**Name Haseeb Ullah**

**ID: F20232661009**

**Problem Statement 1:**

Implement a base class called Animal with a virtual function makeSound(). Derive three classes

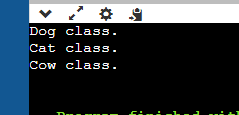
from Animal: Dog, Cat, and Cow. Override the makeSound() function in each derived class to

output the sound of the respective animal.

**Source code:**

|  |
| --- |
| // Problem Statement 1:  // Implement a base class called Animal with a virtual function makeSound(). Derive three classes  // from Animal: Dog, Cat, and Cow. Override the makeSound() function in each derived class to  // output the sound of the respective animal.  #include <iostream>  using namespace std;  class Animal  {  public:  virtual void sound()  {  cout << "Animnal class."<<endl;  }  };  class Dog : public Animal  {  public:  void sound()  {  cout << "Dog class."<<endl;  }  };  class Cat: public Animal  {  public:  void sound()  {  cout << "Cat class."<<endl;  }  };  class Cow: public Animal  {  public:  void sound()  {  cout << "Cow class."<<endl;  }  };  int main()  {  Animal\* animal;  Dog dog;  Cat cat;  Cow cow;  animal = &dog;  animal->sound();  animal = &cat;  animal->sound();  animal = &cow;  animal->sound();  return 0;  } |

**Output:**



**Problem Statement 2:**

Create a base class called Vehicle with pure virtual functions startEngine() and stopEngine().

Derive two classes from Vehicle: Car and Motorcycle. Implement the startEngine() and

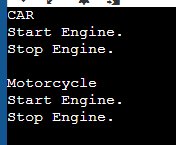
stopEngine() functions in each derived class to provide the appropriate actions for each type of

vehicle.

**Source code:**

|  |
| --- |
| // // Problem Statement 2:  // // Create a base class called Vehicle with pure virtual functions startEngine() and stopEngine().  // // Derive two classes from Vehicle: Car and Motorcycle. Implement the startEngine() and  // // stopEngine() functions in each derived class to provide the appropriate actions for each type of  // // vehicle.  #include <iostream>  using namespace std;  class Vehicle  {  public:  virtual void startEngine() = 0;  virtual void stopEngine() = 0;    };  class Car : public Vehicle  {  public:  void startEngine()  {  cout << "Start Engine."<<endl;  }  void stopEngine()  {  cout << "Stop Engine."<<endl;  }  };  class Motorcycle: public Vehicle  {  public:  void startEngine()  {  cout << "Start Engine."<<endl;  }  void stopEngine()  {  cout << "Stop Engine."<<endl;  }  };  int main()  {  Vehicle\* vehicle;  Car car;  Motorcycle motorcycle;  vehicle = &car;  cout << "CAR" <<endl;  car.startEngine();  car.stopEngine();  cout << " "<<endl;  vehicle = &motorcycle;  cout << "Motorcycle" <<endl;  motorcycle.startEngine();  motorcycle.stopEngine();  return 0;  } |

**Output:**



**Problem Statement 3:**

Design a base class called Employee with virtual functions calculateSalary() and

displayInformation(). Derive two classes from Employee: Manager and Engineer. Override the

calculateSalary() function in each derived class to calculate the salary based on different criteria,

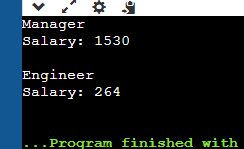
and implement the displayInformation() function to output the relevant information for each

employee type.’

**Source code:**

|  |
| --- |
| // Problem Statement 3:  // Design a base class called Employee with virtual functions calculateSalary() and  // displayInformation(). Derive two classes from Employee: Manager and Engineer. Override the  // calculateSalary() function in each derived class to calculate the salary based on different criteria,  // and implement the displayInformation() function to output the relevant information for each  // employee type.  #include <iostream>  using namespace std;  class Employee  {  protected:  string name;  int workinghours;  int salaryperhour;    virtual int calculateSalary()  {  return workinghours \* salaryperhour;  }  public:  Employee(string n, int wh, int salph): name(n), workinghours(wh),salaryperhour(salph) {}    virtual void displayInformation()  {  cout << "Salary: " << calculateSalary() << endl;  }  };  class Manager: public Employee  {  int calculateSalary()  {  return workinghours \* salaryperhour;  }  public:  Manager(string n, int wh, int salph): Employee(n,wh,salph) {}  void displayInformation()  {  cout << "Manager"<<endl;  cout << "Salary: " << calculateSalary() << endl;  }  };  class Engineer: public Employee  {  int calculateSalary()  {  return workinghours \* salaryperhour;  }  public:  Engineer(string n, int wh, int salph): Employee(n,wh,salph) {}    void displayInformation()  {  cout << "Engineer"<<endl;  cout << "Salary: " << calculateSalary() << endl;  }  };  int main()  {  Employee\* emp;  Manager m("haseeb", 45, 34);  emp = &m;  emp->displayInformation();  cout << " "<<endl;  Engineer e("rafy", 66, 4);  emp = &e;  emp->displayInformation();    delete emp;    return 0;  } |

**Output**



**Problem Statement 4:**

Create a base class called Shape with a pure virtual function draw(). Derive two classes from

Shape: Rectangle and Circle. Implement the draw() function in each derived class to display the

respective shape using ASCII characters.

**Source code:**

|  |
| --- |
| // Problem Statement 4:  // Create a base class called Shape with a pure virtual function draw(). Derive two classes from  // Shape: Rectangle and Circle. Implement the draw() function in each derived class to display the  // respective shape using ASCII characters.  #include <iostream>  using namespace std;  class Shape  {  protected:  int height;  int weight;  double radius;    public:  Shape(int h,int w) : height(h), weight(w) {}  Shape(int r) : radius(r) {}  virtual char draw() = 0;  };  class Rectangle: public Shape  {  public:  Rectangle(int h,int w): Shape(h,w) {}  char draw()  {  char ch = height \* weight;  return ch;  }  };  class Circle: public Shape  {  public:  Circle(int r): Shape(r) {}  char draw()  {  char ch = (3.14) \* radius \* radius;  return ch;  }  };  int main()  {  Shape\* shape;  Rectangle m(45, 34);  shape = &m ;  cout << shape->draw();  cout << " "<<endl;  Circle e(6);  shape = &e;  cout << shape->draw();    return 0;  } |

**Output:**

