```
/*
Design a Call taxi booking application
-There are n number of taxi's. For simplicity, assume 4. But it should work for any number of taxi's.
-The are 6 points(A,B,C,D,E,F)
-All the points are in a straight line, and each point is 15kms away from the adjacent points.
-It takes 60 mins to travel from one point to another
-Each taxi charges Rs.100 minimum for the first 5 kilometers and Rs.10 for the subsequent kilometers.
-For simplicity, time can be entered as absolute time. Eg: 9hrs, 15hrs etc.
-All taxi's are initially stationed at A.
-When a customer books a Taxi, a free taxi at that point is allocated
-If no free taxi is available at that point, a free taxi at the nearest point is allocated.
-If two taxi's are free at the same point, one with lower earning is allocated
-Note that the taxi only charges the customer from the pickup point to the drop point. Not the distance
it travels from an adjacent point to pickup the customer.
-If no taxi is free at that time, booking is rejected
Design modules for

1)  Call taxi booking
Input 1:
Customer ID: 1
Pickup Point: A
Drop Point: B
Pickup Time: 9

Output 1:
Taxi can be allotted.
Taxi-1 is allotted

*/
#include<bits/stdc++.h>
using namespace std;

class BookingDetails{
        public:
                int bookingId;
                int customerId;
                char from;
                char to;
                int pickUpTime;
                int dropTime;
                int amount;
};

class Taxi{

        private:
                int relaxationTime;
```

```
		int earning;
		char currentLocation;
		vector<BookingDetails> bookingDetails;
	public:
		Taxi()
		{
			relaxationTime = 0;
			earning = 0;
			currentLocation = 'A';
		}
		void addBooking(int bookingId , int customerId, char from , char to , int pickUpTime , int
dropTime ,int amount)
		{
			BookingDetails bookingDetail;
			bookingDetail.bookingId = bookingId;
			bookingDetail.customerId = customerId;
			bookingDetail.dropTime = dropTime;
			bookingDetail.from = from;
			bookingDetail.to = to;
			bookingDetail.pickUpTime = pickUpTime;
			bookingDetail.amount = amount;
			bookingDetails.push_back(bookingDetail);
		}
		int getEarning()
		{
			return earning;
		}
		void setEarning(int earning)
		{
			this->earning = earning;
		}
		int getRelaxationTime()
		{
			return relaxationTime;
		}
		void setRelaxationTime(int relaxationTime)
		{
			this->relaxationTime = relaxationTime;
		}
		int getCurrentLocation()
		{
			return currentLocation;
		}
		void setCurrentLocation(int currentLocation)
		{
			this->currentLocation = currentLocation;
		}
		vector<BookingDetails> getBookings()
```

```
		{
			return bookingDetails;
		}
};

class TaxiManagement{

	private:
		int noOfTaxis;
		vector<Taxi> taxi;
		int getDistanceBetweenPoints(char pointA , char pointB)
		{
			return abs( pointA - pointB);
		}

	public:
		static int currentTime;
		static int noOfBooknigs;
		TaxiManagement(int noOfTaxis)
		{
			this->noOfTaxis = noOfTaxis;
			taxi.resize(noOfTaxis);
		}

		void bookTaxi()
		{
			int customerId , pickUpTime;
			char pickUpPoint , dropPoint;
			int closestDistance = INT_MAX;

			cout<<"Customer Id: ";
			cin>>customerId;
			cout<<"Pickup Point: ";
			cin>>pickUpPoint;
			cout<<"Drop Point: ";
			cin>>dropPoint;
			cout<<"Pickup Time: ";
			cin>>pickUpTime;

			currentTime = pickUpTime;

			for(int i = 0 ; i < noOfTaxis ; i++)
				if(taxi[i].getRelaxationTime() <= pickUpTime)
					closestDistance = min(closestDistance ,
getDistanceBetweenPoints(taxi[i].getCurrentLocation() , pickUpPoint));
			int minimumEarning = INT_MAX;
			int bookedTaxiIndex = -1;
			for(int i = 0 ; i < noOfTaxis ; i++)
```

```
                                if(getDistanceBetweenPoints(taxi[i].getCurrentLocation() , pickUpPoint)
== closestDistance &&
                                taxi[i].getEarning() < minimumEarning)
                                {
                                        bookedTaxiIndex = i;
                                        minimumEarning = taxi[i].getEarning();
                                }
                        if(bookedTaxiIndex == -1)
                        {
                                cout<<"No taxis available as of now"<<endl;
                                return;
                        }

                        noOfBooknigs++;


        cout<<"++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++";
                        cout<<endl<<endl<<"Taxi can be alloted."<<endl;
                        cout<<"Taxi-";cout<<bookedTaxiIndex + 1;cout<<" is alloted"<<endl<<endl;
                        cout<<"It will be arriving in ";cout<<closestDistance;
                        cout<<" hours"<<endl;
                        cout<<"Total Fare : ";
                        cout<<getDistanceBetweenPoints(pickUpPoint , dropPoint)*200<<endl<<endl;
                        taxi[bookedTaxiIndex].setCurrentLocation(dropPoint);
                        taxi[bookedTaxiIndex].setEarning(taxi[bookedTaxiIndex].getEarning() +
getDistanceBetweenPoints(pickUpPoint , dropPoint)*200);
                        taxi[bookedTaxiIndex].setRelaxationTime(closestDistance + currentTime +
getDistanceBetweenPoints(pickUpPoint , dropPoint));
                        cout<<"Customer will reach the destination by ";
                        cout<<taxi[bookedTaxiIndex].getRelaxationTime();
                        cout<<":00 Hrs"<<endl<<endl<<endl;

        cout<<"++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++"<<endl<<endl;

                        taxi[bookedTaxiIndex].addBooking(noOfBooknigs , customerId , pickUpPoint ,
dropPoint , pickUpTime
,taxi[bookedTaxiIndex].getRelaxationTime(),getDistanceBetweenPoints(pickUpPoint , dropPoint)*200);

                        return;
                }
                void printBookingDetails()
                {
                        int i = 0;
                        for(auto it : taxi)
                        {
                                cout<<"Taxi No.\tTotal Earning"<<endl;
```

```
                                        cout<<"----------------------------------------------------------------"<<endl;
                                        cout<<"Taxi-";cout<<i +
1;cout<<"\t\t";cout<<it.getEarning()<<endl<<endl;
                                        if(it.getBookings().size() > 0)

        cout<<"BookingID\tCustomerID\tFrom\tTo\tPickupTime\tDropTime\tAmount"<<endl;
                                        for(auto itr : it.getBookings())
                                        {

        cout<<itr.bookingId<<"\t\t"<<itr.customerId<<"\t\t"<<itr.from<<"\t"<<itr.to<<"\t"<<itr.pickUpTi
me<<"\t\t"
                                                        <<itr.dropTime<<"\t\t"<<itr.amount<<endl;
                                        }
                                        i++;
                                        cout<<endl<<endl;
                                }
                        }
};

int TaxiManagement::currentTime = 0;
int TaxiManagement::noOfBooknigs = 0;

int main()
{
        TaxiManagement taxiManagement(4);
        while(1)
        {
                taxiManagement.bookTaxi();
                taxiManagement.printBookingDetails();
        }
}
```

```
/*
Food Delivery Booking

A food delivery company has 'n' number of delivery executives. For simplicity take the count as 5 but the program should work for any number of delivery executives (Let their names be identified as DE1, DE2....DE-n)

There are only 5 restaurants in the city for pick-up and 5 drop locations (Each location can have multiple customers) After delivering a food package , the delivery executive waits there for devlivery allotment.

Each customer is identified uniquely by a Customer-ID

Write a program that does the following :

Constraints :
```

1.Delivery charge for every single order is Rs 50 for the delivery executive.

2. If multiple orders (say n) are from the same delivery location within 15 mins period, combine orders to a maximum of 5 per delivery executive.

In such case, the delivery charge will be base rate Rs.50 + Rs.5 for every other order (50+5 * (n-1)).

3. An allowance of Rs.10 will be given for every trip made. Combined orders will be counted as a single trip.

4. Assign the subsequent bookings giving preference to the executive who has earned the least delivery charge among the other available delivery executives excluding trip allowance.

5. Every trip will take 30 mins to reach the destination.

Questions :

1. Write a function to handle booking.

2. Write a function to assign delivery executive

3. Write a function that can display delivery executive's activity thus far.

This should contain commision earned , allowance earned(calculated based on criteria 2 and 3).

Input 1

Customer ID: 1

Restaurant: A

Destionation Point : D

Time : 9.00 AM

Output

Booking ID : 1

Available Executives :

Executive	Delivery Charge Earned
DE1	0
DE2	0
DE3	0
DE4	0

DE5 0

Allotted Delivery Executive: DE1

Input 2

Customer ID: 2

Restaurant : B

Destination Point : A

Time : 10.00 AM

Output

Booking ID : 2

Available Executives :

Executive Delivery Charge Earned

DE1 50

DE2 0

DE3 0

DE4 0

DE5 0

Allotted Delivery Executive: DE2

Input 3

Customer ID: 3

Restaurant : B

Destionation Point : A

Time : 10.10 AM

Output

Booking ID : 3

Available Executives :

Executive Delivery Charge Earned

DE1 50

DE2 50

DE3 0

DE4 0

DE5 0

Allotted Delivery Executive: DE2 (because same location within 15mins)

Input 4

Customer ID: 3

Restaurant : D

Destionation Point : C

Time : 10.35 AM

Output

Booking ID : 3

Available Executives :
Executive Delivery Charge Earned
DE1 50
DE2 55
DE3 0
DE4 0
DE5 0
Allotted Delivery Executive: DE3

Delivery History
Output

TRIP	EXECUTIVE	RESTAURANT	DESTINATION POINT	ORDERS	PICK-UP_TIME	DELIVERY_TIME	DELIVERY CHARGE
1	DE1	A	D	1	9:15	9:45	50
2	DE2	B	A	2	10:15	10:45	55
3	DE3	D	C	1	10:50	11:20	50

Total earned

Executive	Allowance		Deliver Charges Total
DE1	10	50	60
DE2	10	55	65
DE3	10	50	60

*/

```
#include<bits/stdc++.h>
using namespace std;

struct Booking{
	string executive;
	char restraunt;
	char destinationPoint;
	int orders;
	int pickUpTime;
	int deliveryTime;
	int deliveryCharge;
};

class Executive{
	private:

		int earning;
		int totalEarning;
		int allowance;
		int noOfTrips;
		char restrauntLocation;
		char deliveryLocation;
		int pickUpTime;
		int relaxationTime;

	public:
		Executive()
		{
			this->earning = 0;
			this->totalEarning = 0;
			this->allowance = 0;
			this->noOfTrips = 0;
			this->pickUpTime = 0;
```

```
		this->relaxationTime = 0;
		this->deliveryLocation = '$';
		this->restrauntLocation = '$';
	}
	void setPickUpTime(int pickUpTime)
	{
		this->pickUpTime = pickUpTime;
	}
	int getPickUpTime()
	{
		return this->pickUpTime;
	}
	void setRelaxationTime(int relaxationTime)
	{
		this->relaxationTime = relaxationTime;
	}
	int getRelaxationTime()
	{
		return this->relaxationTime;
	}
	void addToEarning(int fare)
	{
		this->earning += fare;
	}
	int getEarning()
	{
		return this->earning;
	}
	void addNoOfTrips()
	{
		this->noOfTrips += 1;
	}
	int getNoOfTrips()
	{
		return this->noOfTrips;
	}
	int getAllowance()
	{
		this->allowance = 10*this->noOfTrips;
		return this->allowance;
	}
	int getTotalEarning()
	{
		this->totalEarning = this->allowance + this->earning;
		return this->totalEarning;
	}
	void setRestrauntAndDelivery(char restrauntLocation , char deliveryLocation)
	{
```

```
                this->restrauntLocation = restrauntLocation;
                this->deliveryLocation = deliveryLocation;
        }
        char getRestraunt()
        {
                return this->restrauntLocation;
        }
        char getDeliveryLocation()
        {
                return this->deliveryLocation;
        }
};

class DeliveryManagement{
    private:
        vector<Executive> executive;
        int noOfExecutives;
        vector<Booking> bookings;
    public:
        int convertToMinutes(string time)
        {
                int timeInMinutes = 0;
                if(time.size() == 7)
                        time = "0" + time;
                if(time[time.size() - 2] == 'P' || time[time.size() - 2] == 'p')
                        timeInMinutes += 12*60;
                timeInMinutes += ((time[0] - '0' )*10 + (time[1] - '0'))*60;
                timeInMinutes += (time[3] - '0')*10 + (time[4] - '0');
                return timeInMinutes;
        }
        string convertToHours(int time)
        {
                string result = "";
                string suffix;
                if(time/60 >= 12)
                {
                        time -= 12*60;
                        suffix = "PM";
                }
                else
                        suffix = "AM";
                result += (time/60)/10 + '0';
                result += (time/60)%10 + '0';
                result += ":";
                result += (time%60)/10 + '0';
                result += (time%60)%10 + '0';
                result += " " + suffix;
                return result;
```

```
		}
		static int currentTime;
		static int noOfDeliveries;

		DeliveryManagement(int noOfExecutives)
		{
			this->noOfExecutives = noOfExecutives;
			executive.resize(noOfExecutives);
		}

		void printAvailableExecutives()
		{
			cout<<"Available Executives"<<endl<<endl;
			cout<<"Executive\tDelivery Charge Earned"<<endl;
			int i = 0;
			for(auto it : executive)
			{
				cout<<"DE"<<i + 1<<"\t\t"<<it.getEarning()<<endl;
				i++;
			}
		}

		void foodDelivery()
		{
			int customerId;
			char restraunt , destinationPoint;
			string time;
			cout<<"\nCustomer ID : ";
			cin>>customerId;
			cout<<"Restraunt : ";
			cin>>restraunt;
			cout<<"Destination Point : ";
			cin>>destinationPoint;
			cout<<"Time : ";
			cin.ignore();
			getline(cin , time);
			currentTime = convertToMinutes(time);

			bool alreadGoing = false;
			int minimumEarning = INT_MAX;
			int bookedIndex;
			int i = 0;
			for(Executive it : executive)
			{
				if(it.getRestraunt() == restraunt && it.getDeliveryLocation() ==
destinationPoint
					&& it.getEarning() < minimumEarning && it.getPickUpTime() >=
currentTime)
```

```
                                {
                                        alreadGoing = true;
                                        minimumEarning = it.getEarning();
                                        bookedIndex = i;
                                }
                                i++;
                        }

                        if(alreadGoing)
                        {
                                cout<<"\n\nBooking Id : "<<noOfDeliveries<<endl;
                                printAvailableExecutives();
                                cout<<"Alloted Delivery Executive: DE"<<bookedIndex + 1<<"(because
same location within 15 minutes"<<endl<<endl;
                                executive[bookedIndex].addToEarning(5);

                                bookings[noOfDeliveries - 1].orders++;
                                bookings[noOfDeliveries - 1].deliveryCharge += 5;
                                return;
                        }

                        noOfDeliveries++;
                        cout<<"\n\nBooking Id : "<<noOfDeliveries<<endl;
                        printAvailableExecutives();

                        minimumEarning = INT_MAX;
                        i = noOfExecutives - 1;
                        for(auto it = executive.rbegin() ; it != executive.rend() ; it++)
                        {
                                if(it->getRelaxationTime() <= currentTime && it->getEarning() <=
minimumEarning)
                                {
                                        minimumEarning = it->getEarning();
                                        bookedIndex = i;
                                }
                                i--;
                        }
                        cout<<"Alloted Delivery Executive: DE"<<bookedIndex + 1<<endl;

                        executive[bookedIndex].addToEarning(50);
                        executive[bookedIndex].addNoOfTrips();
                        executive[bookedIndex].setPickUpTime(currentTime + 15);
                        executive[bookedIndex].setRelaxationTime(currentTime + 45);
                        executive[bookedIndex].setRestrauntAndDelivery(restraunt , destinationPoint);

                        Booking booking;
                        booking.deliveryCharge = executive[bookedIndex].getEarning();
                        booking.deliveryTime = executive[bookedIndex].getRelaxationTime();
```

```
                booking.orders = 1;
                booking.pickUpTime = executive[bookedIndex].getPickUpTime();
                booking.executive = "DE";
                booking.executive += (bookedIndex + 1 +'0');
                booking.destinationPoint = executive[bookedIndex].getDeliveryLocation();
                booking.restraunt = executive[bookedIndex].getRestraunt();

                bookings.push_back(booking);
        }

        void printDeliveryDetails()
        {
                cout<<"\n\nDelivery History"<<endl;
                cout<<"Trip\tExecutive\tRestraunt\tDestination Point\tOrders\tPick Up
Time\tDelivery Time\tDelivery Charge\n";
                int i = 0;
                for(auto it : bookings)
                {
                        cout<<i +
1<<"\t\t"<<it.executive<<"\t\t"<<it.restraunt<<"\t\t"<<it.destinationPoint<<"\t\t"<<it.orders

        <<"\t"<<convertToHours(it.pickUpTime)<<"\t"<<convertToHours(it.deliveryTime)<<"\t\t"<<it.de
liveryCharge<<endl;
                        i++;
                }
        }

        void printTotalEarned()
        {
                int i = 0;
                cout<<"\n\nTotal Earning\n";
                cout<<"Executive\tAllowance\tDelivery Charge\tTotal\n";

                for(auto it : executive)
                {

        cout<<"DE"<<i+1<<"\t\t"<<it.getAllowance()<<"\t\t"<<it.getEarning()<<"\t\t"<<it.getTotalEarnin
g()<<endl;
                        i++;
                }
        }
};

int DeliveryManagement::currentTime = 0;
int DeliveryManagement::noOfDeliveries = 0;

int main()
{
```

```
	DeliveryManagement deliveryManagement(5);
	while(1)
	{
		cout<<"\nChoose from the following options : "<<endl;
		cout<<"1. Handle Delivery\n2. Print Delivery Details\n3. Print Executive Details\n";
		int choice;
		cout<<endl<<"Your Choice : ";
		cin>>choice;
		if(choice == 1)
			deliveryManagement.foodDelivery();
		else if (choice == 2)
			deliveryManagement.printDeliveryDetails();
		else if (choice == 3)
			deliveryManagement.printTotalEarned();
		else
			break;
	}
}
```

```
/*
Hyperloop Passenger Booking

Problem :

You are to program a passenger booking system for a hyperloop transport of a particular station.

Assumptions

1. All the hyperloop routes & the starting station will be given as input. (See the below input section for details)
2. Assumptions is that, the distance and time taken travel a route is constant.

3. A pod can travel from one connection to another in any direction in the given route. (i.e both A to B and B to A)
4. One pod can accommodate only one passenger at a time.

5. Passengers will be booked to their pods one by one.

a. All passengers will start from the given starting station.

b. Whenever a pod is started, the oldest person in the queue will boarding pod first.
6. Passengers can arrive at any time as well as pods can start at any time.

7. Assume infinite supply of pods and collision will never happen.


System Logic :
```

Whenever a pod is started, the pod should pickup

- the oldest person from the passenger queue and
- take the route which has minimum interconnections

The command line should handle the following Commands

INIT

ADD_PASSENGER

START_POD

PRINT_Q

The arguments and details of the commands are mentioned below.

1. INIT Command - Initializes the system with

a. Number of interconnecting routes (N) and the Starting station.

b. Next N lines denotes connection between two interconnections. Without initializing - All other commands should throw proper error.

2. ADD_PASSENGER command adds passenger to the line.

a. ADD_PASSENGER X adds X number of passengers to the line. X lines following the ADD command denotes the passenger's name, age and destination
b. NAME AGE DEST

3. START_POD command starts pod with a passenger having highest age to his destination following the minimum interconnection points. Print the passenger name and route.
a. START_POD X starts X number of passengers of highest age. (X lines are printed with name and route)
b. NAME ROUTE

4. PRINT_Q command prints the number of passengers and their details who are remaining in the queue.
a. COUNT

b. NAME AGE

EXAMPLE

Input: Output:

INIT 7 A
A B
A C
B D
B C
B E
C E
D E

ADD_PASSENGER 1
RAVI 22 C

ADD_PASSENGER 2
HARI 33 D
BALA 10 E

START_POD 1 HARIABD

PRINT_Q 2
RAVI 22
BALA 10

ADD_PASSENGER 1
KATHIR 22 B

PRINT_Q 3
RAVI 22
BALA 10
KATHIR 22

START_POD 2 RAVI A C
KATHIR A B

START_POD 1 BALAABEOR(ACE)

*/

```
#include<bits/stdc++.h>
using namespace std;

class Passenger{
        private:
                string name;
```

```
        int age;
        char destination;
    public:

        Passenger(string name , int age , char destination)
        {
            this->name = name;
            this->age = age;
            this->destination = destination;
        }
        string getName()
        {
            return this->name;
        }
        int getAge()
        {
            return this->age;
        }
        char getDestination()
        {
            return this->destination;
        }
};

class BookingManagement{
    private:
        vector<Passenger> passengers;
        priority_queue<pair<int , int>> passengerQueue;
        unordered_map<char , vector<char>> stations;
        char startingStation;

        void BFS(char destination)
        {
            queue<char> q;
            unordered_map<char , char> prev;
            set<char> visited;
            q.push(this->startingStation);
            while(!q.empty())
            {
                char node = q.front();
                q.pop();
                for(auto it : stations[node])
                {
                    if(visited.find(it) == visited.end())
                    {
                        q.push(it);
                        visited.insert(it);
                        prev[it] = node;
```

```
                                        }
                                }
                        }
                        vector<char> result;
                        result.push_back(destination);
                        while(destination != this->startingStation || prev.find(destination) ==
prev.end())
                        {
                                destination = prev[destination];
                                result.push_back(destination);
                        }
                        reverse(result.begin() , result.end());
                        if(result[0] == this->startingStation)
                        {
                                for(auto it : result)
                                        cout<<it<<" ";
                                cout<<endl;
                        }
                        else
                                cout<<"No route present to the destination"<<endl;
                }
        public:
                static int noOfPassengers;
                static int noOfInterConnections;
                void init()
                {
                        cin>>this->noOfInterConnections>>this->startingStation;
                        char u , v;
                        for(int i = 0 ; i < this->noOfInterConnections ; i++)
                        {
                                cin>>u>>v;
                                stations[u].push_back(v);
                                stations[v].push_back(u);
                        }
                }
                void addPassenger()
                {
                        int n;
                        cin>>n;
                        while(n--)
                        {
                                string name;
                                int age;
                                char destination;

                                cin>>name>>age>>destination;
                                Passenger passenger(name , age ,destination);
                                passengers.push_back(passenger);
```

```
                                passengerQueue.push(pair<int , int> (passenger.getAge() , this->noOfPassengers));
                                this->noOfPassengers++;

                        }
                }
                void startPod()
                {
                        int n;
                        cin>>n;
                        while(n--)
                        {
                                cout<<passengers[passengerQueue.top().second].getName()<<" ";
                                BFS(passengers[passengerQueue.top().second].getDestination());
                                passengerQueue.pop();
                        }
                }
                void printQueue()
                {
                        cout<<passengerQueue.size()<<endl;
                        priority_queue<pair<int , int>> passengerQueueCopy = passengerQueue;
                        while(!passengerQueueCopy.empty())
                        {
                                cout<<passengers[passengerQueueCopy.top().second].getName()<<" "<<passengers[passengerQueueCopy.top().second].getAge()<<endl;
                                passengerQueueCopy.pop();
                        }
                }
};

int::BookingManagement::noOfPassengers = 0;
int::BookingManagement::noOfInterConnections = 0;
int main()
{
        BookingManagement bookingManagement;
        while(1)
        {
                string input;
                cin>>input;
                if(input == "INIT")
                        bookingManagement.init();
                else if(input == "ADD_PASSENGER")
                {
                        bookingManagement.addPassenger();
                        bookingManagement.printQueue();
                }
                else if(input == "START_POD")
                        bookingManagement.startPod();
```

```
		else if(input == "PRINT_Q")
			bookingManagement.printQueue();
	}
}
```

```
/*
Write an application for booking railway ticket reservation system. The application should have four functionalities.

Book
Cancel
Print booked tickets (details with summary)
Print available tickets (details with summary)
Conditions for booking:

There are a total of 63 berths for 63 confirmed tickets, 9 berths for 18 RAC tickets and 10 tickets in waiting-list. If the waiting-list ticket count goes above 10, print as 'No tickets available'. The following passenger details should be obtained from the user.

Name
Age
Gender
Berth Preference
The tickets should not be allocated for children below age 5.But, their details should be stored. Lower berth should be allocated for persons whose age is above 60 and ladies with children if available. Side-lower berths should be allocated for RAC passengers.

Conditions for cancelling:

Whenever a ticket is cancelled, a ticket from RAC should be confirmed and a waiting-list ticket should move to RAC.

Conditions for printing booked tickets:

Print all the tickets that are filled along with the passenger details and at the end, print the total number of tickets that are filled.

Conditions for printing available tickets:

Print all the tickets that are unoccupied and at the end, print the total number of tickets that are unoccupied.


>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>RAILWAY<<<<<<<<<<<<<<<<<<<<
<<<<<<<


*/
```

```
/*
Java Code
package com.reservation;

import java.util.ArrayList;

public class Train {

	ArrayList<Ticket> confirmedTickets=new ArrayList<Ticket>();
	ArrayList<Ticket> racTickets=new ArrayList<Ticket>();
	ArrayList<Ticket> waitingListTickets=new ArrayList<Ticket>();
	Berth berth =new Berth();



	public void bookTicket(Ticket ticket) {

		if(confirmedTickets.size()<2) {
			berth.allocateBerth(ticket);
			confirmedTickets.add(ticket);
		}
		else if(racTickets.size()<1) {
			ticket.setConfirmationStatus("In RAC");
			racTickets.add(ticket);
		}
		else if(waitingListTickets.size()<1){
			ticket.setConfirmationStatus("In Waiting list");
			waitingListTickets.add(ticket);
		}
		else {
			System.out.println("No Tickets Available");
			return;
		}

		displayTicket(ticket);
	}



	public void cancelTicket(Ticket ticket) {

		if(removeTicket(ticket,confirmedTickets))
		{
			confirmedTickets.add(confirmedTickets.size(),racTickets.get(0));
			racTickets.remove(0);
			if(!waitingListTickets.isEmpty())
```

```
                    {
                    racTickets.add(racTickets.size(),waitingListTickets.get(0));
                    waitingListTickets.remove(0);
                    }
          }else if(removeTicket(ticket,waitingListTickets)) {
                    return;
          }
          else if(removeTicket(ticket,racTickets)) {
                    if(!waitingListTickets.isEmpty()) {
                   racTickets.add(racTickets.size()-1,waitingListTickets.get(0));
                    waitingListTickets.remove(0);
                    }
          }
          else {
                              System.out.println("your search does not match");
                              return;
                    }



}

private boolean removeTicket(Ticket ticket,ArrayList<Ticket> ticketBookings ) {
          for(Ticket tickets:ticketBookings) {
      if(tickets.getName().equals(ticket.getName()) && tickets.getAge()==ticket.getAge()) {
           racTickets.get(0).setConfirmationStatus(tickets.getConfirmationStatus());
           confirmedTickets.remove(tickets);

           return true;
      }
          }
          return false;
}



private ArrayList<Ticket> findTicketStatus(Ticket ticket) {
          if(confirmedTickets.contains(ticket))
                    return confirmedTickets;
          else if(racTickets.contains(ticket))
                    return racTickets;
          else {
                    return waitingListTickets;
         }
}

public void displayTicket(Ticket ticket) {
```

```
System.out.println(ticket.getName()+"\n"+ticket.getAge()+"\n"+ticket.getGender()+"\n"+ticket.getConfi
rmationStatus()+" "
                                        +findTicketStatus(ticket).indexOf(ticket)
                                        );
                }



        public void displayAllTickets() {
                for(Ticket ticket:confirmedTickets) {

System.out.println(ticket.getName()+"\n"+ticket.getAge()+"\n"+ticket.getGender()+"\n"+ticket.getConfi
rmationStatus()+" "
                                        +findTicketStatus(ticket).indexOf(ticket)
                                        );
                }
                for(Ticket ticket:racTickets) {

System.out.println(ticket.getName()+"\n"+ticket.getAge()+"\n"+ticket.getGender()+"\n"+ticket.getConfi
rmationStatus()+" "
                                        +findTicketStatus(ticket).indexOf(ticket)
                                        );
                        }
                for(Ticket ticket:waitingListTickets) {

System.out.println(ticket.getName()+"\n"+ticket.getAge()+"\n"+ticket.getGender()+"\n"+ticket.getConfi
rmationStatus()+" "
                                        +findTicketStatus(ticket).indexOf(ticket)
                                        );
                        }
        }
        }
*/
```

```
/*
Simple Mailing System
Write a simple mailing application, that performs the following :

Duration - 3hrs

1.      User creation
The program must accept input to add users. A user can have a user name, email and password. User
names and email must be unique. Display the list of users available in the system.

2.      User Groups creation
The program must accept input to add groups. A group can have a group name, group email id,
password, description and group members. A User can belong to more than one Group. A User can be
added or removed from the Group anytime. Group names and email must be unique. Display the list of
```

available groups in the system.

3. Compose Mail

Accept input to send a mail. The input can contain the "From", "To", "Mail Subject" and "Content". The "To address" can be a user's email id or a group's email id.

4. Viewing Inbox

Display the list of emails that belong to a user in the received order. Recently received mails appear at the top.

5. Viewing Sent Mails

Display the list of emails that are sent by a user. Recently sent mails appear at the top.

6. Delete mails from Inbox or Sent folder

Accept inputs to delete a mail from a user's "Inbox" or "Sent mails" folder.

7. Recall Mail

A User can Recall the mails sent by him. When a mail is recalled, the mail gets deleted from the Inbox of the received users. The mail however remains in the user's Sent mail folder. When viewing the Sent mails, show the "Recalled" status.

8. Inbox Sharing

A User can share his Inbox folder with other users. When users view their Inbox, list the contents of the shared Inbox folder also. A User can revoke the sharing anytime, and when revoked, the shared folder no longer gets visible to the earlier shared users.

Hote

a. Do proper validation of inputs and print error messages appropriately, during all operations
b. For example, if a invalid "To" address is provided during mail sending, print a valid error message
c. Sample Input/output are provided for reference only and does not cover all operations mentioned above.

Sample Input/Output

User Creation :
1) Create User
2) Create Group
3) Group Assignment
4) Compose Mail
S) Inbox
6) Sent Mail
7) Delete Mail
8) Recall
9) Share Inbox
10)) Exit
Select any option : 1
Enter User Name : U1

Enter Email ID : user1 @abc.com Enter Password : 12345678 User Created

uroup reazion :
1) Create User
2) Create Group
3) Group Assignment
4) Compose Mail
5) Inbox
6) Sent Mail
7) Delete Mail
8) Recall
9) Share Inbox ß) Exit

Select any option : 2
Enter Group Name : G1
Enter Group Mail ID : group1 @abc.com Enter Group Mail Password : 12345678 Enter Group Description : Lorem Ipsum Group Created

Group Assignment :
1) Create User
2) Create Group
3) Group Assignment
4) Compose Mail
S) Inbox
6) Sent Mail
7) Delete Mail
8) Recall
9) Share Inbox
10) Exit
Select any option : 3
, ...
8) Recall
9) Share Inbox
10) Exit

Select any option : 2
Enter Group Name : G1
Enter Group Mail ID : group1 @abc.com Enter Group Mail Password : 12345678 Enter Group Description : Lorem Ipsum Group Created

Group Assignment :
1) Create User
2) Create Group
3) Group Assignment
4) Compose Mail
S) Inbox
6) Sent Mail
7) Delete Mail

8) Recall
9) Share Inbox
10) Exit
Select any option : 3
, ...
8) Recall
9) Share Inbox
10) Exit

Select any option : 2
Enter Group Name : G1
Enter Group Mail ID : group1 @abc.com Enter Group Mail Password : 12345678 Enter Group Description : Lorem Ipsum Group Created

Group Assignment :
1) Create User
2) Create Group
3) Group Assignment
4) Compose Mail
S) Inbox
6) Sent Mail
7) Delete Mail
8) Recall
9) Share Inbox
10) Exit
Select any option : 3
, ...
8) Recall
9) Share Inbox
10) Exit

*/

```
#include<bits/stdc++.h>
using namespace std;

class Mail{
	private:
		string from;
		string to;
		string subject;
		string content;
	public:
		void setFrom(string from)
		{
			this->from = from;
		}
		void setTo(string to)
```

```
		{
			this->to = to;
		}
		void setSubject(string subject)
		{
			this->subject = subject;
		}
		void setContent(string content)
		{
			this->content= content;
		}
		string getFrom()
		{
			return from;
		}
		string getTo()
		{
			return to;
		}
		string getContent()
		{
			return content;
		}
		string getSubject()
		{
			return subject;
		}

};

class User{
	private:
		string username;
		string email;
		string password;
	public:
		vector<Mail> receivedMails;
		vector<Mail> sentMails;
		void setUserName(string username)
		{
			this->username = username;
		}
		void setEmail(string email)
		{
			this->email = email;
		}
		void setPassword(string password)
		{
```

```
                this->password = password;
        }
        string getUserName()
        {
                return this->username;
        }
        string getPassword()
        {
                return this->password;
        }
        string getEmail()
        {
                return this->email;
        }
};
class Group{
    private:
        string username;
        string email;
        string password;
        string description;
    public:
        vector<Mail> receivedMails;
        vector<Mail> sentMails;
        unordered_map<string , User> users;
        void setUserName(string username)
        {
                this->username = username;
        }
        void setEmail(string email)
        {
                this->email = email;
        }
        void setPassword(string password)
        {
                this->password = password;
        }
        void setDescription(string description)
        {
                this->description = description;
        }
        string getUserName()
        {
                return this->username;
        }
        string getPassword()
        {
                return this->password;
```

```
        }
        string getEmail()
        {
                return this->email;
        }
};


class UserManagement{
    private:
        set<string> emails;
    public:
        unordered_map<string , User> users;

        void createUser()
        {
                string name , email , password;
                cout<<"Enter User Name: ";
                cin>>name;
                cout<<"Enter Emaile Id: ";
                cin>>email;
                cout<<"Enter Password : ";
                cin>>password;

                User user;
                user.setUserName(name);
                user.setEmail(email);
                user.setPassword(password);
                if(users.find(name) != users.end())
                {
                        cout<<"User Name already taken"<<endl<<endl;
                        return;
                }
                if(emails.find(email) != emails.end())
                {
                        cout<<"Email already taken"<<endl<<endl;
                        return;
                }

                users[name] = user;
                emails.insert(email);
                cout<<"User Created"<<endl<<endl;
        }
        void printUsers()
        {
                cout<<endl<<"Printing users : "<<endl;
                for(auto it: this->users)
                {
```

```
                                cout<<"Username : "<<it.first<<endl;
                                cout<<"Email : "<<it.second.getEmail()<<endl;
                                cout<<endl;
                        }
                }

}userManagement;

class GroupManagement{
        private:
                set<string> emails;
        public:
                unordered_map<string , Group> groups;
                void createGroup()
                {
                        string name , email , password ,description;
                        cout<<"Enter Group Name: ";
                        cin>>name;
                        cout<<"Enter Group Mail Id: ";
                        cin>>email;
                        cout<<"Enter Group Mail Password : ";
                        cin>>password;
                        cout<<"Enter Group Description : ";
                        cin>>description;

                        Group group;
                        group.setUserName(name);
                        group.setEmail(email);
                        group.setPassword(password);
                        group.setDescription(description);
                        if(groups.find(name) != groups.end())
                        {
                                cout<<"Group Name already taken"<<endl<<endl;
                                return;
                        }
                        if(emails.find(email) != emails.end())
                        {
                                cout<<"Group Email already taken"<<endl<<endl;
                                return;
                        }

                        groups[name] = group;
                        emails.insert(email);
                        cout<<"Group Created"<<endl<<endl;
                }
                void groupAssignment()
                {
                        string groupName , userName , option;
```

```
                        cout<<"Enter Group Name: ";
                        cin>>groupName;
                        cout<<"Enter User : ";
                        cin>>userName;
                        cout<<"Add / Remove   : ";
                        cin>>option;
                        if(userManagement.users.find(userName) == userManagement.users.end())
                        {
                                cout<<"User does not exist"<<endl<<endl;
                                return;
                        }
                        if(groups.find(groupName) == groups.end())
                        {
                                cout<<"Group does not exist"<<endl;
                                return;
                        }
                        if(option == "Add")
                        {
                                if(groups[groupName].users.find(userName) !=
groups[groupName].users.end())
                                {
                                        cout<<"User already present in the group"<<endl<<endl;
                                        return;
                                }
                                groups[groupName].users[userName] =
userManagement.users[userName];
                                cout<<"User added to group."<<endl<<endl;
                        }
                        else if(option == "Remove")
                        {
                                if(groups[groupName].users.find(userName) ==
groups[groupName].users.end())
                                {
                                        cout<<"User not present present in the group"<<endl<<endl;
                                        return;
                                }
                                groups[groupName].users.erase(userName);
                                cout<<endl<<"Removed the user"<<endl<<endl;
                        }
                        cout<<"All users present int the group : "<<endl;
                        for(auto it : groups[groupName].users)
                        {
                                cout<<"UserName : "<<it.first<<endl;
                                cout<<"Email : "<<it.second.getEmail()<<endl;
                        }
                        cout<<endl<<endl;
                }
                void printGroups()
```

```
		{
			cout<<endl<<"Printing groups : "<<endl;
			for(auto it: this->groups)
			{
				cout<<"Group Username : "<<it.first<<endl;
				cout<<"Group Email : "<<it.second.getEmail()<<endl;
				cout<<endl;
			}
		}

}groupManagement;

class MailManagement{
	private:
		vector<Mail> mails;
	public:
		void compose()
		{
			string user , toAddress, subject ,content , to;
			cout<<"Enter the user : ";
			cin>>user;
			cout<<"Enter to address : ";
			cin>>toAddress;
			cout<<"Enter subject : ";
			cin>>subject;
			cout<<"Enter content : ";
			cin>>content;
			Mail tempMail;
			bool userFrom = false, userTo = false;
			if(userManagement.users.find(user) != userManagement.users.end())
				userFrom = true;
			if(!userFrom)
			{
				if(groupManagement.groups.find(user) ==
groupManagement.groups.end())
				{
					cout<<"From user not found !"<<endl<<endl;
					return;
				}
			}
			for(auto it : userManagement.users)
			{
				if(it.second.getEmail() == toAddress)
				{
					to = it.first;
					userTo = true;
					break;
				}
```

```
                }
                if(!userTo)
                {
                        bool found = false;
                        for(auto it : groupManagement.groups)
                        {
                                if(it.second.getEmail() == toAddress)
                                {
                                        to = it.first;
                                        found = true;
                                        break;
                                }
                        }
                        if(!found)
                        {
                                cout<<"To user not found"<<endl<<endl;
                                return;
                        }
                }

                tempMail.setContent(content);
                tempMail.setFrom(userManagement.users[user].getEmail());
                tempMail.setTo(toAddress);
                tempMail.setSubject(subject);
                if(userFrom)
                        userManagement.users[user].sentMails.push_back(tempMail);
                else
                        groupManagement.groups[user].sentMails.push_back(tempMail);
                if(userTo)
                        userManagement.users[to].receivedMails.push_back(tempMail);
                else
                {
                        groupManagement.groups[to].receivedMails.push_back(tempMail);
                        for(auto it : groupManagement.groups[to].users)
                        {
                                it.second.receivedMails.push_back(tempMail);

userManagement.users[it.first].receivedMails.push_back(tempMail);
                        }
                }

                cout<<"Mail sent"<<endl<<endl;
        }
        void inbox()
        {
                string user;
                cout<<"Enter the User Name : ";
                cin>>user;
```

```
                        bool userFrom = false;
                        if(userManagement.users.find(user) != userManagement.users.end())
                                userFrom = true;
                        if(!userFrom)
                        {
                                if(groupManagement.groups.find(user) ==
groupManagement.groups.end())
                                {
                                        cout<<"From user not found !"<<endl<<endl;
                                        return;
                                }
                        }
                        if(userFrom)
                        {
                                int i = 1;
                                cout<<"S.No.\t\tFrom\t\tTo\t\tSubject\t\tContent"<<endl;
                                cout<<"--------------------------------------------------------------------------------
"<<endl;
                                vector<Mail> tempMails = userManagement.users[user].receivedMails;
                                for(auto it = tempMails.rbegin() ; it  != tempMails.rend() ; it++)
                                {
                                        cout<<i<<"\t\t"<<it->getFrom()<<"\t\t"<<it-
>getTo()<<"\t\t"<<it->getSubject()<<"\t\t"<<it->getContent()<<endl;
                                        i++;
                                }
                                cout<<endl;
                        }
                        else
                        {
                                int i = 1;
                                cout<<"S.No.\t\tFrom\t\tTo\t\tSubject\t\tContent"<<endl;
                                cout<<"--------------------------------------------------------------------------------
"<<endl;
                                vector<Mail> tempMails =
groupManagement.groups[user].receivedMails;
                                for(auto it = tempMails.rbegin() ; it  != tempMails.rend() ; it++)
                                {
                                        cout<<i<<"\t\t"<<it->getFrom()<<"\t\t"<<it-
>getTo()<<"\t\t"<<it->getSubject()<<"\t\t"<<it->getContent()<<endl;
                                        i++;
                                }
                                cout<<endl;
                        }

                }
                void sentMail()
                {
                        string user;
```

```
                        cout<<"Enter user name : ";
                        cin>>user;
                        bool userFrom = false;
                        if(userManagement.users.find(user) != userManagement.users.end())
                                userFrom = true;
                        if(!userFrom)
                        {
                                if(groupManagement.groups.find(user) ==
groupManagement.groups.end())
                                {
                                        cout<<"From user not found !"<<endl<<endl;
                                        return;
                                }
                        }
                        if(userFrom)
                        {
                                int i = 1;
                                cout<<"S.No.\t\tTo\t\tSubject\t\tContent"<<endl;
                                cout<<"--------------------------------------------------------------------------------
"<<endl;
                                vector<Mail> tempMails = userManagement.users[user].sentMails;
                                for(auto it = tempMails.rbegin() ; it  != tempMails.rend() ; it++)
                                {
                                        cout<<i<<"\t\t"<<it->getTo()<<"\t\t"<<it-
>getSubject()<<"\t\t"<<it->getContent()<<endl;
                                        i++;
                                }
                                cout<<endl;
                        }
                        else
                        {
                                int i = 1;
                                cout<<"S.No.\t\tTo\t\tSubject\t\tContent"<<endl;
                                cout<<"--------------------------------------------------------------------------------
"<<endl;
                                vector<Mail> tempMails = groupManagement.groups[user].sentMails;
                                for(auto it = tempMails.rbegin() ; it  != tempMails.rend() ; it++)
                                {
                                        cout<<i<<"\t\t"<<it->getTo()<<"\t\t"<<it-
>getSubject()<<"\t\t"<<it->getContent()<<endl;
                                        i++;
                                }
                                cout<<endl;
                        }
                }
                void deleteMail()
                {
                        string user ,option;
```

```
                int n;
                cout<<"Enter the username : ";
                cin>>user;
                cout<<"From Inbox/Sent : ";
                cin>>option;
                cout<<"Enter serial number : ";
                cin>>n;
                bool userFrom = false;
                if(userManagement.users.find(user) != userManagement.users.end())
                        userFrom = true;
                if(!userFrom)
                {
                        if(groupManagement.groups.find(user) ==
groupManagement.groups.end())
                        {
                                cout<<"From user not found !"<<endl<<endl;
                                return;
                        }
                }
                if(option == "Inbox")
                {
                        if(userFrom)
                        {
                                vector<Mail> tempMails =
userManagement.users[user].receivedMails;
                                if(n > tempMails.size())
                                {
                                        cout<<"Index not present"<<endl;
                                        return;
                                }

        userManagement.users[user].receivedMails.erase(userManagement.users[user].receivedMails.
end() - n);

                                int i = 1;
                                cout<<"S.No.\t\tFrom\t\tTo\t\tSubject\t\tContent"<<endl;
                                cout<<"----------------------------------------------------------------------
---"<<endl;
                                tempMails = userManagement.users[user].receivedMails;
                                for(auto it = tempMails.rbegin() ; it  != tempMails.rend() ; it++)
                                {
                                        cout<<i<<"\t\t"<<it->getFrom()<<"\t\t"<<it-
>getTo()<<"\t\t"<<it->getSubject()<<"\t\t"<<it->getContent()<<endl;
                                        i++;
                                }
                                cout<<endl;
                        }
                        else
```

```
                                        {
                                                vector<Mail> tempMails =
groupManagement.groups[user].receivedMails;
                                                if(n > tempMails.size())
                                                {
                                                        cout<<"Index not present"<<endl;
                                                        return;
                                                }

        groupManagement.groups[user].receivedMails.erase(groupManagement.groups[user].received
Mails.end() - n);
                                                int i = 1;
                                                cout<<"S.No.\t\tFrom\t\tTo\t\tSubject\t\tContent"<<endl;
                                                cout<<"----------------------------------------------------------------------
---"<<endl;
                                                tempMails = groupManagement.groups[user].receivedMails;
                                                for(auto it = tempMails.rbegin() ; it  != tempMails.rend() ; it++)
                                                {
                                                        cout<<i<<"\t\t"<<it->getFrom()<<"\t\t"<<it-
>getTo()<<"\t\t"<<it->getSubject()<<"\t\t"<<it->getContent()<<endl;
                                                        i++;
                                                }
                                                cout<<endl;
                                        }
                                }
                                else
                                {
                                        if(userFrom)
                                        {
                                                vector<Mail> tempMails =
userManagement.users[user].sentMails;
                                                if(n > tempMails.size())
                                                {
                                                        cout<<"Index not present"<<endl;
                                                        return;
                                                }

        userManagement.users[user].sentMails.erase(userManagement.users[user].sentMails.end() -
n);
                                                int i = 1;
                                                cout<<"S.No.\t\tTo\t\tSubject\t\tContent"<<endl;
                                                cout<<"----------------------------------------------------------------------
---"<<endl;
                                                 tempMails = userManagement.users[user].sentMails;
                                                for(auto it = tempMails.rbegin() ; it  != tempMails.rend() ; it++)
                                                {
                                                        cout<<i<<"\t\t"<<it->getTo()<<"\t\t"<<it-
>getSubject()<<"\t\t"<<it->getContent()<<endl;
```

```
                                        i++;
                                }
                                cout<<endl;
                        }
                        else
                        {
                                vector<Mail> tempMails =
groupManagement.groups[user].sentMails;
                                if(n > tempMails.size())
                                {
                                        cout<<"Index not present"<<endl;
                                        return;
                                }

        groupManagement.groups[user].sentMails.erase(groupManagement.groups[user].sentMails.en
d() - n);
                                int i = 1;
                                cout<<"S.No.\t\tTo\t\tSubject\t\tContent"<<endl;
                                cout<<"----------------------------------------------------------------------------
---"<<endl;
                                tempMails = groupManagement.groups[user].sentMails;
                                for(auto it = tempMails.rbegin() ; it  != tempMails.rend() ; it++)
                                {
                                        cout<<i<<"\t\t"<<it->getTo()<<"\t\t"<<it-
>getSubject()<<"\t\t"<<it->getContent()<<endl;
                                        i++;
                                }
                                cout<<endl;
                        }
                }
        }

}mailManagement;

int main()
{
        int option = 11;
        while(option != 10)
        {
                cout<<"1)Create User\n2)Create Group\n3)Group
Assignment\n4)Compose\n5)Inbox\n6)Sent Mail\n7)Delete Mail\n8)Recall\n9)Recall\n10)Exit";
                cout<<endl;
                cout<<"Select any option: ";
                cin>>option;
                if(option > 10 || option < 1)
                {
                        cout<<"Improper input"<<endl;
                        return 0;
```

```
                }
                if(option == 1)
                {
                        userManagement.createUser();
                        userManagement.printUsers();
                }
                else if(option == 2)
                {
                        groupManagement.createGroup();
                        groupManagement.printGroups();
                }
                else if(option == 3)
                {
                        groupManagement.groupAssignment();
                }
                else if(option == 4)
                {
                        mailManagement.compose();
                }
                else if(option == 5)
                {
                        mailManagement.inbox();
                }
                else if(option == 6)
                {
                        mailManagement.sentMail();
                }
                else if(option == 7)
                {
                        mailManagement.deleteMail();
                }

        }

}
```

```
/*
Dungeon Game
Hard

4302

83

Add to List

Share
```

The demons had captured the princess and imprisoned her in the bottom-right corner of a dungeon. The dungeon consists of m x n rooms laid out in a 2D grid. Our valiant knight was initially positioned in the top-left room and must fight his way through dungeon to rescue the princess.

The knight has an initial health point represented by a positive integer. If at any point his health point drops to 0 or below, he dies immediately.

Some of the rooms are guarded by demons (represented by negative integers), so the knight loses health upon entering these rooms; other rooms are either empty (represented as 0) or contain magic orbs that increase the knight's health (represented by positive integers).

To reach the princess as quickly as possible, the knight decides to move only rightward or downward in each step.

Return the knight's minimum initial health so that he can rescue the princess.

Note that any room can contain threats or power-ups, even the first room the knight enters and the bottom-right room where the princess is imprisoned.

Example 1:

Input: dungeon = [[-2,-3,3],[-5,-10,1],[10,30,-5]]
Output: 7
Explanation: The initial health of the knight must be at least 7 if he follows the optimal path: RIGHT-> RIGHT -> DOWN -> DOWN.
Example 2:

Input: dungeon = [[0]]
Output: 1

Constraints:

m == dungeon.length
n == dungeon[i].length
1 <= m, n <= 200
-1000 <= dungeon[i][j] <= 1000

*/

```
#include<bits/stdc++.h>
using namespace std;

struct hash_pair {
   template <class T1, class T2>
   size_t operator()(const pair<T1, T2>& p) const
   {
      auto hash1 = hash<T1>{}(p.first);
      auto hash2 = hash<T2>{}(p.second);
```

```
    return hash1 ^ hash2;
  }
};

char dungeon[1001][1001];
vector<pair<int , int>> BFS(int n , int m , char starting , char ending , char block , char monster , bool
hasGun)
{
        queue<int> qx , qy;
        unordered_map<pair<int , int>, pair<int , int>, hash_pair> prev;
        int sr , sc;
        int visited[n][m];
        memset(visited , 0 , sizeof(visited));

        for(int i = 0 ; i < n ; i++)
                for(int j = 0 ; j < m ; j++)
                        if(dungeon[i][j] == starting)
                        {
                                sr = i;
                                sc = j;
                                break;
                        }

        qx.push(sr);
        qy.push(sc);

        bool reachedEnd = false;
        int countLeft = 1;
        int countNext = 0;
        int count = 0;

        int er, ec;

        while(!qx.empty())
        {
                int r = qx.front();
                int c = qy.front();
                qx.pop();
                qy.pop();

                if(dungeon[r][c] == ending)
                {
                        reachedEnd = true;
                        er = r;
                        ec = c;
                        break;
                }
```

```
            int dr[8] = {-1 , 1 , 0 , 0 , -1 ,  1 , 1 , -1};
            int dc[8] = {0 , 0 , -1 , 1 ,  1 , -1 , 1 , -1};
            for(int i = 0 ; i < 8 ; i++)
            {
                        int rr = r + dr[i];
                        int cc = c + dc[i];
                        if(rr < 0 || rr >= n || cc < 0 || cc >= m)
                                    continue;
                        if(visited[rr][cc] || dungeon[rr][cc] == block)
                                    continue;
                        if((!hasGun) && dungeon[rr][cc] == monster)
                                    continue;
                        visited[rr][cc] = 1;

                        prev[pair<int , int> (rr , cc)] = pair<int , int> (r , c);
                        qx.push(rr);
                        qy.push(cc);
                        countNext++;

            }
            countLeft--;
            if(countLeft == 0)
            {
                        countLeft = countNext;
                        countNext = 0;
                        count++;
            }
}

vector<pair<int , int>> result;

if(reachedEnd)
{
            result.push_back(pair<int , int> (count , count));
            while(count >= 1)
            {
                        result.push_back(prev[pair<int , int> (er , ec)]);
                        int temp = prev[pair<int , int> (er , ec)].first;
                        ec = prev[pair<int , int> (er , ec)].second;
                        er = temp;
                        count--;
            }

}
else
            result.push_back(pair<int , int> (-1 , -1));

return result;
```

```
}
int main()
{
	int n , m;
	cin>>n>>m;
	cout<<"\t. = free space\n\t# = block\n\tS = Starting Postion\n\tE = Destination\n\t8 =
Monster\n\t^ = Gun"<<endl<<endl;
	for(int i = 0 ; i < n ; i++)
	{
		string input;
		cin>>input;
		for(int j = 0 ; j < m ; j++)
			dungeon[i][j] = input[j];
	}

	vector<pair<int , int>> routeWithoutGun = BFS(n , m , 'S' , 'E' , '#' , '8' , false);
	vector<pair<int , int>> routeTowardsGun = BFS(n , m , 'S' , '^' , '#' , '8' , false);
	vector<pair<int , int>> routeExitWithGun = BFS(n , m , '^' , 'E' , '#' , '8' , true);



	if(routeWithoutGun[0].first < routeTowardsGun[0].first + routeExitWithGun[0].first &&
routeWithoutGun[0].first >= 1)
	{
		for(int i = 1 ; i <= routeWithoutGun[0].first ; i++)
			dungeon[routeWithoutGun[i].first][routeWithoutGun[i].second] = '*';
	}
	else
	{
		for(int i = 1 ; i <= routeTowardsGun[0].first ; i++)
			dungeon[routeTowardsGun[i].first][routeTowardsGun[i].second] = '*';
		for(int i = 1 ; i <= routeExitWithGun[0].first ; i++)
			dungeon[routeExitWithGun[i].first][routeExitWithGun[i].second] = '*';
	}
	cout<<endl;
	for(int i = 0 ; i < n ; i++)
	{
		for(int j = 0 ; j < m ; j++)
			cout<<dungeon[i][j];
		cout<<endl;
	}
}
```