**Problem 1:**

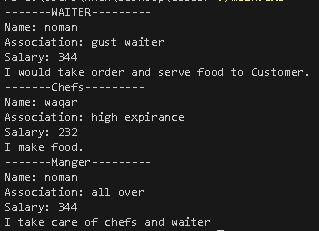
You are tasked with developing a software for a restaurant chain that manages different types of employees, such as chefs, waiters/waitresses, and managers. Each employee type has specific roles and responsibilities. How would you utilize inheritance in C++ to represent these employee types?   
Discuss how you would handle the shared characteristics and functionalities across these classes?

ANS: I would just use the constructor to initialize the value and print attribute within the constructor there would not be any getter and setter. And show within the constructor what the functionalities derived class has.

**Source code**

|  |
| --- |
| #include <iostream>  using namespace std;  class Employee  {      protected:          string name,association;          int salary;      public:      Employee(string n, string asso, int sal): name(n), association(asso), salary(sal)      {          cout << "Name: " << name <<endl;          cout << "Association: " << association <<endl;          cout << "Salary: " << salary <<endl;      }  };  class waiter : public Employee  {      public:          waiter(string n, string asso, int sal) : Employee(n,asso,sal)          {              cout << "I would take order and serve food to Customer."<<endl;          }  };  class chefs : public Employee  {      public:      chefs(string n, string asso, int sal) : Employee(n,asso,sal)      {          cout << "I make food."<<endl;      }  };  class manger : public Employee  {      public:      manger(string n, string asso, int sal) : Employee(n,asso,sal)      {          cout << "I take care of chefs and waiter"<<endl;      }  };  int main()  {      cout << "-------WAITER---------"<<endl;      waiter w("noman","gust waiter",344);      cout << "-------Chefs---------"<<endl;      chefs c("waqar","high expirance",232);      cout << "-------Manger---------"<<endl;      manger m("noman","all over",344);        return 0;  } |

**Output:**



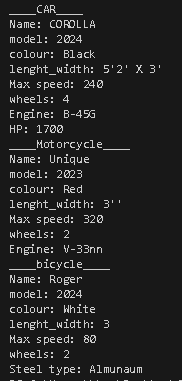
**Problem 2:**

You are building a game that involves different types of vehicles, such as cars, motorcycles, and bicycles. Each vehicle has unique characteristics and behaviours. How would you use inheritance in C++ to represent these vehicles? Provide an example of a base class and derived classes that exhibit different types of inheritance, such as single inheritance, multiple inheritance, or hierarchical inheritance.

**Source code**

|  |
| --- |
| #include <iostream>  using namespace std;  class Vehicles  {      string name,model,colour,lenght\_width;      int max\_speed,wheels;      public:  Vehicles(string n,string m,string c,string l,int s,int w):name(n), model(m),colour(c),lenght\_width(l),max\_speed(s),wheels(w) {}      void display()      {          cout << "Name: " << name <<endl;          cout << "model: " << model <<endl;          cout << "colour: " << colour <<endl;          cout << "lenght\_width: " << lenght\_width <<endl;          cout << "Max speed: " << max\_speed <<endl;          cout << "wheels: " << wheels <<endl;      }  };  class Car : public Vehicles  {      string engine;      int hp;      public:      Car(string n,string m,string c,string l,int s,int w, string e,int h) : Vehicles( n, m, c, l, s, w), engine(e), hp(h) {}      void display()      {          cout << "\_\_\_\_CAR\_\_\_\_"<<endl;          Vehicles::display();          cout << "Engine: " << engine <<endl;          cout << "HP: " << hp <<endl;      }  };  class Motorcycle : public Vehicles  {      string engine;      public:      Motorcycle(string n,string m,string c,string l,int s,int w, string e) : Vehicles( n, m, c, l, s, w), engine(e) {}      void display()      {                  cout << "\_\_\_\_Motorcycle\_\_\_\_"<<endl;          Vehicles::display();          cout << "Engine: " << engine <<endl;      }  };  class bicycle : public Vehicles  {      string steel\_type;      public:      bicycle(string n,string m,string c,string l,int s,int w, string e) : Vehicles( n, m, c, l, s, w), steel\_type(e) {}      void display()      {                  cout << "\_\_\_\_bicycle\_\_\_\_"<<endl;          Vehicles::display();          cout << "Steel type: " << steel\_type <<endl;      }  };  int main()  {      Car c("COROLLA","2024","Black","5'2' X 3' ",240,4,"B-45G",1700);      c.display();      Motorcycle m("Unique","2023","Red","3'' ",320,2,"V-33nn");      m.display();      bicycle b("Roger","2024","White","3",80,2,"Almunaum");      b.display();      return 0;  } |

**Output:**

****

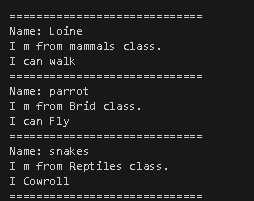
**Problem 3:**

You are working on a simulation program for a zoo that contains various animals. Each animal belongs to a specific species and exhibits different behaviors. How would you use inheritance in C++ to model the relationship between the species and animals? Provide an example of how you would implement inheritance to represent these relationships and define the common attributes and methods.

**Source code**

|  |
| --- |
| #include <iostream>  using namespace std;  class Animmals  {      protected:      string name;      public:      Animmals(string n): name(n) {}      void display()      {          cout << "Name: "<<name<<endl;      }  };  class Mammals : public Animmals  {      public:      Mammals(string n) : Animmals(n)      {          Animmals::display();          cout << "I m from mammals class."<<endl;          cout << "I can walk"<<endl;      }  };  class Brid : public Animmals  {      public:      Brid(string n) : Animmals(n)      {          Animmals::display();          cout << "I m from Brid class."<<endl;          cout << "I can Fly"<<endl;      }  };  class Reptiles : public Animmals  {      public:      Reptiles(string n) : Animmals(n)      {          Animmals::display();          cout << "I m from Reptiles class."<<endl;          cout << "I Cowroll"<<endl;      }  };  int main()  {      cout << "============================="<<endl;      Mammals m("Loine");      cout << "============================="<<endl;      Brid b("parrot");      cout << "============================="<<endl;      Reptiles("snakes");      cout << "============================="<<endl;      return 0;  } |

**Output:**

****

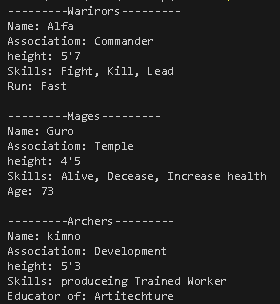
**Problem 4:**

Consider a scenario where you are developing a game that involves different characters, such as warriors, mages, and archers. How would you utilize hierarchical inheritance in C++ to represent these character types and their shared attributes? Provide an example of a base class and derived classes to illustrate the concept.

**Source code:**

|  |
| --- |
| #include <iostream>  using namespace std;  class Character  {      protected:      string name,association,height,skill;      public:      Character(string n,string asso,string h,string s): name(n),association(asso), height(h),skill(s) {}  };  class Warrirors : public Character  {      string run;      public:          Warrirors(string n,string asso,string h,string s,string r): run(r), Character(n,asso,h,s) {}          void display()          {              cout << "---------Warirors---------"<<endl;              cout << "Name: " << name <<endl;              cout << "Associatiom: " << association <<endl;              cout << "height: " << height <<endl;              cout << "Skills: " << skill <<endl;              cout << "Run: " << run <<endl;            }  };  class Mages : public Character  {      string old;      public:          Mages(string n,string asso,string h,string s,string o): old(o), Character(n,asso,h,s) {}          void display()          {              cout << "---------Mages---------"<<endl;              cout << "Name: " << name <<endl;              cout << "Associatiom: " << association <<endl;              cout << "height: " << height <<endl;              cout << "Skills: " << skill <<endl;              cout << "Age: " << old <<endl;            }  };  class Archers : public Character  {      string edu;      public:          Archers(string n,string asso,string h,string s,string e): edu(e), Character(n,asso,h,s) {}          void display()          {              cout << "---------Archers---------"<<endl;              cout << "Name: " << name <<endl;              cout << "Associatiom: " << association <<endl;              cout << "height: " << height <<endl;              cout << "Skills: " << skill <<endl;              cout << "Educator of: " << edu <<endl;            }  };  int main()  {      Warrirors w("Alfa","Commander","5'7","Fight, Kill, Lead", "Fast");      w.display();      cout << "\n";      Mages m("Guro","Temple","4'5","Alive, Decease, Increase health","73");      m.display();      cout << "\n";      Archers a("kimno","Development","5'3","produceing Trained Worker", "Artitechture");      a.display();      return 0;  } |

**Output**

****

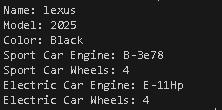
**Problem 5:**

In the context of a vehicle manufacturing company, explain how multiple inheritance can be applied to represent different vehicle models that have both specific features (e.g., sports car, electric car) and common characteristics (e.g., engine, wheels). Discuss the advantages and challenges of using multiple inheritance in this scenario

**Source code:**

|  |
| --- |
| #include <iostream>  using namespace std;  class Sport\_car  {  protected:      string engine;      int wheels;  };  class Electric\_car  {  protected:      string engine;      int wheels;  };  class Vehical : public Sport\_car, public Electric\_car  {      string name;      string model;      string color;  public:      Vehical() {}      Vehical(string n, string m, string c, string se, int sw, string ee, int ew) : name(n), model(m), color(c)      {          Electric\_car::engine = ee;          Electric\_car::wheels = ew;          Sport\_car ::engine = se;          Sport\_car ::wheels = sw;          // engine = se;      }      void display()      {          cout << "Name: " << name << endl;          cout << "Model: " << model << endl;          cout << "Color: " << color << endl;          cout << "Sport Car Engine: " << Sport\_car::engine << endl;          cout << "Sport Car Wheels: " << Sport\_car::wheels << endl;          cout << "Electric Car Engine: " << Electric\_car::engine << endl;          cout << "Electric Car Wheels: " << Electric\_car::wheels << endl;      }  };  int main()  {      Vehical V;      V = Vehical("lexus", "2025", "Black", "B-3e78", 4, "E-11Hp", 4);      V.display();      return 0;  } |

**Output:**

****

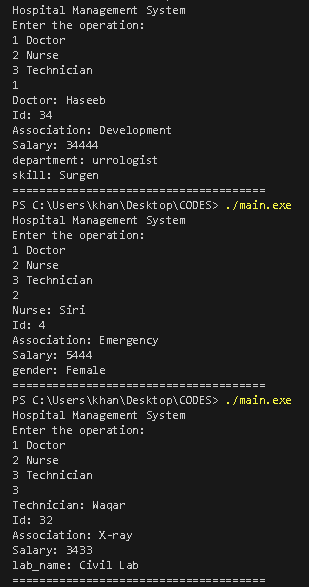
**Problem 6:**

Imagine you are building a software for a hospital that manages different types of medical professionals, including doctors, nurses, and technicians. How can you use virtual inheritance in C++ to handle cases where a class needs to inherit from multiple base classes with a shared base class, such as a hospital employee? Explain the benefits and considerations of using virtual inheritance in this context.

**Source code;**

|  |
| --- |
| #include <iostream>  using namespace std;  class Hospital\_employee  {  protected:      string name;      string association;      int salary;      int id;  public:      Hospital\_employee(string n, string asso, int sal, int ids) : name(n), association(asso), salary(sal), id(ids) {}  };  class Doctor : virtual public Hospital\_employee  {  protected:      string department;      string skill;  public:      Doctor(string n, string asso, int sal, int id, string dept, string skll) : Hospital\_employee(n, asso, sal, id), department(dept), skill(skll) {}      void display()      {          cout << "Doctor: " << name << endl;          cout << "Id: " << id << endl;          cout << "Association: " << association << endl;          cout << "Salary: " << salary << endl;          cout << "department: " << department << endl;          cout << "skill: " << skill << endl;      }  };  class Nurse : virtual public Hospital\_employee  {  protected:      string gender;  public:      Nurse(string n, string asso, int sal, int id, string gen) : Hospital\_employee(n, asso, sal, id), gender(gen) {}      void display()      {          cout << "Nurse: " << name << endl;          cout << "Id: " << id << endl;          cout << "Association: " << association << endl;          cout << "Salary: " << salary << endl;          cout << "gender: " << gender << endl;      }  };  class Technician : virtual public Hospital\_employee  {  protected:      string lab\_name;  public:      Technician(string n, string asso, int sal, int id, string lab) : Hospital\_employee(n, asso, sal, id), lab\_name(lab) {}      void display()      {          cout << "Technician: " << name << endl;          cout << "Id: " << id << endl;          cout << "Association: " << association << endl;          cout << "Salary: " << salary << endl;          cout << "lab\_name: " << lab\_name << endl;      }  };  class Management  {  public:      Management()      {          cout << "Hospital Management System" << endl;          cout << "Enter the operation:" << endl;          cout << "1 Doctor" << endl;          cout << "2 Nurse" << endl;          cout << "3 Technician" << endl;          int s;          cin >> s;          switch (s)          {          case 1:          {              Doctor doc("Haseeb", "Development", 34444, 34, "urrologist", "Surgen");              doc.display();              cout << "======================================" << endl;              break;          }          case 2:          {              Nurse nurse("Siri", "Emergency", 5444, 4, "Female");              nurse.display();              cout << "======================================" << endl;              break;          }          case 3:          {              Technician tech("Waqar", "X-ray", 3433, 32, "Civil Lab");              tech.display();              cout << "======================================" << endl;              break;          }          default:              cout << "Invalid" << endl;              break;          }      }  };  int main()  {      Management management;      return 0;  } |

**Output:**

****

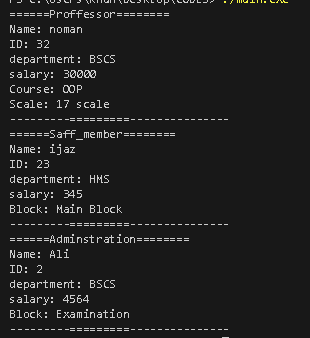
**Problem 7:**

Create a class hierarchy for a university management system that includes different types of employees, such as professors, staff members, and administrators. Implement appropriate member functions and attributes for each employee type, such as salary calculation and department assignment. Utilize inheritance to handle the common functionalities and attributes among the employee types.

**Source code:**

|  |
| --- |
| #include <iostream>  using namespace std;  class Employee  {  protected:      string name;      int ID;      string department;      double salary;  public:      Employee(string n, int ID, string d, double sal) : name(n), ID(ID), department(d), salary(sal) {}      void display()      {          cout << "Name: " << name << endl;          cout << "ID: " << ID << endl;          cout << "department: " << department << endl;          cout << "salary: " << salary << endl;      }  };  class Proffessor: private Employee  {      string course;      string scale;      public:      Proffessor(string n, int ID, string d, double sal,string crse,string s) : Employee( n,  ID,  d, sal), course(crse),scale(s){}      void display()      {          cout << "======Proffessor========"<<endl;          Employee::display();          cout << "Course: "<<course<<endl;          cout << "Scale: "<<scale<<endl;      }  };  class Saff\_member: private Employee  {      string block;      public:      Saff\_member(string n, int ID, string d, double sal,string b) : Employee( n,  ID,  d, sal), block(b) {}      void display()      {          cout << "======Saff\_member========"<<endl;          Employee::display();          cout << "Block: "<<block<<endl;      }  };  class Adminstration: private Employee  {      string block;      public:      Adminstration(string n, int ID, string d, double sal,string b) : Employee( n,  ID,  d, sal), block(b) {}      void display()      {          cout << "======Adminstration========"<<endl;          Employee::display();          cout << "Block: "<<block<<endl;      }  };  int main()  {      Proffessor p("noman",32,"BSCS",30000,"OOP","17 scale");      p.display();      cout << "---------=========---------------\n";      Saff\_member s("ijaz",23,"HMS",345,"Main Block");      s.display();      cout << "---------=========---------------\n";      Adminstration a("Ali",2,"BSCS",4564,"Examination");      a.display();      cout << "---------=========---------------";      return 0;  } |

**Output:**

****

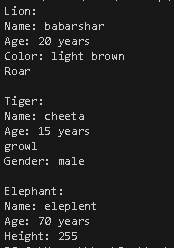
**Problem 8:**

Develop a simulation program for a zoo that contains various animals. Create a base class called Animal with attributes such as name, age, and habitat. Implement derived classes for different animal species, such as lions, tigers, and elephants. Include unique member functions and attributes for each species, such as feeding habits or special abilities. Use inheritance to capture the shared characteristics and behaviors among the animal classes.

**Source code:**

|  |
| --- |
| #include <iostream>  #include <string>  using namespace std;  class Animal  {  protected:      string name;      int age;  public:      Animal(string n, int a)          : name(n), age(a) {}      void display()      {          cout << "Name: " << name << endl;          cout << "Age: " << age << " years" << endl;      }  };  class Lion : public Animal  {  private:      string color;  public:      Lion(string n, int a, string c) : Animal(n, a), color(c) {}      void display()      {          cout << "Lion:" << endl;          Animal::display();          cout << "Color: " << color << endl;          cout << "Roar" << endl;      }  };  class Tiger : public Animal  {  private:      string gender;  public:      Tiger(string n, int a, string g) : Animal(n, a), gender(g) {}      void display()      {          cout << "Tiger:" << endl;          Animal::display();          cout << "growl" << endl;          cout << "Gender: " << gender << endl;      }  };  class Elephant : public Animal  {  private:      int height;  public:      Elephant(string n, int a, int h) : Animal(n, a), height(h) {}      void display()      {          cout << "Elephant:" << endl;          Animal::display();          cout << "Height: " << height << endl;      }  };  int main()  {      Lion lion("babarshar", 20, "light brown");      Tiger tiger("cheeta", 15, "male");      Elephant elephant("eleplent", 70, 255);      lion.display();      cout << endl;      tiger.display();      cout << endl;      elephant.display();      return 0;  } |

**Output:**

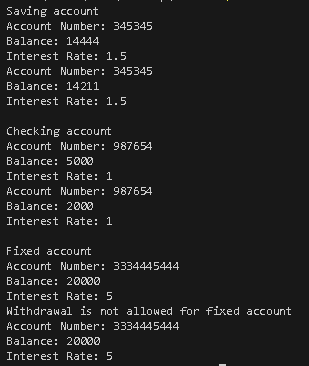
****

**Problem 9:**

**Source code:**

|  |
| --- |
| #include <iostream>  #include <string>  using namespace std;  class Account  {  protected:      string accountNumber;      double balance;      double interestRate;  public:      Account(string anum, double b, double intrate)          : accountNumber(anum), balance(b), interestRate(intrate) {}      void deposit(double amount)      {          balance += amount;      }      void withdraw(double amount)      {          if (amount <= balance)          {              balance -= amount;          }          else          {              cout << "Insufficient balance" << endl;          }      }      void calculateInterest()      {          balance += balance \* (interestRate / 100.0);      }      void display()      {          cout << "Account Number: " << accountNumber << endl;          cout << "Balance: " << balance << endl;          cout << "Interest Rate: " << interestRate << endl;      }  };  class SavingsAccount : public Account  {  public:      SavingsAccount(string anum, double b, double intrate)          : Account(anum, b, intrate) {}      void withdraw(double amount)      {          if (amount <= balance)          {              balance -= amount;          }          else          {              cout << "Insufficient balance" << endl;          }      }  };  class CheckingAccount : public Account  {  public:      CheckingAccount(string anum, double b, double intrate)          : Account(anum, b, intrate) {}      void withdraw(double amount)      {          if (amount <= balance)          {              balance -= amount;          }          else          {              cout << "Insufficient balance" << endl;          }      }  };  class FixedDepositAccount : public Account  {  public:      FixedDepositAccount(string anum, double b, double intrate)          : Account(anum, b, intrate) {}      void withdraw(double amount)      {          cout << "Withdrawal is not allowed for fixed account" << endl;      }  };  int main()  {      SavingsAccount savings("345345", 14444, 1.5);      CheckingAccount checking("987654", 5000, 1.0);      FixedDepositAccount fixed("3334445444", 20000, 5.0);      cout << "Saving account" << endl;      savings.display();      savings.withdraw(233);      savings.display();      cout << endl;      cout << "Checking account" << endl;      checking.display();      checking.withdraw(3000);      checking.display();      cout << endl;      cout << "Fixed account" << endl;      fixed.display();      fixed.withdraw(5000);      fixed.display();      return 0;  } |

**Output**

****

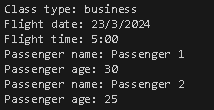
**Problem 10:**

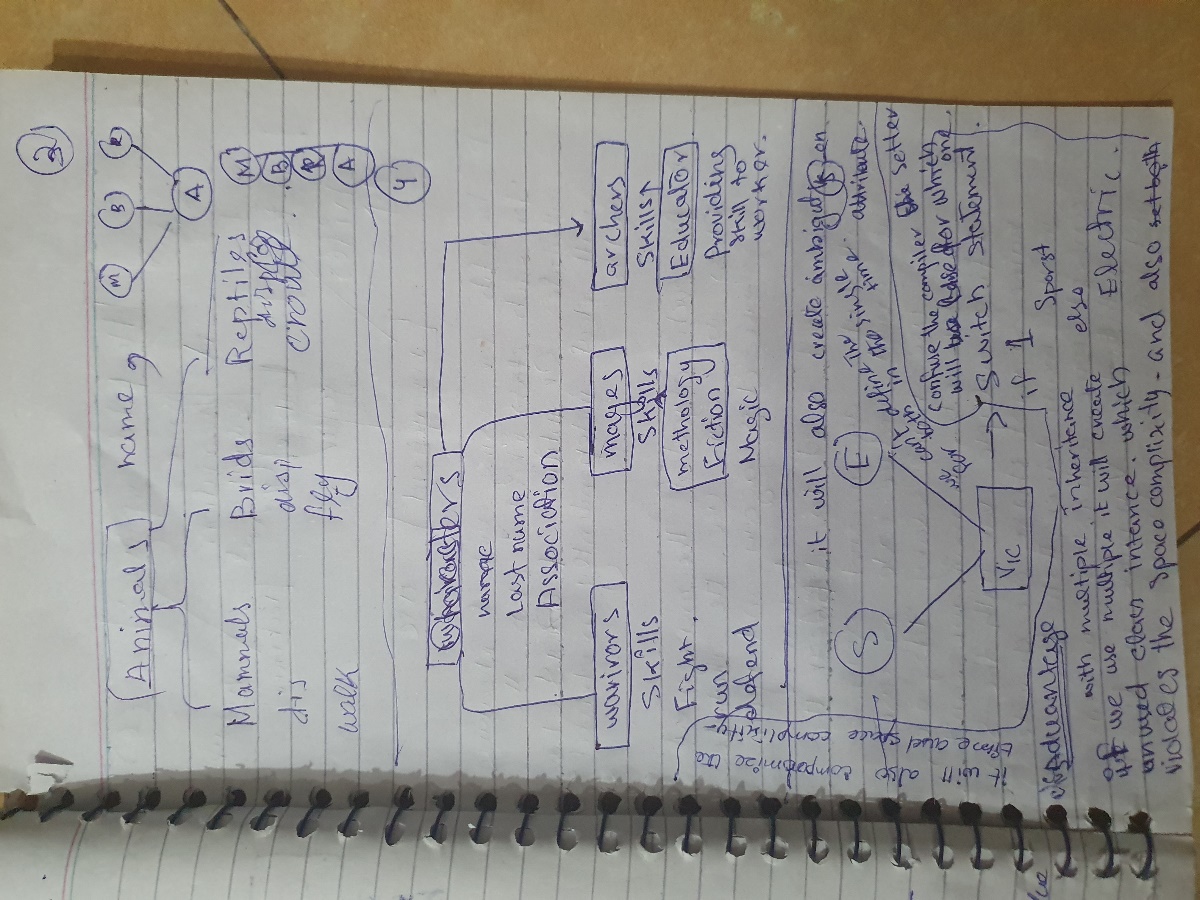
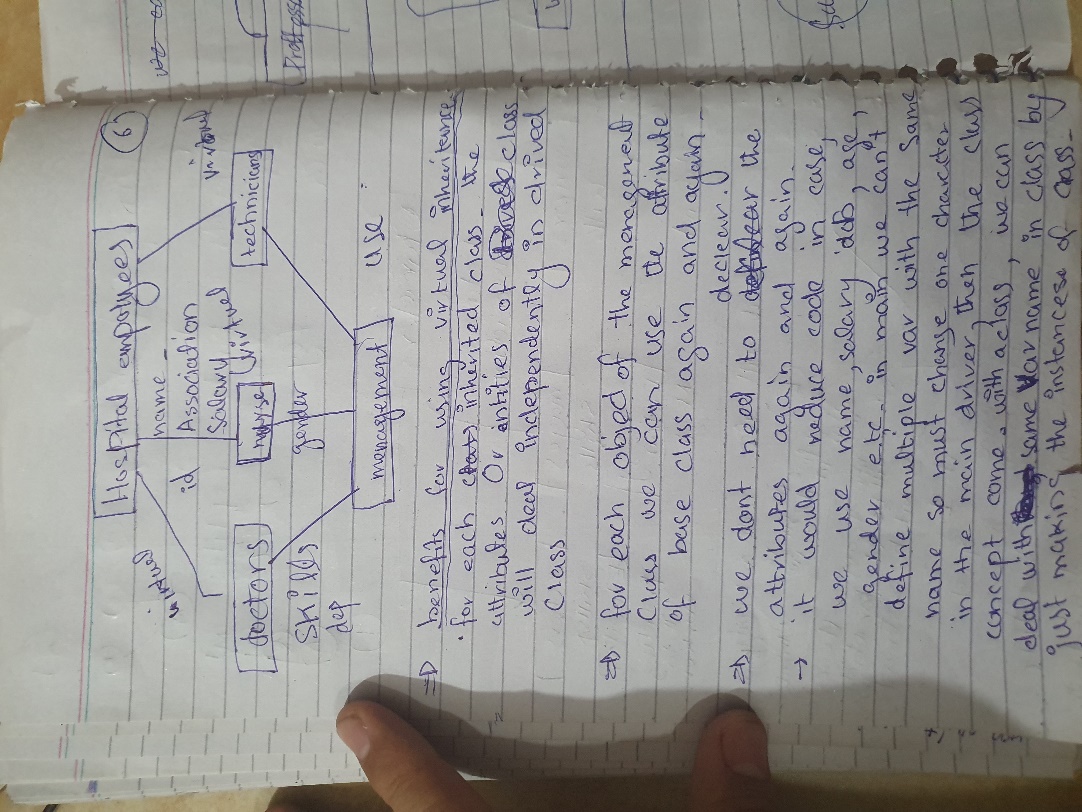
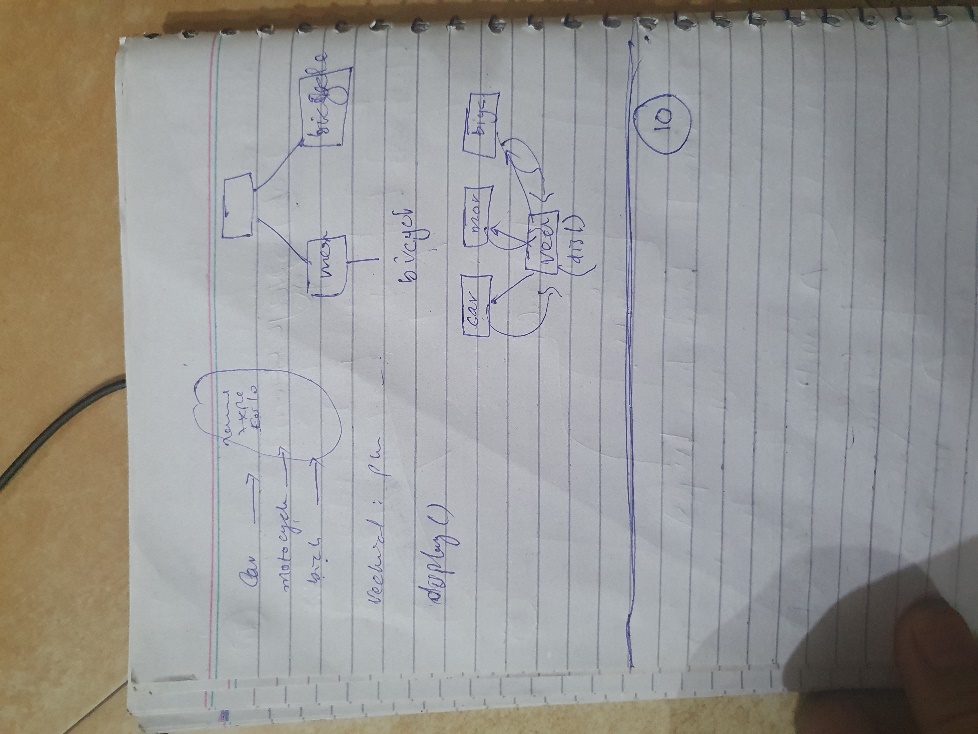
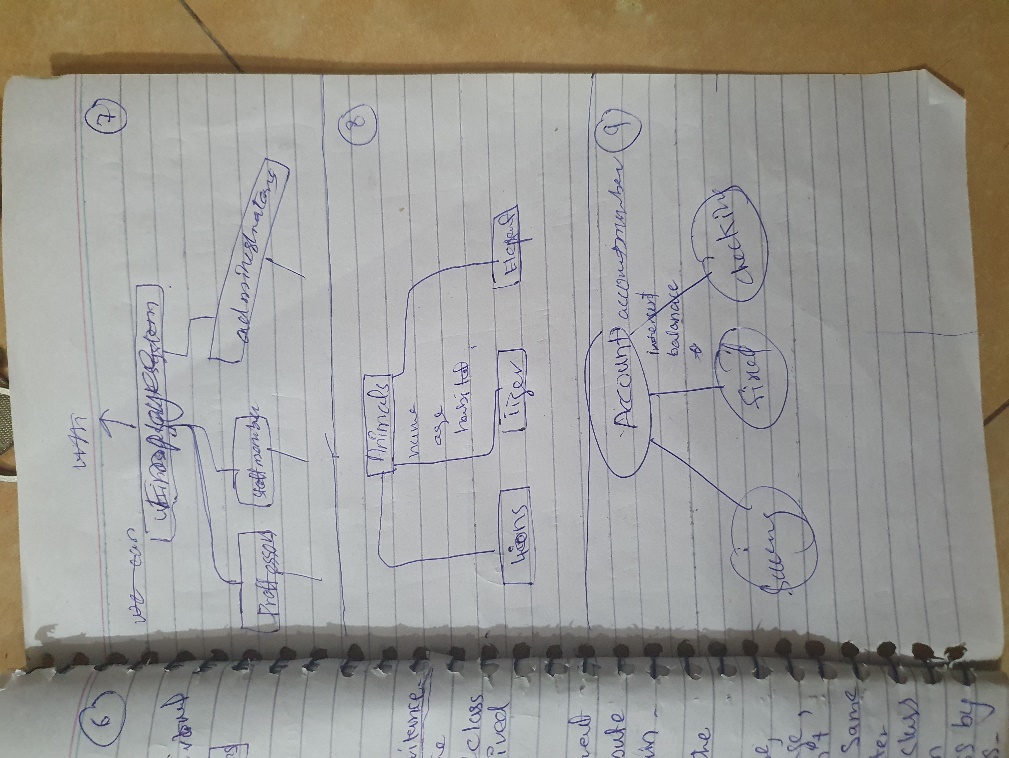
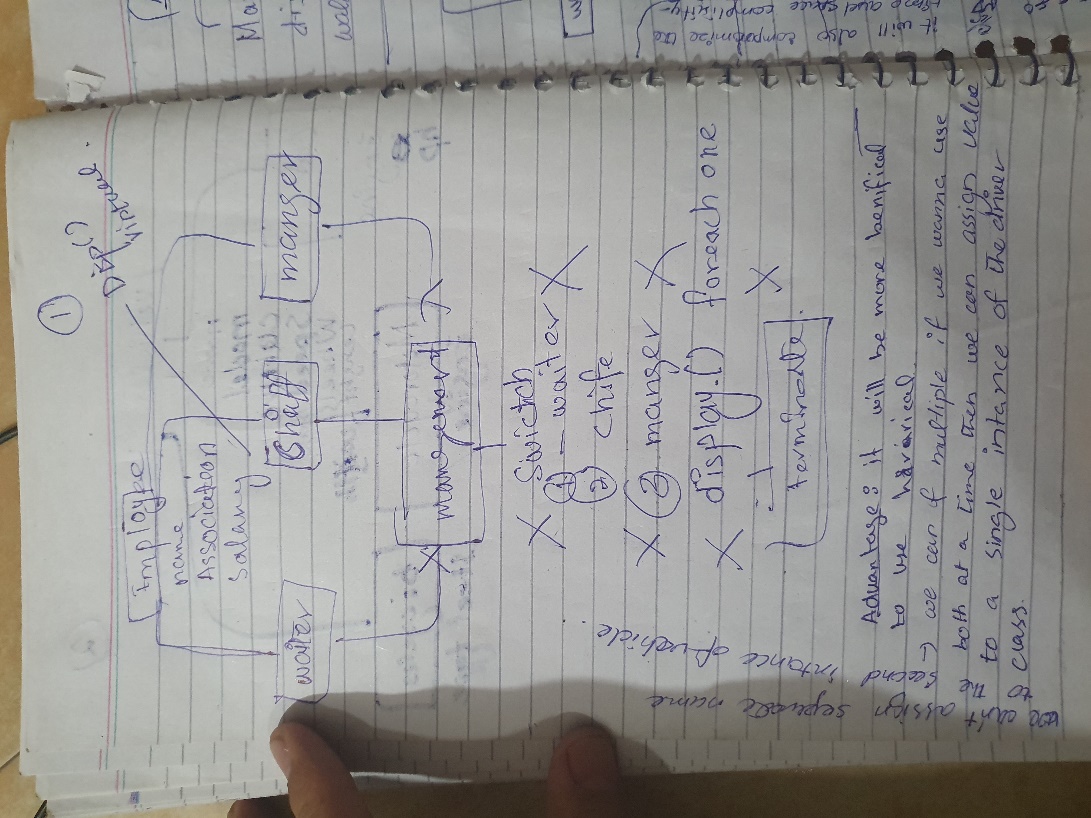
Implement a comprehensive airline reservation system that handles various aspects, such as flight scheduling, passenger bookings, and ticket management. Design a class hierarchy that represents different entities, such as flights, passengers, and tickets, with advanced features such as seat allocation, passenger preferences, and payment processing. Utilize inheritance to manage the shared functionalities and attributes among the entities while maintaining flexibility and scalability

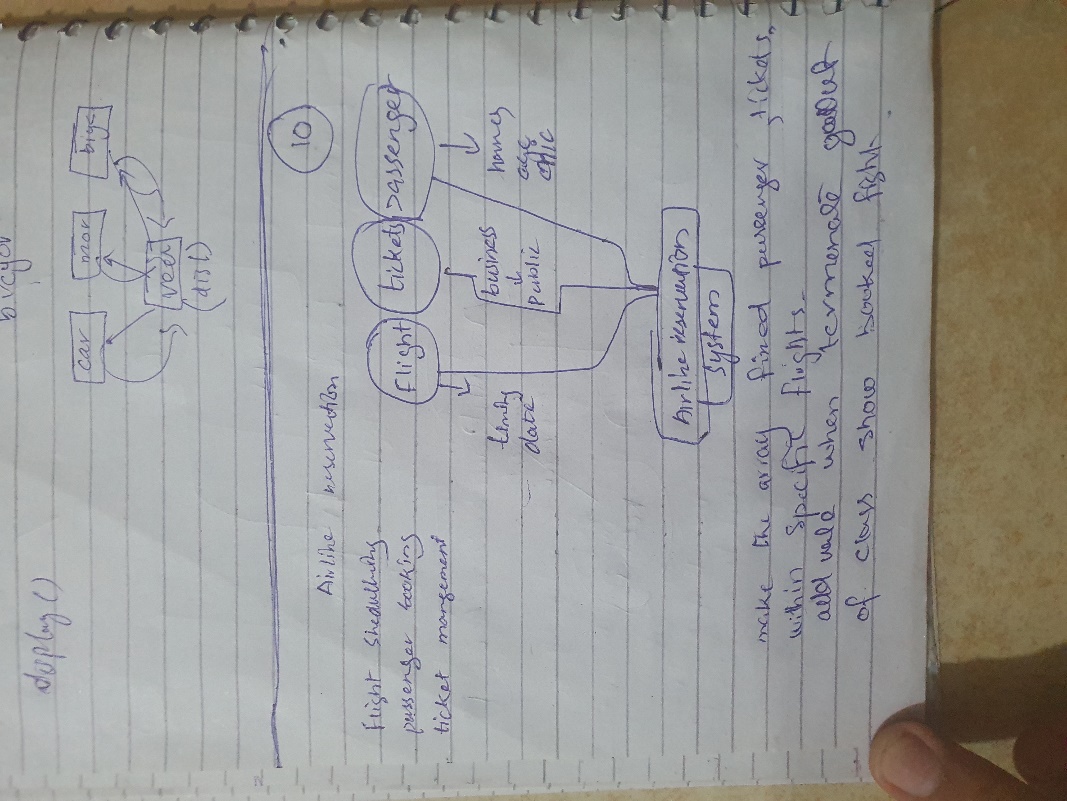
**Source code:**

|  |
| --- |
| #include <iostream>  using namespace std;  class Flight  {  private:      string time;      string date;  public:      Flight(string t, string d) : time(t), date(d) {}      void display()      {          cout << "Flight date: " << date << endl;          cout << "Flight time: " << time << endl;      }  };  class Passenger  {  private:      string name;      int age;  public:      Passenger(string n, int a) : name(n), age(a) {}      void display()      {          cout << "Passenger name: " << name << endl;          cout << "Passenger age: " << age << endl;      }  };  class Ticket  {  private:      string class\_type;      Flight flight;      Passenger \*passengers[10];      int Count;  public:      Ticket(string type, Flight f) : class\_type(type), flight(f), Count(0) {}      void addPassenger(Passenger &p)      {          if (Count < 10)          {              passengers[Count++] = &p;          }          else          {              cout << "Ticket is full" << endl;          }      }      void display()      {          cout << "Class type: " << class\_type << endl;          flight.display();          for (int i = 0; i < Count; ++i)          {              passengers[i]->display();          }      }  };  int main()  {      Flight flight1("5:00", "23/3/2024");      Passenger p1("haseeb khan", 30);      Passenger p2("faheem", 25);      Ticket ticket1("business", flight1);      ticket1.addPassenger(p1);      ticket1.addPassenger(p2);      ticket1.display();      return 0;  } |

**Output:**

****

****

****