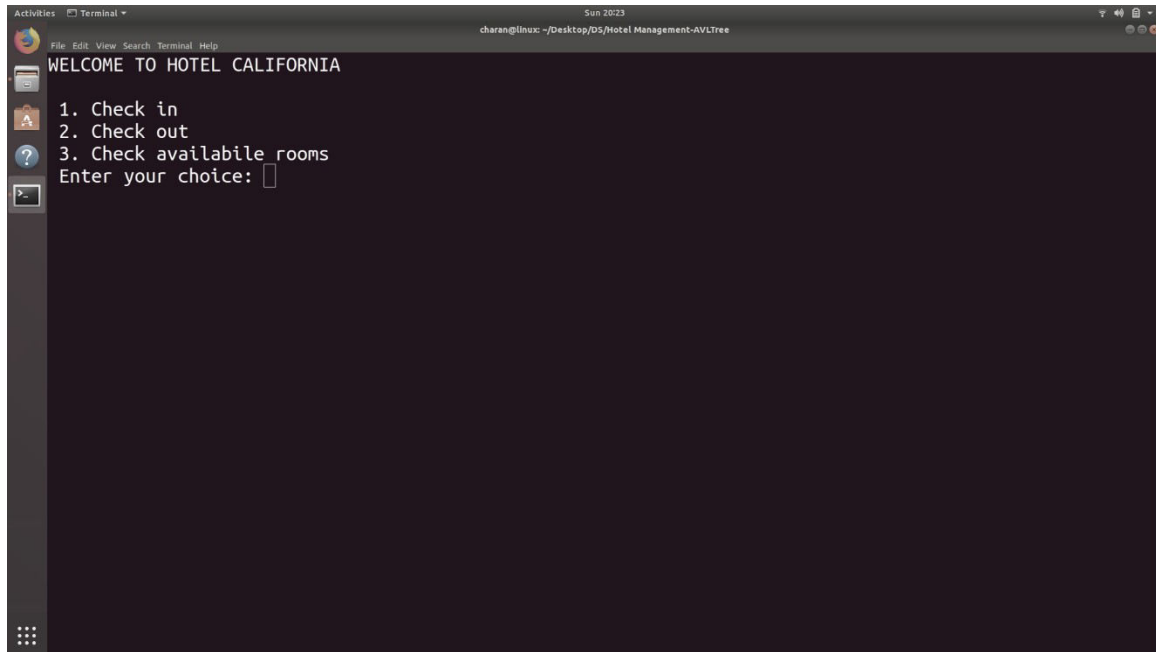
**Date Wise History :** Displays the rooms and the customers who were resident on a given date. This is done by comparing the check in and check out dates of the customer with the requested date. If the date falls within the period, this record is displayed. This information is stored in a binary file 'RoomHistory.dat'. It is accessed by the function

```
void Date_History()
```

**Reservations :** If the rooms are full, the customer is given the option to make a reservation. The customer is added to a priority queue. When a room is vacant, the customer with the highest priority is assigned the room. This is done using priority queue and accessing the AVL Tree.

```
void handleReservations(PQueue Q, AVLTree t)
```

## Output:

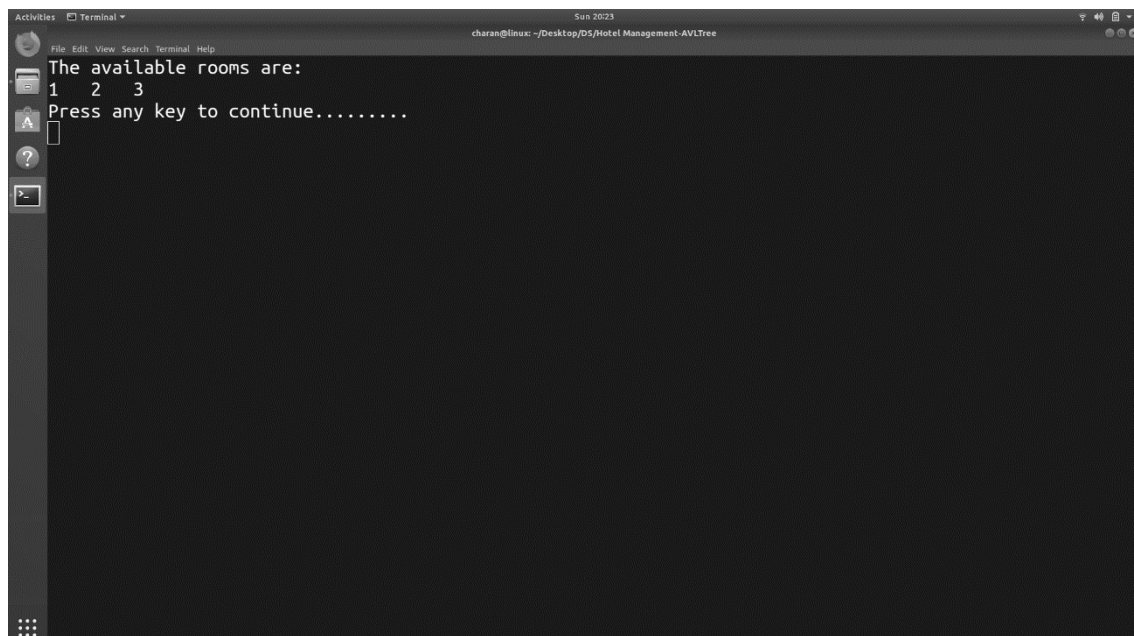


```
Activities Terminal
Sun 2023
charan@linux: ~/Desktop/DS/Hotel Management-AVLTree
File Edit View Search Terminal Help
WELCOME TO HOTEL CALIFORNIA
1. Check in
2. Check out
3. Check available rooms
Enter your choice: 
```

A terminal window titled "Terminal" showing the output of a program. The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal output displays "WELCOME TO HOTEL CALIFORNIA" followed by a numbered list of three options: "1. Check in", "2. Check out", and "3. Check available rooms". Below the list, it prompts the user with "Enter your choice:" followed by a cursor.

1.The Main Menu

2. Viewing all available rooms



```
Activities Terminal
Sun 2023
charan@linux: ~/Desktop/DS/Hotel Management-AVLTree
File Edit View Search Terminal Help
The available rooms are:
1 2 3
Press any key to continue.....

```

A terminal window titled "Terminal" showing the output of a program. The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal output displays "The available rooms are:" followed by the numbers "1 2 3" on the next line. Below that, it prompts the user with "Press any key to continue....." followed by a cursor.

```
Activities Terminal Sun 2024
charan@linux: ~/Desktop/Hotel Management-AVLTree

Existing customer? 1/0 1
Enter the customer ID: 1
ID : 1
Name : Mahesh
Phone Number : 9962243082
Visit Number : 2

Room Number : 1
Check In date : 11/10/2019
Check Out date : 12/10/2019

CUST_ID: 1
-----

Confirm details 1/0 1
The available rooms are:
1 2 3
Enter a rno from the above list: 2
Enter the Check In date[dd/mm/yyyy]: 13/10/2019
Enter the Check Out date[dd/mm/yyyy]: 14/10/2019
Successfully booked room!

Press any key to continue.....
█
```

### 3. Booking A Room

### 4.Admin Login

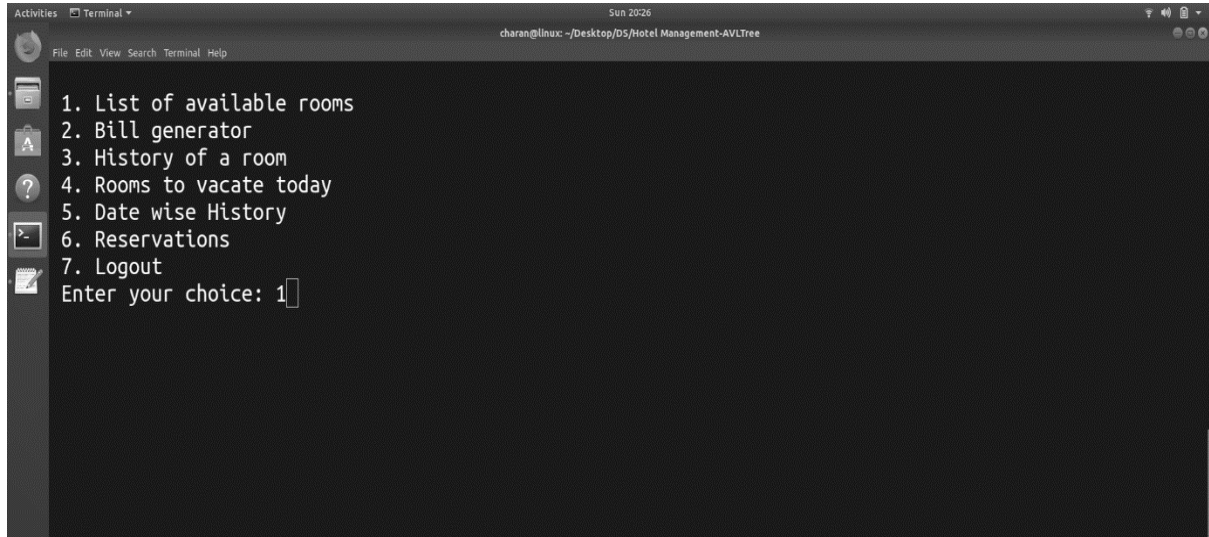
```
Activities Terminal Sun 2026
charan@linux: ~/Desktop/D5/Hotel Management-AVLTree

1. Administrator
2. Customer
3. Exit
Enter your choice: 1
Enter the username : admin
Enter the password:

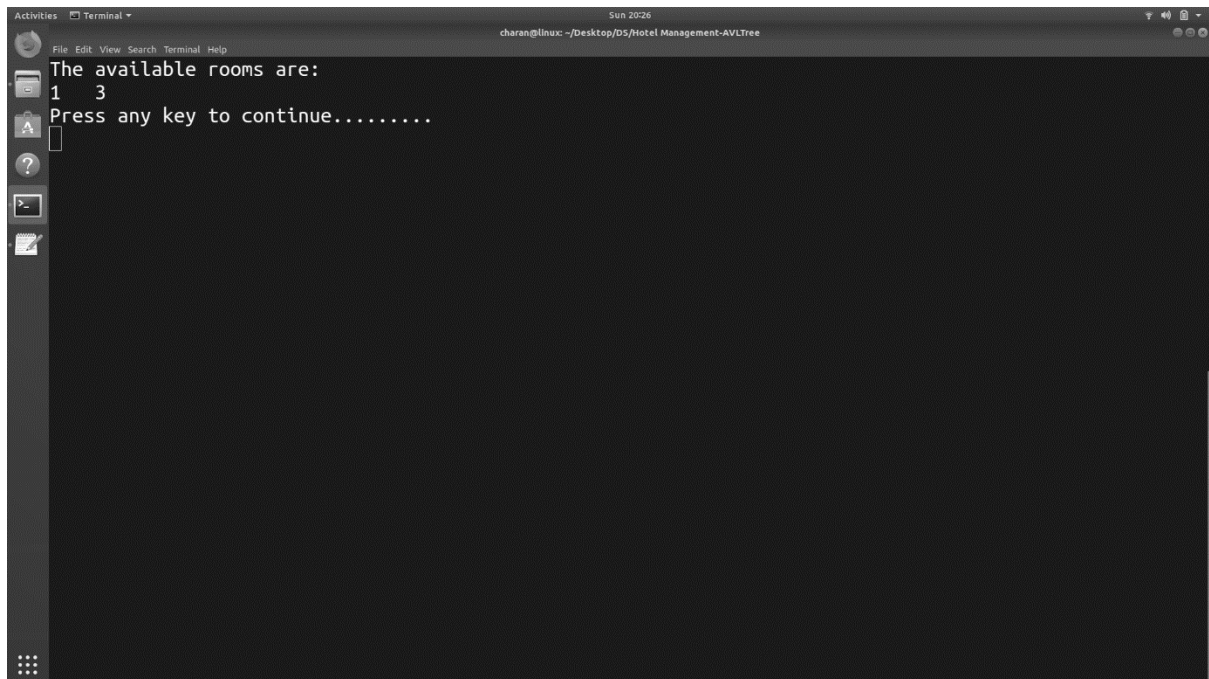
Access Granted

Press any key to continue.....
█
```

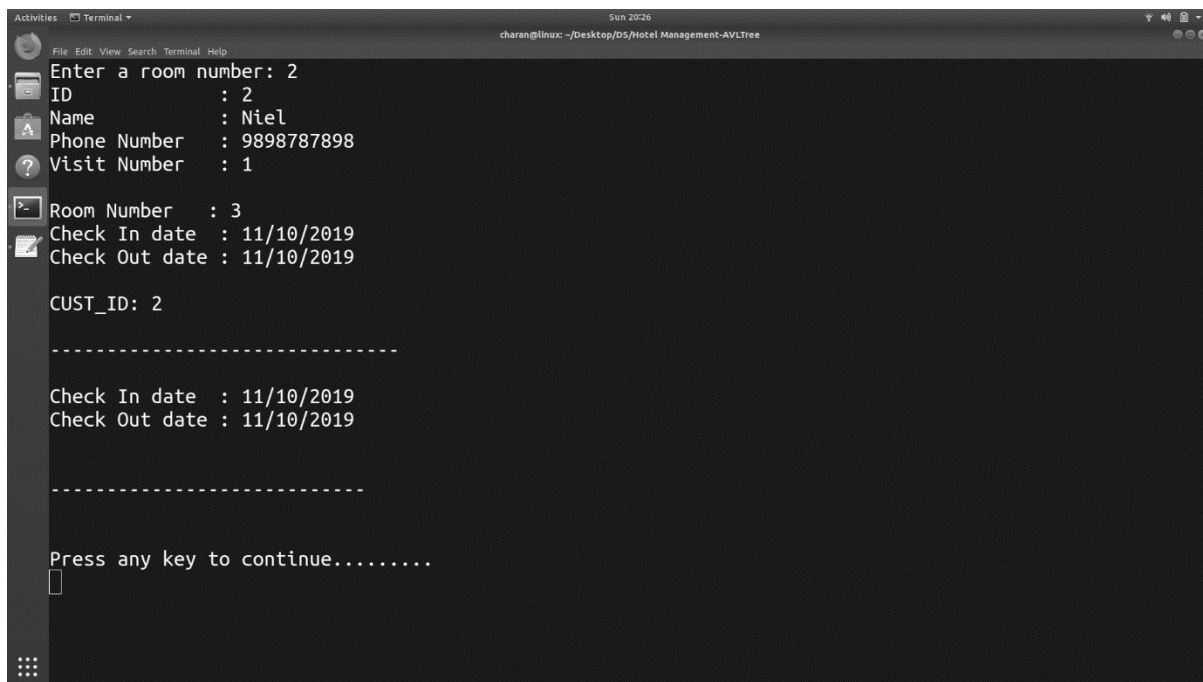
## 5. Admin Menu Screen



## 6. Viewing All available Rooms



## 7. Viewing the History of a Room



A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar (Sun 20:26, charan@linux: ~/Desktop/D5/Hotel Management-AVLTree). The terminal displays the following text:

```
Enter a room number: 2
ID       : 2
Name     : Niel
Phone Number : 9898787898
Visit Number : 1

Room Number : 3
Check In date : 11/10/2019
Check Out date : 11/10/2019

CUST_ID: 2

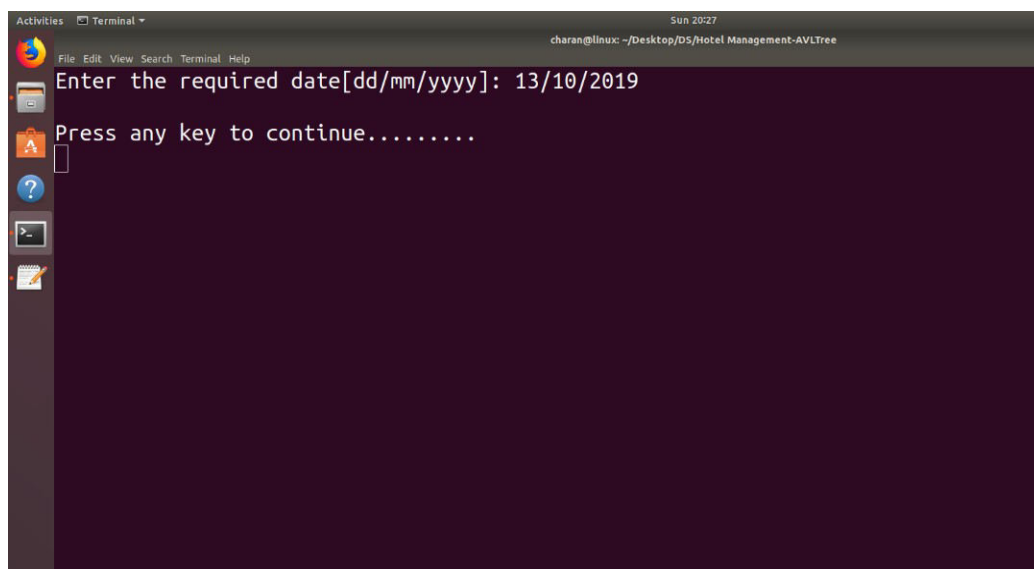
-----

Check In date : 11/10/2019
Check Out date : 11/10/2019

-----

Press any key to continue.....
█
```

## 8. Viewing History based on given Date



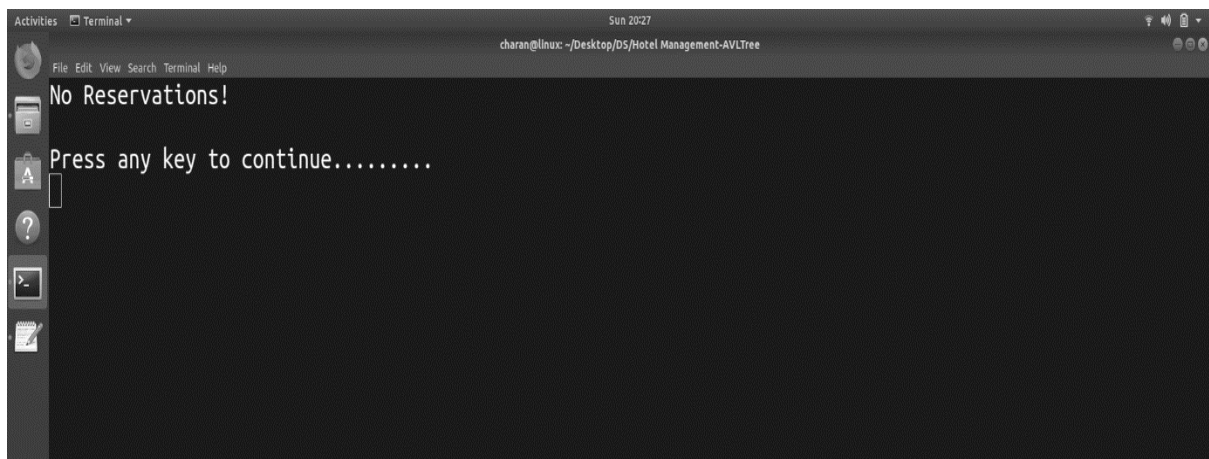
A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar (Sun 20:27, charan@linux: ~/Desktop/D5/Hotel Management-AVLTree). The terminal displays the following text:

```
Enter the required date[dd/mm/yyyy]: 13/10/2019

Press any key to continue.....
█
```



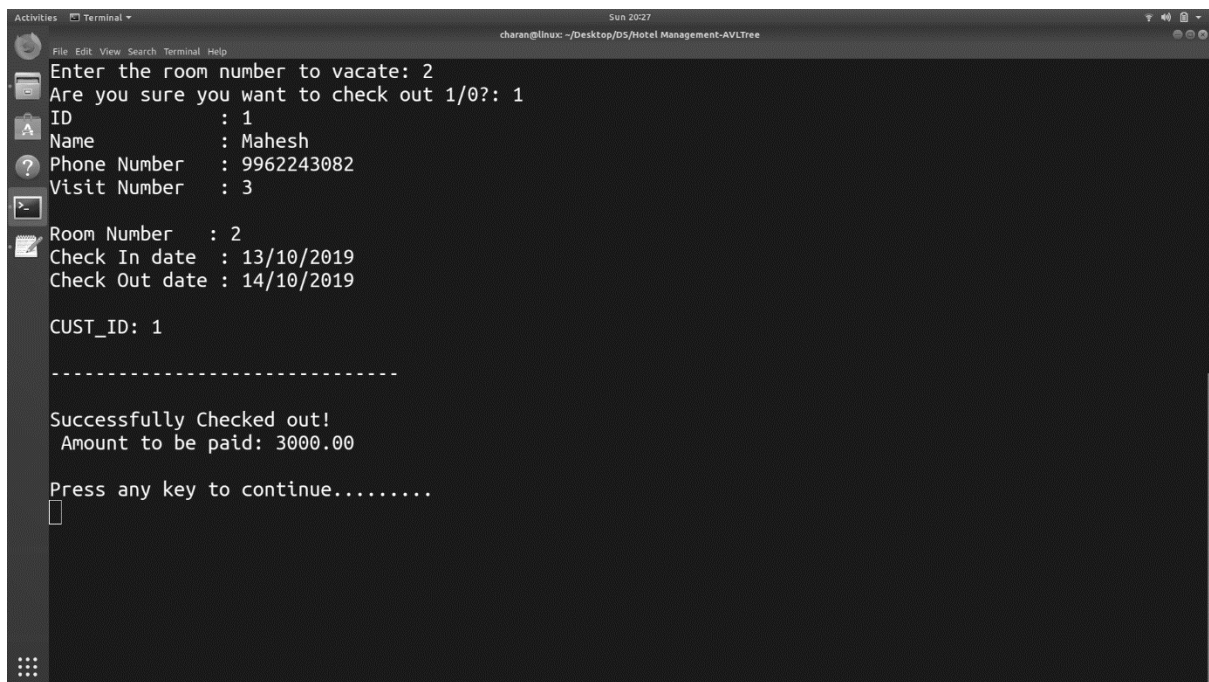
## 9. Viewing Customers who have reservation(Empty at the moment)



A terminal window titled 'Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar (Sun 20:27, charan@linux: ~/Desktop/DS/Hotel Management-AVLTree). The terminal displays the message 'No Reservations!' followed by 'Press any key to continue.....' and a cursor.

```
File Edit View Search Terminal Help
No Reservations!
Press any key to continue.....
█
```

## 10. Checking Out and Generating Bill of a Customer



A terminal window titled 'Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar (Sun 20:27, charan@linux: ~/Desktop/DS/Hotel Management-AVLTree). The terminal displays the following text: 'Enter the room number to vacate: 2', 'Are you sure you want to check out 1/0?: 1', a list of customer details (ID: 1, Name: Mahesh, Phone Number: 9962243082, Visit Number: 3), room details (Room Number: 2, Check In date: 13/10/2019, Check Out date: 14/10/2019), 'CUST\_ID: 1', a separator line, 'Successfully Checked out!', 'Amount to be paid: 3000.00', and 'Press any key to continue.....' with a cursor.

```
File Edit View Search Terminal Help
Enter the room number to vacate: 2
Are you sure you want to check out 1/0?: 1
ID          : 1
Name        : Mahesh
Phone Number : 9962243082
Visit Number : 3

Room Number : 2
Check In date : 13/10/2019
Check Out date : 14/10/2019

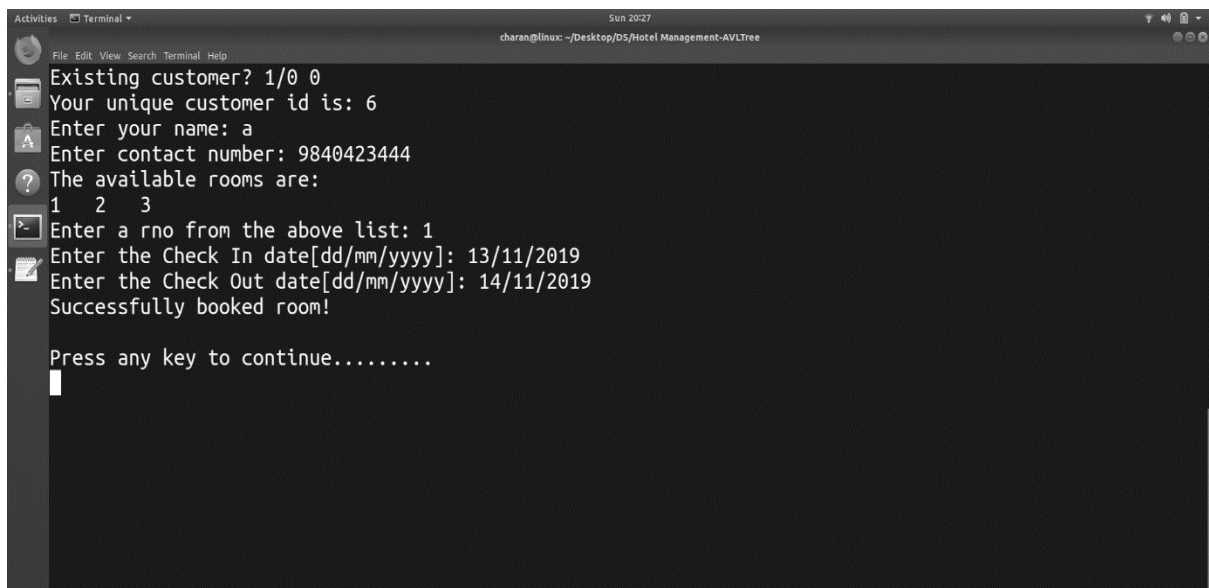
CUST_ID: 1
-----

Successfully Checked out!
Amount to be paid: 3000.00

Press any key to continue.....
█
```



## 11. New Customer booking a Room

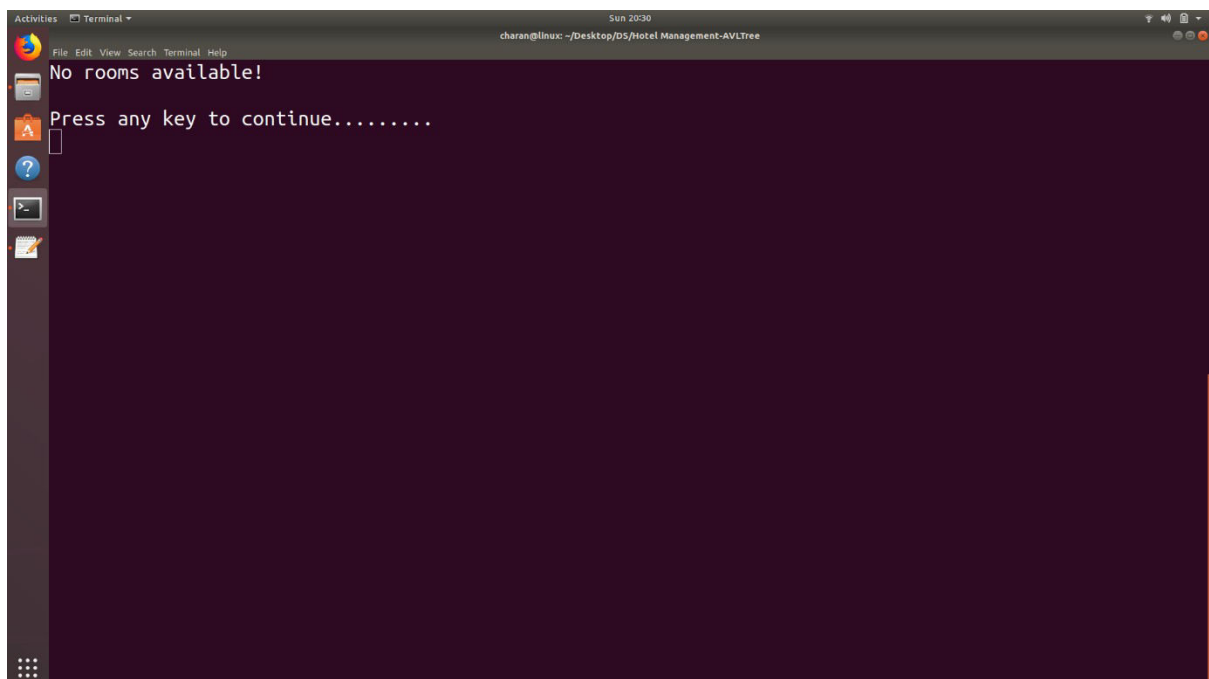


A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar (Sun 20:27, charan@linux: ~/Desktop/D5/Hotel Management-AVLTree). The terminal displays the following text:

```
Existing customer? 1/0 0
Your unique customer id is: 6
Enter your name: a
Enter contact number: 9840423444
The available rooms are:
1 2 3
Enter a rno from the above list: 1
Enter the Check In date[dd/mm/yyyy]: 13/11/2019
Enter the Check Out date[dd/mm/yyyy]: 14/11/2019
Successfully booked room!

Press any key to continue.....
█
```

## 12. After All rooms are occupied:

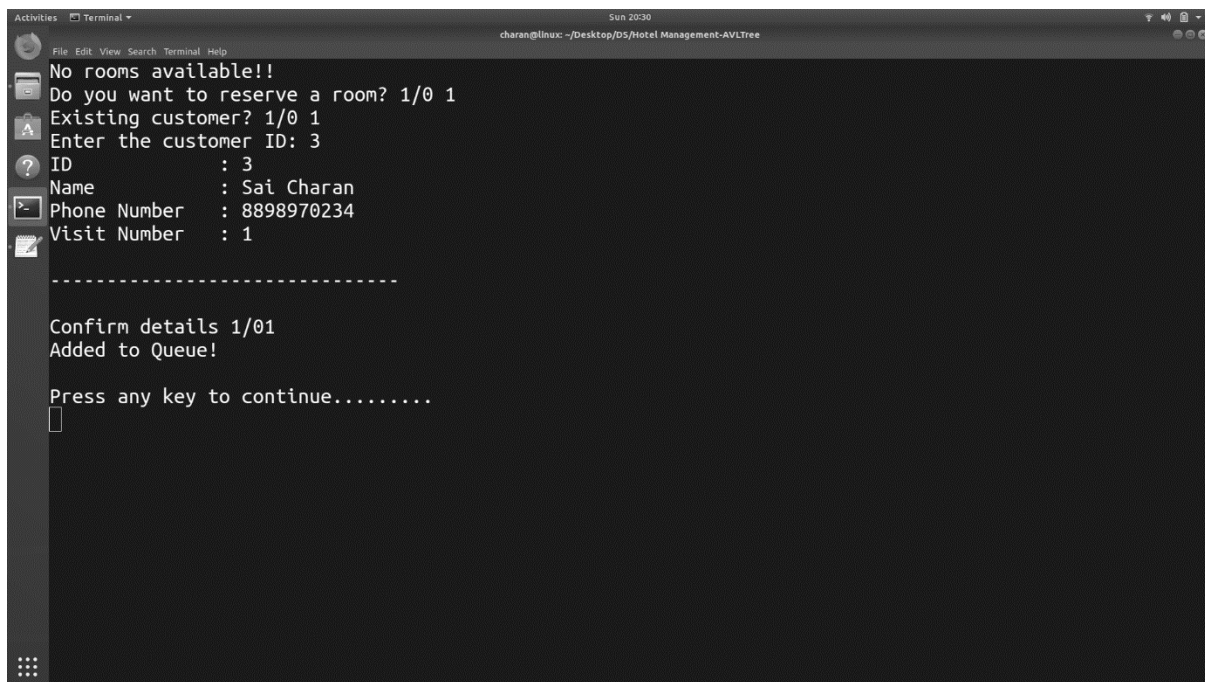


A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar (Sun 20:30, charan@linux: ~/Desktop/D5/Hotel Management-AVLTree). The terminal displays the following text:

```
No rooms available!

Press any key to continue.....
█
```

### 13. Customer Reserving a room

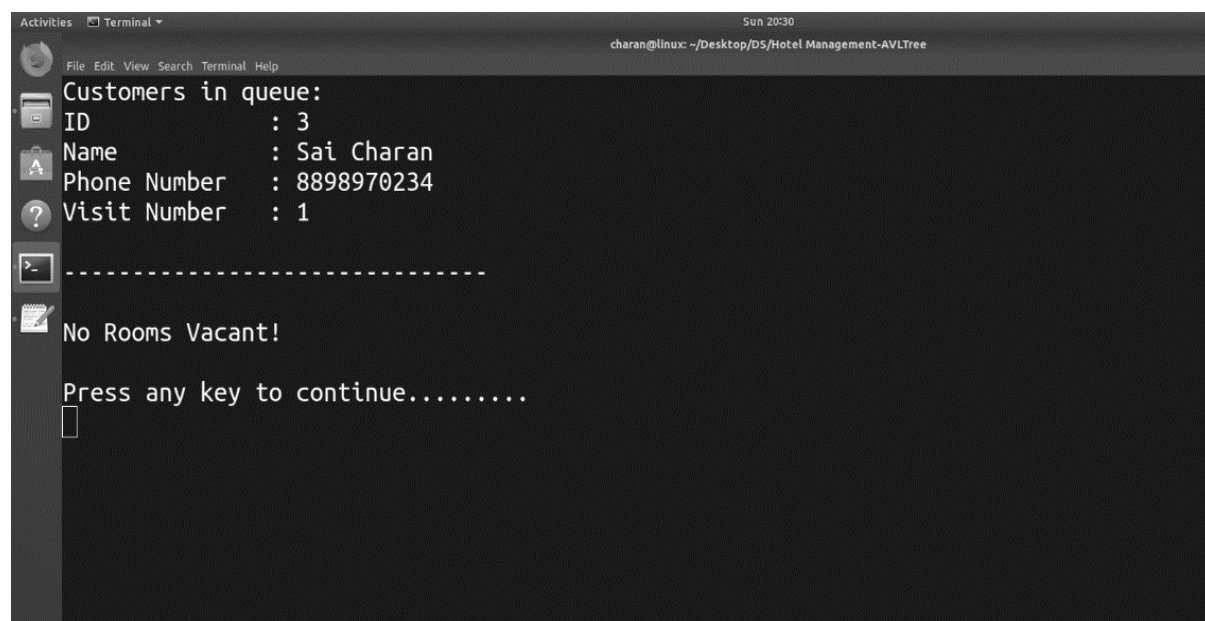


A terminal window titled "Terminal" showing a program execution. The program outputs "No rooms available!!" and asks "Do you want to reserve a room? 1/0 1". The user inputs "1". The program then asks "Existing customer? 1/0 1" and the user inputs "1". It prompts "Enter the customer ID: 3" and the user inputs "3". The program displays the entered details: "Name : Sai Charan", "Phone Number : 8898970234", and "Visit Number : 1". A dashed line separates this from the next prompt: "Confirm details 1/01". The user inputs "1". The program outputs "Added to Queue!" and "Press any key to continue.....". A cursor is visible on the line following the prompt.

```
Activities Terminal
Sun 20:30
charan@linux: ~/Desktop/DS/Hotel Management-AVLTree

No rooms available!!
Do you want to reserve a room? 1/0 1
Existing customer? 1/0 1
Enter the customer ID: 3
ID : 3
Name : Sai Charan
Phone Number : 8898970234
Visit Number : 1
-----
Confirm details 1/01
Added to Queue!
Press any key to continue.....
█
```

### 14. The Queue after previous input:



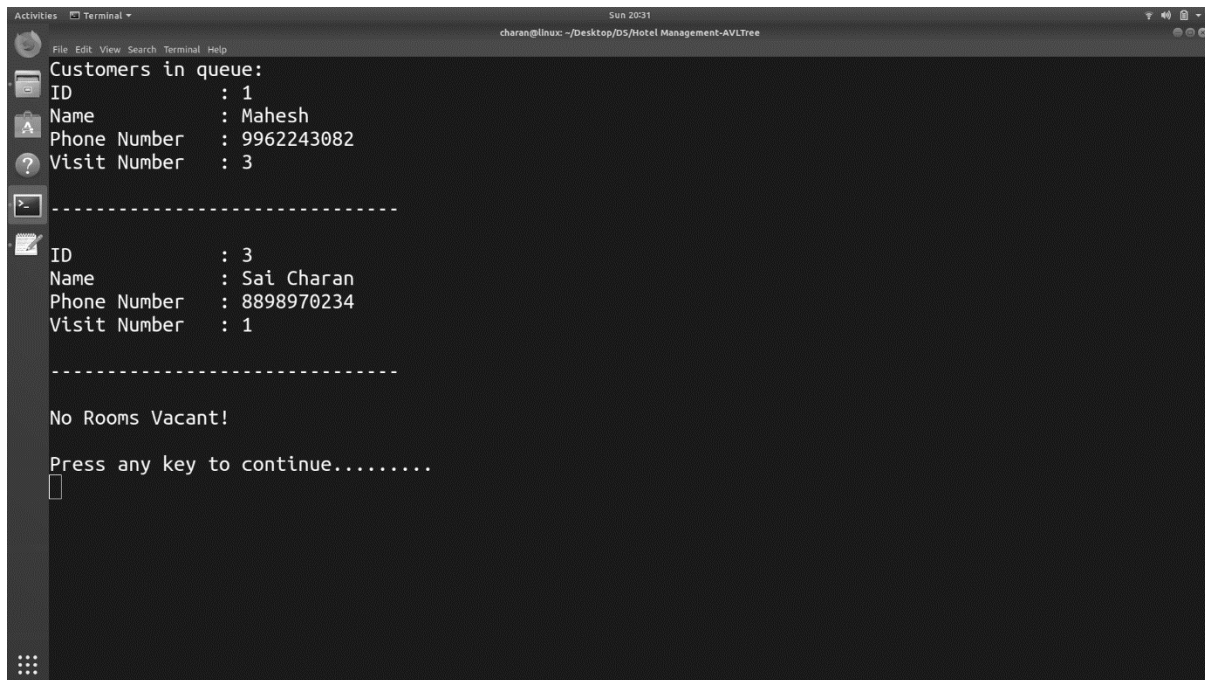
A terminal window titled "Terminal" showing the state of the queue. It outputs "Customers in queue:" followed by the details of the customer in the queue: "ID : 3", "Name : Sai Charan", "Phone Number : 8898970234", and "Visit Number : 1". A dashed line separates this from the next prompt: "No Rooms Vacant!". The program then outputs "Press any key to continue.....". A cursor is visible on the line following the prompt.

```
Activities Terminal
Sun 20:30
charan@linux: ~/Desktop/DS/Hotel Management-AVLTree

Customers in queue:
ID : 3
Name : Sai Charan
Phone Number : 8898970234
Visit Number : 1
-----
No Rooms Vacant!
Press any key to continue.....
█
```

15. After another user (Mahesh) joins queue.

*NOTE the higher precedence given to Mahesh due to higher visit no*



A terminal window titled "Terminal" showing the output of a program. The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal output is as follows:

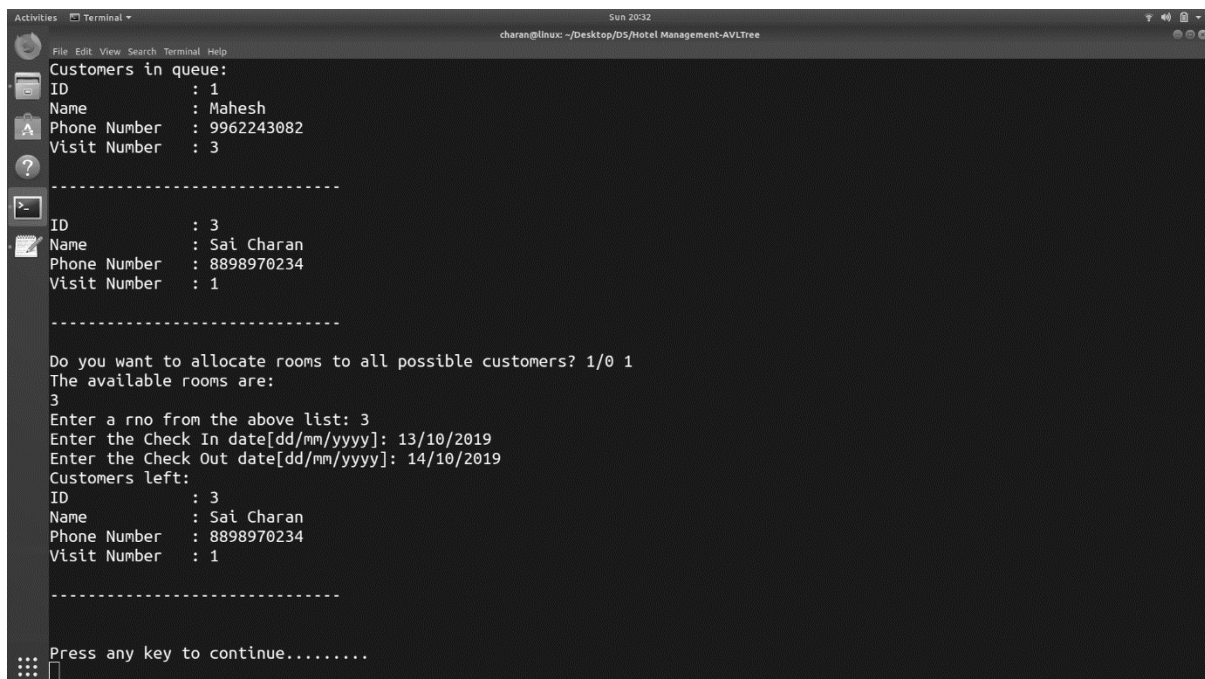
```
Customers in queue:
ID      : 1
Name    : Mahesh
Phone Number : 9962243082
Visit Number : 3
-----
ID      : 3
Name    : Sai Charan
Phone Number : 8898970234
Visit Number : 1
-----

No Rooms Vacant!

Press any key to continue.....

```

16. After a Room is vacated, the reservations are handled



A terminal window titled "Terminal" showing the output of a program. The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal output is as follows:

```
Customers in queue:
ID      : 1
Name    : Mahesh
Phone Number : 9962243082
Visit Number : 3
-----
ID      : 3
Name    : Sai Charan
Phone Number : 8898970234
Visit Number : 1
-----

Do you want to allocate rooms to all possible customers? 1/0 1
The available rooms are:
3
Enter a rno from the above list: 3
Enter the Check In date[dd/mm/yyyy]: 13/10/2019
Enter the Check Out date[dd/mm/yyyy]: 14/10/2019
Customers left:
ID      : 3
Name    : Sai Charan
Phone Number : 8898970234
Visit Number : 1
-----

Press any key to continue.....

```

***Code:***

**#Structures.h**

```
typedef struct{  
    int m,d,y;  
} Date;
```

```
typedef struct{  
    int rno;  
    Date Check_In , Check_Out;  
    unsigned isOccupied : 1;  
    unsigned int cust_id;
```

```
} Room;
```

```
typedef struct{  
    unsigned int id;  
    unsigned int visit_no;  
    char name[30];  
    char ph[11];  
    Room r;  
} Customer;
```

```
typedef Room Record;
```

```
typedef struct avlnode{  
    Record r;  
    struct avlnode * left , * right;  
    short int height;  
}AVLTreeNode;
```

```
typedef AVLTreeNode * AVLTree;
```

```
=====
```

### **#Class\_Date.h**

```
void CreateDate(Date * const Dptr){  
    Dptr -> m = Dptr -> d = Dptr -> y = -1;  
}  
//Constructor equivalent
```

```
int datecmp(const Date d1,const Date d2){           //Returns 1 if d1 is after
0 if same -1 if before
```

```
    if(d2.y > d1.y || (d2.y == d1.y && d2.m > d1.m) || (d2.y == d1.y && d2.m ==
d1.m && d2.d > d1.d))
```

```
        return -1;
```

```
    if(d2.y < d1.y || (d2.y == d1.y && d2.m < d1.m) || (d2.y == d1.y && d2.m ==
d1.m && d2.d < d1.d))
```

```
        return 1;
```

```
    return 0;
```

```
}
```

```
void getDate(const char * const str,Date * const Dptr,const Date Current_Date){
```

```
    int date,month,year;
```

```
    Date tmp;
```

```
    do{
```

```
        printf("Enter the %s date[dd/mm/yyyy]: ",str);
```

```
        scanf("%d/%d/%d",&date,&month,&year);
```

```
        getchar();
```

```
        tmp.m = month;
```

```
        tmp.y = year;
```

```
        tmp.d = date;
```

```
        if( date<1 || date>31 || month>12 || month<1 ||
```

```
        ( (month==4 || month==6 || month==7 || month==9 || month==11)
        && date>30 ) ||
```

```
        ( (year%4!=0) && (month==2) && date>28 ) ||
```

```
        ( (year%4==0) && (month==2) && date>29 ) ||
```

```
        datecmp(tmp,Current_Date) == -1)
```

```
        printf("Invalid Date!Please enter again!\n\n");
```

```
    }while( date<1 || date>31 || month>12 || month<1 ||
```

```
        ( (month==4 || month==6 || month==7 || month==9 || month==11)
        && date>30 ) ||
```

```
        ( (year%4!=0) && (month==2) && date>28 ) ||
```

```
        ( (year%4==0) && (month==2) && date>29 ) ||
```

```
        datecmp(tmp,Current_Date) == -1 );
```

```
    Dptr -> d = date;
```

```
    Dptr -> m = month;
```

```
    Dptr -> y = year;
```

```
}
```

```
Date getCurrentDate(){                                //get current date
```

```
    Date tmp;
```

```
    time_t t=time(0);
```

```
    struct tm * now = localtime(&t);
```

```
    tmp.m = now -> tm_mon + 1;
```

```
    tmp.y = now -> tm_year + 1900;
```

```
    tmp.d = now -> tm_mday;
```



```

        return tmp;
    }

void putDate(const char * const str , const Date date){                //Prints
    out the date on screen
        printf("%s : %d/%d/%d\n" , str , date.d , date.m , date.y);
    }

int countLeapYears(Date d){
    int years = d.y;

    if (d.m <= 2)
        years--;

    return years / 4 - years / 100 + years / 400;
}

int date_diff(const Date d1,const Date d2){
    const int monthDays[12] = {31, 28, 31, 30, 31, 30,31, 31, 30, 31, 30, 31};
    long int n1 = d1.y*365 + d1.d;

    // Add days for months passed
    for (int i=0; i<d1.m - 1; i++)
        n1 += monthDays[i];

    // Add a day for every leap year
    n1 += countLeapYears(d1);

```

```

long int n2 = d2.y*365 + d2.d;

//Add days for months passed
for (int i=0; i<d2.m - 1; i++)
    n2 += monthDays[i];

//Add a day for every leap year
n2 += countLeapYears(d2);
    return (n1 - n2);
}

```

## **#Class\_Room.h**

```
#define MAX_ROOMS 3
```

```
extern Record* search(AVLTree t,int rno);
```

```

void CreateRoom(Room * Rptr){
    Rptr->rno = 0;
    CreateDate(&(Rptr->Check_In) );
    CreateDate(&(Rptr->Check_Out) );
}

```

```
    Rptr->isOccupied = 0;
    Rptr->cust_id = 0
}
```

```
void DisplayAvailableRooms(int * arr){

    if(arr == 0){
        printf("No rooms available!\n");
        return;
    }

    printf("The available rooms are: \n");
    for(int i = 0 ; i < MAX_ROOMS && arr[i] ; i++){
        printf("%-3d ",arr[i]);
        if( (i+1)%10 == 0)
            printf("\n");
    }

}
```

```
int * const getRoomNo(AVLTree t){
    static int arr[MAX_ROOMS];
    for(int i = 0 ; i < MAX_ROOMS ; i++)
        arr[i] = 0;
    int c = -1;
```

```

Room * tmp = 0;
for(int i = 1 ; i <= MAX_ROOMS ; i++){
    tmp = search(t,i);
    if(tmp)                //Room exists
        if(tmp->isOccupied == 0) //Room available
            arr[++c] = tmp->rno;
}
if(c != -1) //Some rooms available
    return arr;
return 0;
}

```

```

Room* Book_Room(AVLTree t){
    Room * tmp = 0;

    const Date Current_Date = getCurrentDate();

    int rno , found = 0;

    int * available_rno = getRoomNo(t);

    if(available_rno == 0)
        return tmp;

    DisplayAvailableRooms(available_rno);

    printf("\n");
}

```

```

do{

    printf("Enter a rno from the above list: ");

    scanf("%d",&rno);

    getchar();


    for(int i = 0 ; i < MAX_ROOMS && available_rno[i]; i++)

        if(rno == available_rno[i]){

            found = 1;

            break;

        }

    if(!found)

        printf("Enter another room number!\n");


}while(!found);


tmp = search(t,rno);


tmp -> isOccupied = 1;


do{

    getDate("Check In" , &(tmp -> Check_In) , Current_Date);

    getDate("Check Out", &(tmp -> Check_Out) , Current_Date);


    if(datecmp( (tmp -> Check_In),(tmp -> Check_Out) ) == 1 ||

        datecmp( (tmp -> Check_In),Current_Date ) == -1)

        printf("Invalid Check In Check Out Dates!Try again!\n\n");

```

```

        }while(datecmp( (tmp -> Check_In),(tmp -> Check_Out) ) == 1 ||
                datecmp( (tmp -> Check_In),Current_Date ) == -1);

    return tmp;
}

```

=====

### **#Class\_Customer.h**

```
#define RATE 3000.0
```

```

void putCustomer(const Customer C){
    printf("ID          : %d\n",C.id);
    printf("Name          : %s\n",C.name);
    printf("Phone Number   : %s\n",C.ph);
    printf("Visit Number   : %d\n",C.visit_no);

    if(C.r.rno != 0){
        printf("\nRoom Number   : %d\n",C.r.rno);
        putDate("Check In date ",C.r.Check_In);
        putDate("Check Out date",C.r.Check_Out);
        printf("\nCUST_ID: %d\n",C.r.cust_id);
    }

    printf("\n-----\n\n");
}

```

```

void CreateCustomer(Customer * const Cptr){//Constructor equivalent
    (*Cptr).name[0] = '\0';
    (*Cptr).ph[0] = '\0';
    Cptr -> id = 0;
    Cptr -> visit_no = 0;
    CreateRoom( &(Cptr->r) );
}
//float CheckOut(){

```

```

float CheckOut(AVLTree t){
    const Date CurrentDate = getCurrentDate();
    //FILE * fp = fopen("Rooms.dat","rb+");
    //Room *tmp = (Room*)malloc(sizeof(Room));
    Room * tmp;
    Customer *temp = (Customer*)malloc(sizeof(Customer));
    int rno,opt;

    do{
        do{
            printf("Enter the room number to vacate: ");
            scanf("%d",&rno);
            getchar();

            if(rno < 1 || rno > MAX_ROOMS)
                printf("Invalid Input!Try again!\n");

        }while(rno<1 || rno>MAX_ROOMS);
    }
}

```



```
tmp = search(t,rno);
```

```
if( (!tmp -> isOccupied) || (!tmp))
```

```
    printf("Invalid Room Number!Try again!\n");
```

```
}while(!tmp -> isOccupied);
```

```
const int cust_id = tmp -> cust_id;
```

```
do{
```

```
    printf("Are you sure you want to check out 1/0?: ");
```

```
    scanf("%d",&opt);
```

```
    getchar();
```

```
    if(!(opt == 1 || opt == 0))
```

```
        printf("Invalid Input!Try again\n");
```

```
}while(!(opt==1 || opt==0));
```

```
if (opt == 0)
```

```
    return -1;
```

```
CreateRoom(tmp);
```

```
tmp->rno = rno;
```

```
FILE * fp = fopen("Customer.dat","rb+");
```

```
fseek(fp,(cust_id-1)*sizeof(Customer),0);
```

```
fread(temp,1,sizeof(Customer),fp);
```

```

temp -> visit_no++;
putCustomer(*temp);
Room r = temp -> r;
CreateRoom(&(temp -> r));
fseek(fp,(cust_id-1)*sizeof(Customer),0);

fwrite(temp,1,sizeof(Customer),fp);
fclose(fp);

fp = fopen("RoomHistory.dat","ab");
int date_diff1 = date_diff(r.Check_Out,r.Check_In);
int date_diff2 = date_diff(CurrentDate,r.Check_In);

if(date_diff2 < date_diff1)
    r.Check_Out = CurrentDate;

fwrite(&r,1,sizeof(Room),fp);

fclose(fp);

int date_diff = (date_diff1>date_diff2)? date_diff2 : date_diff1;
date_diff++;

free(temp);
printf("Successfully Checked out!");
return date_diff * RATE;
}

```

```

void Room_History(){
    int rno;
    int cust_id;

    do{
        printf("Enter a room number: ");
        scanf("%d",&rno);
        getchar();

        if(rno<1 || rno>MAX_ROOMS)
            printf("Invalid Input!Try again!\n");

    }while(rno<1 || rno>100);

    FILE *f1 = fopen("RoomHistory.dat","rb"),
        *f2 = fopen("Customer.dat","rb");

    Customer *tmp = (Customer*)malloc(sizeof(Customer));
    Room * r = (Room*)malloc(sizeof(Room));

    fread(r,1,sizeof(Room),f1);
    while(!feof(f1)){
        if(r -> rno == rno){
            cust_id = r -> cust_id;
            fseek(f2,(cust_id-1)*sizeof(Customer),0);

```

```

        fread(tmp,1,sizeof(Customer),f2);
        putCustomer( (*tmp) );
        putDate("Check In date ",r -> Check_In);
        putDate("Check Out date",r -> Check_Out);
        printf("\n\n-----\n\n");
    }
    fread(r,1,sizeof(Room),f1);
}

fclose(f1);
fclose(f2);
free(r);
free(tmp);
}

```

```

void Date_History(){
    Date tmp,d;
    int cust_id;

    CreateDate(&tmp);
    CreateDate(&d);
    getDate("required",&d,tmp);

    FILE * f1 = fopen("Customer.dat","rb"),
        * f2 = fopen("RoomHistory.dat","rb");

    Customer *c = (Customer*)malloc(sizeof(Customer));
    Room *r = (Room*)malloc(sizeof(Room));
}

```

```

fread(r,1,sizeof(Room),f2);
while(!feof(f2)){
    if( (date_diff(r -> Check_In,d) <= 0) && (date_diff(r -> Check_Out,d) >=
0) ){

        cust_id = r -> cust_id;
        fseek(f1,(cust_id-1)*sizeof(Customer),0);
        fread(c,1,sizeof(Customer),f1);
        putCustomer( (*c) );
        putDate("Check In date ",r -> Check_In);
        putDate("Check Out date",r -> Check_Out);
        printf("\n\n-----\n\n");
    }
    fread(r,1,sizeof(Room),f2);
}

fclose(f1);
fclose(f2);
free(r);
free(c);
}

```

=====

## #Class\_login.h

```

struct login{
    char username[10];

```

```

    char pass[25];
    int verified;
};

void getCheckOut(){

    FILE * fp = fopen("Rooms.dat","rb");
    FILE * f = fopen("Customer.dat","rb");

    Room * tmp = (Room*)malloc(sizeof(Room));
    Customer * temp = (Customer*)malloc(sizeof(Customer));

    fread(tmp,1,sizeof(Room),fp);

    while(!feof(fp) ){
        if(tmp->isOccupied && datecmp(tmp->Check_Out,getCurrentDate()) ==
0){

            printf("Room Number: %d\n",tmp -> rno);
            fseek(f,(tmp->cust_id - 1)*sizeof(Customer),0);
            fread(temp,1,sizeof(Customer),f);
            putCustomer(*temp);
        }
        fread(tmp,1,sizeof(Room),fp);
    }

    fclose(fp);
    fclose(f);
}

```

```

        free(tmp);
        free(temp);

    }

=====

```

### **#AVLTree.h**

```

int height(AVLTree t){
    if(t == NULL)
        return -1;

    return t -> height;
}

int max(int a,int b){
    return ((a > b)? a : b);
}

AVLTree singlerotatewithleft(AVLTree k2){
    AVLTree k1;
    k1 = k2 -> left;
    k2 -> left = k1 -> right;
    k1 -> right = k2;

    k2 -> height = max(height(k2 -> left),height(k2 -> right)) + 1;
    k1 -> height = max(height(k1 -> left),k2 -> height) + 1;
}

```



```
    return k1;
}
```

```
AVLTree singlerotatewithright(AVLTree k2){
    AVLTree k1;
    k1 = k2 -> right;
    k2 -> right = k1 -> left;
    k1 -> left = k2;

    k2 -> height = max(height(k2 -> left),height(k2 -> right)) + 1;
    k1 -> height = max(height(k1 -> left),k2 -> height) + 1;

    return k1;
}
```

```
AVLTree doublerotatewithleft(AVLTree k3){
    k3 -> left = singlerotatewithright(k3 -> left);

    return singlerotatewithleft(k3);
}
```

```
AVLTree doublerotatewithright(AVLTree k3){
    k3 -> right = singlerotatewithleft(k3 -> right);

    return singlerotatewithright(k3);
}
```

```

AVLTree insert(Record r,AVLTree t){
    if(t == NULL){
        t = (AVLTree)malloc(sizeof(AVLTreeNode));
        t -> r = r;
        t -> left = t -> right = NULL;
    }

    else if(r.rno < t -> r.rno){
        t -> left = insert(r,t -> left);
        if(height(t -> left) - height(t -> right) == 2)
            if(r.rno < t -> left -> r.rno){
                printf("\nSingle rotate performed!\n");
                t = singlerotatewithleft(t);
            }
            else{
                printf("\nDouble rotate performed!\n");
                t = doublerotatewithleft(t);
            }
    }

    else if(r.rno > t -> r.rno){
        t -> right = insert(r,t -> right);
        if(height(t -> right) - height(t -> left) == 2)
            if(r.rno > t -> right -> r.rno){
                printf("\nSingle rotate performed!\n");
                t = singlerotatewithright(t);
            }
            else{

```

```

        printf("\nDouble rotate performed!\n");
        t = doublerotatewithright(t);
    }
}

t -> height = max(height(t -> left),height(t -> right)) + 1;
return t;
}

```

```

Record* search(AVLTree t,int rno){
    if(t == NULL){
        static Room r;
        CreateRoom(&r);
        printf("Not found!\n");
        return &r;
    }

    if(rno < t -> r.rno)
        return search(t -> left,rno);
    else if(rno > t -> r.rno)
        return search(t -> right,rno);
    else
        return &(t -> r);
}

```

```

AVLTree readFile(){
    FILE * f = fopen("Rooms.dat","rb");
    static AVLTree t = NULL;
}

```

```

    Room tmp;
    fread(&tmp,1,sizeof(Room),f);
    while(!feof(f)){
        t = insert(tmp,t);
        fread(&tmp,1,sizeof(Room),f);
    }
    fclose(f);

    return t;
}

```

```

void inorder(AVLTree t){
    if(t == NULL)
        return;
    inorder(t -> left);
    printf("%d\t",t -> r.rno);
    inorder(t -> right);
}

```

```

void writeFile(AVLTree t,FILE * f){
    if(t == NULL)
        return;
    writeFile(t -> left,f);
    fwrite(&(t -> r),1,sizeof(Room),f);
    writeFile(t -> right,f);
}

```

=====

**#Max\_Heap.h**

#define MAX\_LENGTH 20

typedef Customer Data;

typedef struct PriorityQueue{

int capacity;

int size;

Data\* arr;

}PriorityQueue;

typedef PriorityQueue\* PQueue;

int isFull(PQueue Q){

return Q -> size == Q -> capacity;

}

int isEmpty(PQueue Q){

return Q -> size == 0;

}

PQueue createPQueue(){

PQueue tmp = (PQueue)malloc(sizeof(PriorityQueue));

tmp -> capacity = MAX\_LENGTH;

```

tmp -> size = 0;
tmp -> arr = (Data*)malloc(sizeof(Data) * MAX_LENGTH);

tmp -> arr[0].visit_no = 999999;
return tmp;
}

```

```

void enqueue(PQueue q,const Data d){
    if(isFull(q)){
        printf("Queue Full!\n");
        return;
    }

    int i = ++q -> size;
    for(; q -> arr[i/2].visit_no < d.visit_no ; i /= 2)
        q -> arr[i] = q -> arr[i/2];

    q -> arr[i] = d;

}

```

```

Data dequeue(PQueue q){
    if(isEmpty(q)){
        printf("Queue Empty!\n");
        return q -> arr[0];
    }

    int i,child;
    Data min,last;

```

```

min = q -> arr[1];
last = q -> arr[q -> size--];

for(i = 1; i * 2 <= q -> size ; i = child){
    child = i * 2;

    if(child != q -> size && q -> arr[child + 1].visit_no > q ->
arr[child].visit_no)
        child ++;
    if(last.visit_no < q -> arr[child].visit_no)
        q -> arr[i] = q -> arr[child];
    else
        break;
}

q -> arr[i] = last;
return min;
}

```

```

void display(PQueue Q){
    for(int i = 1 ; i <= Q -> size ; i++)
        putCustomer(Q -> arr[i]);
}

```

```

void handleReservations(PQueue Q,AVLTree t){
    int opt;
    if(isEmpty(Q)){

```



```
        printf("No Reservations!\n");
        return;
    }
```

```
printf("Customers in queue: \n");
display(Q);
Customer c;
int * arr = getRoomNo(t);
```

```
if(arr == NULL){
    printf("No Rooms Vacant!\n");
    return;
}
```

```
FILE * f2,* f1 = fopen("Customer.dat","rb+");
```

```
printf("Do you want to allocate rooms to all possible customers? 1/0 ");
scanf("%d",&opt);
getchar();
```

```
//Room tmp;
```

```
Room * tmp = 0;
```

```
if(opt == 1){
    for(int i = 0 ; arr[i] && !isEmpty(Q) ; i++){
        c = dequeue(Q);
        c.r = * ((Room*)(Book_Room(t)));
```

```
        fseek(f1,(c.id - 1) * sizeof(Customer),0);
```

```

        fwrite(&c,1,sizeof(Customer),f1);
    }
    if(!isEmpty(Q)){
        printf("Customers left:\n");
        display(Q);
    }
    else
        printf("All customers allotted rooms!\n");
}

fclose(f1);
}

```

```

void CheckIn(PQueue q,AVLTree t){

    int opt;
    int cust_id;
    Customer *Cptr = (Customer*)malloc(sizeof(Customer));
    CreateCustomer(Cptr);
    FILE * fp = fopen("Customer.dat","rb");

    if(!getRoomNo(t)){
        printf("No rooms available!!\n");
        printf("Do you want to reserve a room? 1/0 ");
        scanf("%d",&opt);
        getchar();
    }
}

```

```

if(opt == 1){
    do{
        printf("Existing customer? 1/0 ");
        scanf("%d",&opt);
        getchar();

        if(opt != 1 && opt != 0)
            printf("Invalid Input!\n");

    }while(opt != 0 && opt != 1);

    if(opt == 1){
        printf("Enter the customer ID: ");
        scanf("%d",&cust_id);
        getchar();

        fseek(fp,(cust_id - 1)*sizeof(Customer),0);
        fread(Cptr,1,sizeof(Customer),fp);
        if(feof(fp)){
            printf("Invalid Input!\n");
            fclose(fp);
            return;
        }
        fclose(fp);

        putCustomer(*Cptr);
        CreateRoom(&(Cptr->r));
    }
}

```

```

do{

    printf("Confirm details 1/0");
    scanf("%d",&opt);
    getchar();

    if(opt != 0 && opt != 1)
        printf("Invalid Input!\n");

}while(opt != 0 && opt != 1);

if(opt == 1){
    enqueue(q,(*Cptr));
    printf("Added to Queue!\n");
}
}

else{
    fseek(fp,0,2);
    const int last_id = ftell(fp)/sizeof(Customer);
    fclose(fp);
    Cptr -> id = last_id + 1;
    Cptr -> visit_no = 0;
    printf("Your unique customer id is: %d\n",Cptr -> id);

    printf("Enter your name: ");
    scanf("%[^\\n]*c",Cptr->name);
    getchar();

    do{

```

```

        printf("Enter contact number: ");
        scanf("%s",Cptr->ph);
        getchar();

        if(strlen(Cptr->ph) != 10)
            printf("Invalid Contact Number!Try again!");

    }while(strlen(Cptr->ph) != 10);

    CreateRoom(&(Cptr -> r));
    fp = fopen("Customer.dat","ab");
    fwrite(Cptr,1,sizeof(Customer),fp);
    fclose(fp);
    enqueue(q,(*Cptr));
    }
}
return;
}

```

```
do{

    printf("Existing customer? 1/0 ");
    scanf("%d",&opt);
    getchar();

    if(opt != 1 && opt != 0)
        printf("Invalid Input!\n");

}while(opt != 0 && opt != 1);

if(opt == 1){
    printf("Enter the customer ID: ");
    scanf("%d",&cust_id);
    getchar();

    fseek(fp,(cust_id - 1)*sizeof(Customer),0);
    fread(Cptr,1,sizeof(Customer),fp);

    if(feof(fp)){
        printf("Invalid Input!\n");
        fclose(fp);
    }
}
```

```

        return;
    }

    fclose(fp);
    putCustomer(*Cptr);

    do{
        printf("Confirm details 1/0 ");
        scanf("%d",&opt);
        getchar();

        if(opt != 0 && opt != 1)
            printf("Invalid Input!\n");

    }while(opt != 0 && opt != 1);

```

```

Room * tmp = Book_Room(t);
tmp -> cust_id = cust_id;
Cptr -> r = (*tmp);

```

```

    fp = fopen("Customer.dat","rb+");
    fseek(fp,(cust_id - 1) * sizeof(Customer),0);
    fwrite(Cptr,1,sizeof(Customer),fp);
    fclose(fp);
}

```

```

else{

```

```
fseek(fp,0,2);
const int last_id = ftell(fp)/sizeof(Customer);
fclose(fp);
Cptr -> id = last_id + 1;
Cptr -> visit_no = 1;
printf("Your unique customer id is: %d\n",Cptr -> id);

printf("Enter your name: ");
scanf("%[^\\n]*c",Cptr->name);
getchar();

do{
    printf("Enter contact number: ");
    scanf("%s",Cptr->ph);
    getchar();

    if(strlen(Cptr->ph) != 10)
        printf("Invalid Contact Number!Try again!");

}while(strlen(Cptr->ph) != 10);

Room * tmp = Book_Room(t);
tmp -> cust_id = Cptr -> id;
Cptr -> r = *tmp;

fp = fopen("Customer.dat","ab");
fwrite(Cptr,1,sizeof(Customer),fp);
fclose(fp);
```



```

    }

    free(Cptr);
    printf("Successfully booked room!\n");

}

```

```
=====
```

### **#Hotel.c**

```

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <unistd.h>

#include "Structures.h"
#include "class_date.h"
#include "class_room.h"
#include "class_customer.h"
#include "class_login.h"
#include "AVLTree.h"
#include "MaxHeap.h"

void Pause(){
    printf("\nPress any key to continue.....\n");
}

```

```

        getchar();
    }

void main(void){
    AVLTree t;
    //inorder(t);

    FILE * f_rooms;
    struct login x = {"", "", 0};
    int choice;
    FILE *id;
    char user[10],password[25];
    PQueue q = createPQueue();

    do{

        t = readFile();

        system("clear");
        printf("\n 1. Administrator\n 2. Customer\n 3. Exit\n Enter your choice:
");

        scanf("%d",&choice);
        getchar();

        if(choice==1){

            if(!x.verified){
                printf(" Enter the username : ");

```

```

scanf("%s",user);
getchar();

strcpy(password,getpass(" Enter the password: "));

id=fopen("id.bin","rb");
fread(&x,1,sizeof(x),id);
fclose(id);

if(strcmp(x.username,user)==0 &&
strcmp(x.pass,password)==0){
    printf("\n Access Granted\n");
    x.verified = 1;
    Pause();
}
else{
    printf("\n Wrong password or username\n");
    Pause();
    continue;
}
}

do{
    system("clear");
    printf("\n 1. List of available rooms\n 2. Bill generator\n
n");

    printf(" 3. History of a room\n 4. Rooms to vacate today\n
n");

```

```

        printf(" 5. Date wise History\n 6. Reservations\n 7.
Logout\n Enter your choice: ");

        scanf("%d",&choice);

        getchar();


        //system("cls");
        system("clear");
        float amount;
        switch(choice){
            case 1 : DisplayAvailableRooms(getRoomNo(t));
Pause(); break;

            case 2 : amount = CheckOut(t);
                if(amount != -1){
                    printf("\n Amount to be paid: %.2f\n
n",amount);

                    f_rooms = fopen("Rooms.dat","wb");
                        writeFile(t,f_rooms);
                        fclose(f_rooms);
                }
Pause(); break;

            case 3 : Room_History();
Pause(); break;

            case 4 : getCheckOut();
Pause(); break;

            case 5 : Date_History();
Pause(); break;

            case 6 : handleReservations(q,t);
                f_rooms = fopen("Rooms.dat","wb");
                writeFile(t,f_rooms);

```

```
fclose(f_rooms);
```

```
Pause(); break;
```

```
case 7 : printf("\n Logged out successfully!\n");
```

```
        x.verified = 0;
```

```
Pause(); break;
```

```
default: printf("\n Invalid Input!Try again!\n");Pause(); break;
```

```
}
```

```
}while(choice!=7);
```

```
choice = -1;
```

```
}
```

```
else if(choice==2){
```

```
    system("clear");
```

```
    printf("WELCOME TO HOTEL CALIFORNIA \n");
```

```
    printf("\n 1. Check in\n 2. Check out\n 3. Check available  
rooms\n Enter your choice: ");
```

```
    scanf("%d",&choice);
```

```
    getchar();
```

```

        system("clear");
        switch(choice){
            case 1 : CheckIn(q,t);

                        f_rooms = fopen("Rooms.dat","wb");
                        writeFile(t,f_rooms);
                        fclose(f_rooms);

            Pause(); break;

            case 2 : printf(" Contact Admin!\n");

Pause(); break;

            case 3 : DisplayAvailableRooms(getRoomNo(t));  Pause();

break;

            default: printf("\n Invalid Input!Try again!\n");Pause();

break;

        }

        choice = -1;
    }

    else if(choice==3){
        printf("\n Exiting.....\n");
    }

    else{
        printf("\n Please enter a valid option\nPress any key to
continue\n");

        getchar();
    }

    /*

f_rooms = fopen("Rooms.dat","wb");
writeFile(t,f_rooms);
fclose(f_rooms);

```

\*/

}while(choice!=3);

}

=====

***Conclusions:***

The system is currently restricted to 3 rooms.

Various data structures and concepts taught to us were implemented to make a working hotel management system.