Object Oriented Programming

Assignment # 02

Muhammad Shakaib Arsalan

Student ID: F2022266626

Course Code: CC1022

Section: Spring 2023

Section: V2

Resource Person: Rehan Raza

School of Systems and Technology

UMT Lahore Pakistan

Task 01

Overview

The code requied classes such as Employee, Chef, Waiter, and Manager. Here's a brief overview of each class:

1. **Employee:** This class represents a generic employee in the restaurant chain. It contains common attributes such as id, name, working_hours, and hourly_wage. It also provides methods to retrieve employee information and display it.

2. **Chef:** This class inherits from Employee and represents a chef in the restaurant. It adds specific attributes like dish1, dish2, and employment_since. It includes additional methods for displaying chef-specific duties and information.

3. **Waiter:** This class also inherits from Employee and represents a waiter/waitress in the restaurant. It introduces attributes like table and employment_since. It includes methods for displaying waiter-specific duties and information.

4. **Manager:** This class inherits from Employee and represents a manager in the restaurant. It extends the Employee class by adding emp (number of employees under the manager) and depart (department) attributes. It includes methods for displaying manager-specific duties and information.

Inheritance Usage

The code effectively utilizes inheritance in the following manner:

Each specialized employee type (chef, waiter, and manager) inherits from the Employee class, allowing them to access and utilize the common attributes and methods defined in the base class.
By inheriting from Employee, the specialized classes automatically have attributes like id, name, working_hours, and hourly_wage without having to redefine them.
The specialized classes can then add their own specific attributes and methods as needed, making it easier to represent the unique characteristics and responsibilities of each employee type.

```
#include <iostream>

using namespace std;

class Employee
{
private:
    int id;
    string name;
    int working_hours;
    float hourly_wage;

public:
    Employee(int id, string name, int working_hours, float hourly_wage);
    void employee_display();
    string getname();
    int getWorkingHours();
    float getHourlyWage();
    ~Employee();
};
```

```
Employee::Employee(int id, string name, int working_hours, float hourly_wage)
{
    this->id = id;
    this->name = name;
    this->working_hours = working_hours;
    this->hourly_wage = hourly_wage;
}
void Employee::employee_display()
{
    cout << "ID: " << id << "\n";
    cout << "Name: " << name << "\n";
}
string Employee ::getname()
{
    return name;
}
int Employee::getWorkingHours()
{
    return working_hours;
}
float Employee::getHourlyWage()
{
    return hourly_wage;
}
Employee::~Employee() {}

class Chef : protected Employee
{
private:
    string dish1;
    string dish2;
    int employment_since;

public:
    Chef(int id, string name, int working_hours, float hourly_wage, int
employment_since, string d1, string d2);
    void chef_display();
    void chef_duties();
    ~Chef();
};
Chef::Chef(int id, string name, int working_hours, float hourly_wage, int
employment_since, string dish1, string dish2) : Employee(id, name, working_hours,
hourly_wage)
{
    this->employment_since = employment_since;
    this->dish1 = dish1;
    this->dish2 = dish2;
}
void Chef::chef_duties()
{
    cout << "Duties of Chef:\n";
    cout << "  1. Menu Planning\n  2. Food Preparation\n  3. Kitchen Management\n
4. Cooking\n  5. Quality Control\n";
}
void Chef::chef_display()
{
    cout << "<><><><><><><> Chef <><><><><><><>\n";
    employee_display();
    cout << "Special Dishes: " << dish1 << ", " << dish2 << "\n";
    cout << "Working Hours: " << getWorkingHours() << "\n";
```

```
    cout << "Hourly Wage: " << getHourlyWage() << "\n";
    cout << "Total Salary: " << getWorkingHours() * getHourlyWage() << "\n";
    cout << "Employment Since: " << employment_since << "\n";
    chef_duties();
}
Chef::~Chef() {}

class Waiter : protected Employee
{
private:
    int table;
    int employment_since;

public:
    Waiter(int id, string name, int working_hours, float hourly_wage, int
employment_since, int table);
    void waiter_display();
    void waiter_duties();
    ~Waiter();
};
Waiter::Waiter(int id, string name, int working_hours, float hourly_wage, int
employment_since, int table) : Employee(id, name, working_hours, hourly_wage)
{
    this->table = table;
    this->employment_since = employment_since;
}
void Waiter::waiter_duties()
{
    cout << "Duties of Waiter:\n";
    cout << "  1. Taking orders\n  2. Serving food and beverages\n  3. Assisting
customers\n";
}
void Waiter::waiter_display()
{
    cout << "<><><><><><><> Waiter <><><><><><><>\n";
    employee_display();
    cout << "Table Served by " << getname() << ": " << table << "\n";
    cout << "Working Hours: " << getWorkingHours() << "\n";
    cout << "Hourly Wage: " << getHourlyWage() << "\n";
    cout << "Total Salary: " << getWorkingHours() * getHourlyWage() << "\n";
    cout << "Employment Since: " << employment_since << "\n";
    waiter_duties();
}
Waiter::~Waiter() {}

class Manager : protected Employee
{
private:
    int emp;
    string depart;
    int employment_since;

public:
    Manager(int id, string name, int working_hours, float hourly_wage, int
employment_since, int emp, string depart);
    void manager_display();
    void manager_duties();
    ~Manager();
};
```

```
Manager::Manager(int id, string name, int working_hours, float hourly_wage, int
employment_since, int emp, string depart) : Employee(id, name, working_hours,
hourly_wage)
{
    this->emp = emp;
    this->depart = depart;
    this->employment_since = employment_since;
}
void Manager::manager_duties()
{
    cout << "Duties of Manager:\n";
    cout << "  1. Staff supervision and training\n  2. Budget management\n  3.
Customer service\n";
}
void Manager::manager_display()
{
    cout << "<><><><><><><> Manager <><><><><><><><>\n";
    employee_display();
    cout << "Department: " << depart << "\n";
    cout << "Employee's Under " << getname() << ": " << emp << "\n";
    cout << "Working Hours: " << getWorkingHours() << "\n";
    cout << "Hourly Wage: " << getHourlyWage() << "\n";
    cout << "Total Salary: " << getWorkingHours() * getHourlyWage() << "\n";
    cout << "Employment Since: " << employment_since << "\n";
    manager_duties();
}
Manager::~Manager() {}

int main()
{
    Chef c1(123, "Eman", 5, 1500, 2002, "Stack", "Pizza");
    c1.chef_display();
    cout << endl;

    Waiter w1(456, "Umair", 6, 2000, 2001, 7);
    w1.waiter_display();
    cout << endl;

    Manager m1(789, "Arman", 4, 3000, 2008, 12, "Back of House");
    m1.manager_display();
    cout << endl;

    return 0;

}
```

```
<><><><><><><> Chef <><><><><><><>
ID: 123
Name: Eman
Special Dishes: Stack, Pizza
Working Hours: 5
Hourly Wage: 1500
Total Salary: 7500
Employment Since: 2002
Duties of Chef:
  1. Menu Planning
  2. Food Preparation
  3. Kitchen Management
  4. Cooking
  5. Quality Control
```

```
<><><><><><><> Waiter <><><><><><><>
ID: 456
Name: Umair
Table Served by Umair: 7
Working Hours: 6
Hourly Wage: 2000
Total Salary: 12000
Employment Since: 2001
Duties of Waiter:
  1. Taking orders
  2. Serving food and beverages
  3. Assisting customers
```

```
<><><><><><><> Manager <><><><><><><>
ID: 789
Name: Arman
Department: Back of House
Employee's Under Arman: 12
Working Hours: 4
Hourly Wage: 3000
Total Salary: 12000
Employment Since: 2008
Duties of Manager:
  1. Staff supervision and training
  2. Budget management
  3. Customer service
```

Task 02

The code requied classes such as:

1. Vehicle:
   The base class "vehicle" represents common characteristics shared by all vehicles. It has private member variables such as year, make, model, and color. It also provides a constructor to initialize these variables, a display() function to output the vehicle details, and a destructor.

2. Car:
   The derived class "car" inherits from the base class "vehicle" using single inheritance. It adds additional private member variables such as num_door, variant, top_speed, and tank. It provides a constructor to initialize these variables by calling the base class constructor, a c_display() function to display car-specific details along with vehicle details, and a destructor.

3. Motorcycle:
   The derived class "motorcycle" also inherits from the base class "vehicle" using single inheritance. It adds private member variables such as type, fuel, and min. It provides a constructor to initialize these variables by calling the base class constructor, an m_display() function to display motorcycle-specific details along with vehicle details, and a destructor.

4. Bicycle:
   The derived class "bicycle" also inherits from the base class "vehicle" using single inheritance. It adds a private member variable called manufacturer. It provides a constructor to initialize this variable by calling the base class constructor, a b_display() function to display bicycle-specific details along with vehicle details, and a destructor.

Main Function:

The main function demonstrates the usage of the classes by creating objects of car, motorcycle, and bicycle. It creates objects with specific values and calls the respective display functions to print the details of each vehicle.

Overall, the code showcases the use of inheritance in C++ to represent different types of vehicles with their unique characteristics and behaviors. It demonstrates single inheritance, where derived classes inherit from a single base class.

```
#include <iostream>

using namespace std;

class vehicle
{
private:
    int year;
    string make;
    string model;
    string color;

public:
    vehicle(int year, string make, string model, string color);
    void display();
    ~vehicle();
};

vehicle::vehicle(int year, string make, string model, string color)
{
    this->year = year;
    this->make = make;
    this->model = model;
    this->color;
}
void vehicle::display()
{
    cout << "   Make: " << make << "\n";
    cout << "   Model: " << model << "\n";
    cout << "   Lounch Year: " << year << "\n";
}
vehicle::~vehicle()
{
}

class car : protected vehicle
{
private:
    int num_door;
    string variant;
    int top_speed;
    int tank;
```

```
public:
    car(int year, string make, string model, string color, int num_door, string
variant, int top_speed, int tank);
    void c_display();
    ~car();
};

car::car(int year, string make, string model, string color, int num_door, string
variant, int top_speed, int tank) : vehicle(year, make, model, color)
{
    this->num_door = num_door;
    this->variant = variant;
    this->top_speed = top_speed;
    this->tank = tank;
}
void car::c_display()
{
    cout << "============= Car =============\n";
    display();
    cout << "   Top Speed: " << top_speed << "\n";
    cout << "   Number of Doos: " << num_door << "\n";
    cout << "   Varient: " << variant << "\n";
    cout << "   Tank Capacity: " << tank << " gal." << "\n";
}
car::~car()
{
}

class motocycle : protected vehicle
{
private:
    string type;
    float fuel;
    float min;

public:
    motocycle(int year, string make, string model, string color, string type,
float fuel, float min);
    void m_display();
    ~motocycle();
};

motocycle::motocycle(int year, string make, string model, string color, string
type, float fuel, float min) : vehicle(year, make, model, color)
{
    this->type = type;
    this->fuel = fuel;
    this->min = min;
}
void motocycle::m_display()
{
    cout << "========== MotorCycle ==========\n";
    display();
    cout << "   Type: " << type << "\n";
    cout << "   1-100 Time: " << min << " sec" << "\n";
    cout << "   Fuel Capacity: " << fuel << " liters" << "\n";
}
motocycle::~motocycle()
{
}
```

```
class bicycle : protected vehicle
{
private:
    string manufacturer;

public:
    bicycle(int year, string make, string model, string color, string
manufacturer);
    void b_display();
    ~bicycle();
};

bicycle::bicycle(int year, string make, string model, string color, string
manufacturer) : vehicle(year, make, model, color)
{
    this->manufacturer = manufacturer;
}
void bicycle::b_display()
{
    cout << "=========== Bi-Cycle ===========\n";
    display();
    cout << "   Manufacturer: " << manufacturer << "\n";
}
bicycle::~bicycle()
{
}

int main()
{
    car c1(2012, "Toyta", "Supra", "Blue", 2, "Sports Car", 155, 13);
    c1.c_display();
    cout << endl;

    motocycle m1(2021, "YAMAHA", "R6", "Black", "Sport Bike", 17.06, 6.08);
    m1.m_display();
    cout << endl;

    bicycle b1(2019, "Morgan U.S.A", "MOR 05", "red", "Mountain Cycle");
    b1.b_display();
    cout << endl;
    return 0;
}
```

```
============= Car =============
   Make: Toyta
   Model: Supra
   Lounch Year: 2012
   Top Speed: 155
   Number of Doos: 2
   Varient: Sports Car
   Tank Capacity: 13 gal.

========== MotorCycle ==========
   Make: YAMAHA
   Model: R6
   Lounch Year: 2021
   Type: Sport Bike
   1-100 Time: 6.08 sec
   Fuel Capacity: 17.06 liters

=========== Bi-Cycle ===========
   Make: Morgan U.S.A
   Model: MOR 05
   Lounch Year: 2019
   Manufacturer: Mountain Cycle
```

Task 03

Overview

The program consists of several classes that represent different levels of inheritance hierarchy:

- Specie class: This is the base class that represents a species in the zoo. It contains common attributes such as name, cage number, age, weight, and average life. It also provides methods for displaying species details.
- Animals class: This class inherits from the specie class and represents a specific animal in the zoo. It adds additional attributes related to animal characteristics, such as reproduction, breed, and color. It provides methods for displaying animal details.
- Reptile class: This class also inherits from the specie class and represents a reptile species in the zoo. It adds attributes specific to reptiles, such as habitat and length. It provides methods for displaying reptile details.
- Dog class: This class inherits from the animals class and represents a dog in the zoo. It adds dog-specific attributes like voice and bite level. It also overrides the print_hobbies() method to provide dog-specific hobbies.
- Cat class: This class also inherits from the animals class and represents a cat in the zoo. It adds cat-specific attributes like gender and feed. It overrides the print_hobbies() method to provide cat-specific hobbies.

- Snack class: This class inherits from the reptile class and represents a snake in the zoo. It adds snack-specific attributes like type and calories. It overrides the print_hobbies() method to provide snake-specific hobbies.

- Crocodile class: This class also inherits from the reptile class and represents a crocodile in the zoo. It adds crocodile-specific attributes like aquatic behavior and number of teeth. It overrides the print_hobbies() method to provide crocodile-specific hobbies.

The main() function demonstrates the usage of these classes by creating instances of different animals, setting their attributes, and displaying their details using the respective display methods.

```
#include <iostream>

using namespace std;

class specie
{
private:
    string name;
    int cage_no;
    int age;
    int weight;
    int avg_life;
public:
    specie(string name, int cage_no, int age, int weight, int avg_life);
    void specie_display();
    ~specie();
};

specie::specie(string name, int cage_no, int age, int weight, int avg_life)
{
    this->name = name;
    this->cage_no = cage_no;
    this->age = age;
    this->weight = weight;
    this->avg_life = avg_life;
}
void specie::specie_display()
{
    cout << "  -> Name: " << name << "\n";
    cout << "  -> Cage Number: " << cage_no << "\n";
    cout << "  -> Age: " << age << " years\n";
    cout << "  -> Weight: " << weight <<" kg\n";
    cout << "  -> Average Life: " << avg_life << " years\n";
}
specie::~specie()
{
}

class animals : protected specie
{
private:
    string reproduction;
    string breed;
    string colour;
```

```
public:
    animals(string name, int cage_no, int age, int weight, int avg_life, string
reproduction, string breed, string colour);
    void animal_display();
    ~animals();
};

animals::animals(string name, int cage_no, int age, int weight, int avg_life,
string reproduction, string breed, string colour)
    : specie(name, cage_no, age, weight, avg_life)
{
    this->reproduction = reproduction;
    this->breed = breed;
    this->colour = colour;
}
void animals ::animal_display()
{
    specie_display();
    cout << "  -> Breed: " << breed << "\n";
    cout << "  -> Colour: " << colour << "\n";
    cout << "  -> Reproduction: " << reproduction << "\n";
}
animals::~animals()
{
}

class reptile : protected specie
{
private:
    string habitat;
    float length;

public:
    reptile(string name, int cage_no, int age, int weight, int avg_life, string
habitat, float length);
    void reptile_display();
    ~reptile();
};

reptile::reptile(string name, int cage_no, int age, int weight, int avg_life,
string habitat, float length)
    : specie(name, cage_no, age, weight, avg_life)
{
    this->habitat = habitat;
    this->length = length;
}
void reptile::reptile_display()
{
    specie_display();
    cout << "  -> Habitat: " << habitat << "\n";
    cout << "  -> Length: " << length << " ft.\n";
}
reptile::~reptile()
{
}

class dog : protected animals
{
private:
    string voice;
```

```
    int bite_level;

public:
    dog(string name, int cage_no, int age, int weight, int avg_life, string
reproduction, string breed, string colour, string voice, int bite_level);
    void dog_dispaly();
    void print_hobbies();
    ~dog();
};

dog::dog(string name, int cage_no, int age, int weight, int avg_life, string
reproduction, string breed, string colour, string voice, int bite_level)
    : animals(name, cage_no, age, weight, avg_life, reproduction, breed, colour)
{
    this->voice = voice;
    this->bite_level = bite_level;
}
void dog ::dog_dispaly()
{
    animal_display();
    cout << "  -> Voice: " << voice << "\n";
    cout << "  -> Bite Level: " << bite_level << " N\n";
    print_hobbies();
}
void dog::print_hobbies()
{
    cout << "  -> Hobbies: Playing fetch\n     Going for walks\n";
}
dog::~dog()
{
}

class cat : protected animals
{
private:
    string gender;
    string feed;

public:
    cat(string name, int cage_no, int age, int weight, int avg_life, string
reproduction, string breed, string colour, string gender, string feed);
    void cat_display();
    void print_hobbies();
    ~cat();
};

cat::cat(string name, int cage_no, int age, int weight, int avg_life, string
reproduction, string breed, string colour, string gender, string feed)
    : animals(name, cage_no, age, weight, avg_life, reproduction, breed, colour)
{
    this->gender = gender;
    this->feed = feed;
}
void cat::cat_display()
{
    animal_display();
    cout << "  -> Gender: " << gender << "\n";
    cout << "  -> Feed: " << feed << "\n";
    print_hobbies();
}
```

```
void cat::print_hobbies()
{
    cout << "  -> Hobbies: Playing with toys\n     Hunting\n";
}
cat::~cat()
{
}

class snack : protected reptile
{
private:
    string type;
    int calories;

public:
    snack(string name, int cage_no, int age, int weight, int avg_life, string
habitat, float length, string type, int calories);
    void snack_display();
    void print_hobbies();
    ~snack();
};

snack::snack(string name, int cage_no, int age, int weight, int avg_life, string
habitat, float length, string type, int calories)
    : reptile(name, cage_no, age, weight, avg_life, habitat, length)
{
    this->type = type;
    this->calories = calories;
}
void snack::snack_display()
{
    reptile_display();
    cout << "  -> Type: " << type << "\n";
    cout << "  -> Calories: " << calories << "\n";
    print_hobbies();
}
void snack::print_hobbies()
{
    cout << "  -> Hobbies: Sunbathing\n     Exploring surroundings\n";
}
snack::~snack()
{
}

class crocodile : protected reptile
{
private:
    string isAquatic;
    int numTeeth;

public:
    crocodile(string name, int cage_no, int age, int weight, int avg_life, string
habitat, float length, string isAquatic, int numTeeth);
    void print_hobbies();
    void crocodile_display();
    ~crocodile();
};

crocodile::crocodile(string name, int cage_no, int age, int weight, int avg_life,
string habitat, float length, string isAquatic, int numTeeth)
```

```
    : reptile(name, cage_no, age, weight, avg_life, habitat, length)
{
    this->isAquatic = isAquatic;
    this->numTeeth = numTeeth;
}
void crocodile::print_hobbies()
{
    cout << "  -> Hobbies: Swimming\n     Basking in the sun\n";
}
void crocodile ::crocodile_display()
{
    reptile_display();
    cout << "  -> Is Aquatic: " << isAquatic << "\n";
    cout << "  -> Number of Teeths: " << numTeeth << "\n";
    print_hobbies();
}
crocodile::~crocodile()
{
}

int main()
{
    cout << "********** Dog Details **********\n";
    dog dog1("Tommy", 1, 3, 15, 10, "Give Birth to Child", "Labrador", "Golden",
"Bark", 230);
    dog1.dog_dispaly();
    cout << endl;

    cout << "********** Cat Details **********\n";
    cat cat1("Whiskers", 2, 5, 8, 15, "Give Birth to Child", "Siamese", "White",
"Female", "Fish");
    cat1.cat_display();
    cout << endl;

    cout << "******** Crocodile Details ********\n";
    crocodile cro1("Crock", 3, 10, 500, 60, "Swamp", 6.2, "Yes", 80);
    cro1.crocodile_display();
    cout << endl;

    cout << "********** Snack Details **********\n";
    snack mySnack("Snakey", 4, 2, 2, 5, "Forest", 1.5, "Python", 100);
    mySnack.snack_display();
    cout << endl;

    return 0;
}
```

```
********** Dog Details **********
  -> Name: Tommy
  -> Cage Number: 1
  -> Age: 3 years
  -> Weight: 15 kg
  -> Average Life: 10 years
  -> Breed: Labrador
  -> Colour: Golden
  -> Reproduction: Give Birth to Child
  -> Voice: Bark
  -> Bite Level: 230 N
  -> Hobbies: Playing fetch
     Going for walks
```

```
********** Cat Details **********
  -> Name: Whiskers
  -> Cage Number: 2
  -> Age: 5 years
  -> Weight: 8 kg
  -> Average Life: 15 years
  -> Breed: Siamese
  -> Colour: White
  -> Reproduction: Give Birth to Child
  -> Gender: Female
  -> Feed: Fish
  -> Hobbies: Playing with toys
     Hunting
```

```
******** Crocodile Details ********
  -> Name: Crock
  -> Cage Number: 3
  -> Age: 10 years
  -> Weight: 500 kg
  -> Average Life: 60 years
  -> Habitat: Swamp
  -> Length: 6.2 ft.
  -> Is Aquatic: Yes
  -> Number of Teeths: 80
  -> Hobbies: Swimming
     Basking in the sun
```

```
********** Snack Details **********
  -> Name: Snakey
  -> Cage Number: 4
  -> Age: 2 years
  -> Weight: 2 kg
  -> Average Life: 5 years
  -> Habitat: Forest
  -> Length: 1.5 ft.
  -> Type: Python
  -> Calories: 100
  -> Hobbies: Sunbathing
     Exploring surroundings
```

Task 04

We define a base class called "character" that includes shared attributes like age, level, strength, name, and gender. The derived classes, namely "warrior," "mages," and "archer," inherit from the base class and add their specific attributes.

Character

The base class character contains the following attributes and methods:

Attributes:

age: Represents the age of the character.
level: Represents the level of the character.
strength: Represents the strength of the character.
name: Represents the name of the character.
gender: Represents the gender of the character.

Methods:

character: Constructor to initialize the character object with the given attributes.
char_display: Method to display the character's information.
~character: Destructor to clean up any resources used by the character object.

Derived Classes

The derived classes warrior, mages, and archer inherit from the base class character and add their specific attributes and methods.

Warrior Class

The warrior class is derived from the character base class and includes the following additional attributes and methods:

Attributes:

speciality: Represents the speciality of the warrior.
armor: Represents the remaining armor of the warrior.
battle: Represents the total number of battles the warrior has participated in.
win_battle: Represents the total number of battles the warrior has won.

Methods:

warrior: Constructor to initialize the warrior object with the given attributes, including base class attributes.
war_display: Method to display the warrior's information, including base class information.
~warrior: Destructor to clean up any resources used by the warrior object.

Mages Class

The mages class is derived from the character base class and includes the following additional attributes and methods:

Attributes:

trick_01: Represents the first magic trick of the mage.
trick_02: Represents the second magic trick of the mage.
kills: Represents the total number of kills achieved by the mage.

Methods:

mages: Constructor to initialize the mage object with the given attributes, including base class attributes.
mag_diaplay: Method to display the mage's information, including base class information.
~mages: Destructor to clean up any resources used by the mage object.

Archer Class

The archer class is derived from the character base class and includes the following additional attributes and methods:

Attributes:

super_power: Represents the superpower of the archer.
accuracy: Represents the accuracy of the archer in throwing arrows.
sight: Represents the distance the archer can throw arrows.

Methods:

archer: Constructor to initialize the archer object with the given attributes, including base class attributes.
arc_display: Method to display the archer's information, including base class information.
~archer: Destructor to clean up any resources used by the archer object.

Main Function

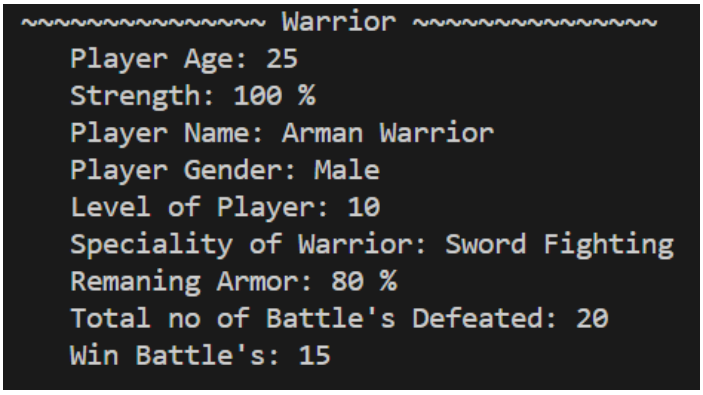
The main function demonstrates the usage of the derived classes. It creates objects of the derived classes, initializes them with specific attributes, and calls the display methods to show the characters' information.
represent different levels.

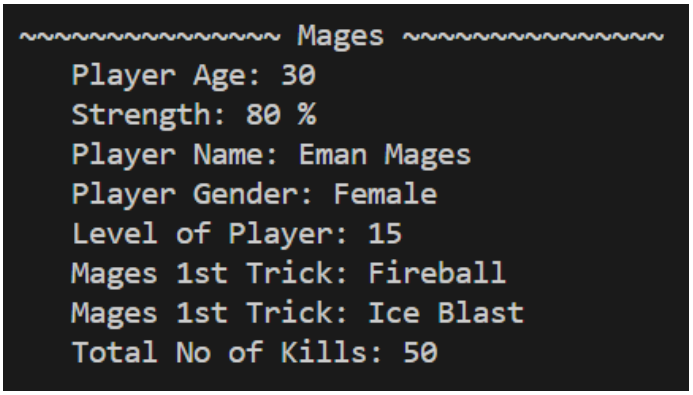
```
#include <iostream>

using namespace std;
class character
{
```

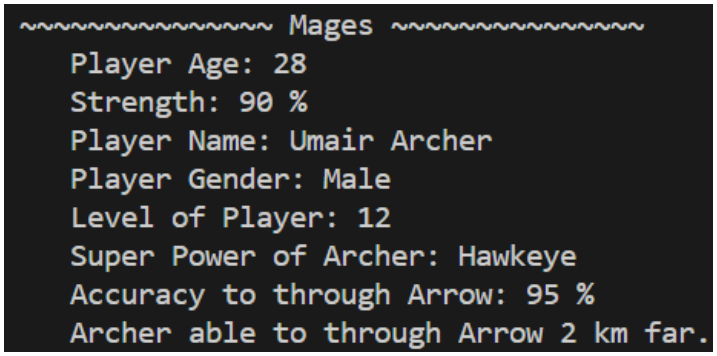
```
private:
    /* data */
    int age;
    int level;
    int strength;
    string name;
    string gender;

public:
    character(string name, int age, int level, int strength, string gender);
    void char_display();
    ~character();
};

character::character(string name, int age, int level, int strength, string
gender)
{
    this->name = name;
    this->age = age;
    this->level = level;
    this->strength = strength;
    this->gender = gender;
}
void character ::char_display()
{
    cout << "   Player Age: " << age << "\n";
    cout << "   Strength: " << strength << " %\n";
    cout << "   Player Name: " << name << "\n";
    cout << "   Player Gender: " << gender << "\n";
    cout << "   Level of Player: " << level << "\n";
}
character::~character()
{
}

class warrior : protected character
{
private:
    /* data */
    string speciality;
    int armor;
    int battle;     // total wars
    int win_battle; // total war wins
public:
    warrior(string name, int age, int level, int strength, string gender, string
speciality, int armor, int battle, int win_battle);
    void war_display();
    ~warrior();
};

warrior::warrior(string name, int age, int level, int strength, string gender,
string speciality, int armor, int battle, int win_battle)
    : character(name, age, level, strength, gender)
{
    this->speciality = speciality;
    this->armor = armor;
    this->battle = battle;
    this->win_battle = win_battle;
}
void warrior ::war_display()
```

```
{
    cout << "~~~~~~~~~~~~~~~ Warrior ~~~~~~~~~~~~~~~\n";
    char_display();
    cout << "   Speciality of Warrior: " << speciality << "\n";
    cout << "   Remaning Armor: " << armor << " %\n";
    cout << "   Total no of Battle's Defeated: " << battle << "\n";
    cout << "   Win Battle's: " << win_battle << "\n";
}
warrior::~warrior()
{
}

class mages : protected character
{
private:
    /* data */
    string trick_01; // magic tick name 01
    string trick_02; // magic tick name 02
    int kills;

public:
    mages(string name, int age, int level, int strength, string gender, string
trick_01, string trick_02, int kills);
    void mag_diaplay();
    ~mages();
};

mages::mages(string name, int age, int level, int strength, string gender, string
trick_01, string trick_02, int kills)
    : character(name, age, level, strength, gender)
{
    this->trick_01 = trick_01;
    this->trick_02 = trick_02;
    this->kills = kills;
}
void mages::mag_diaplay()
{
    cout << "~~~~~~~~~~~~~~~ Mages ~~~~~~~~~~~~~~~\n";
    char_display();
    cout << "   Mages 1st Trick: " << trick_01 << "\n";
    cout << "   Mages 1st Trick: " << trick_02 << "\n";
    cout << "   Total No of Kills: " << kills << "\n";
}
mages::~mages()
{
}

class archer : protected character
{
private:
    /* data */
    string super_power;
    int accuracy;
    int sight;

public:
    archer(string name, int age, int level, int strength, string gender, string
super_power, int accuracy, int sight);
    void arc_display();
    ~archer();
```

```
};

archer::archer(string name, int age, int level, int strength, string gender,
string super_power, int accuracy, int sight)
    : character(name, age, level, strength, gender)
{
    this->super_power = super_power;
    this->accuracy = accuracy;
    this->sight = sight;
}
void archer ::arc_display()
{
    cout << "~~~~~~~~~~~~~~~ Mages ~~~~~~~~~~~~~~~\n";
    char_display();
    cout << "   Super Power of Archer: " << super_power << "\n";
    cout << "   Accuracy to through Arrow: " << accuracy << " %\n";
    cout << "   Archer able to through Arrow " << sight << " km far.\n";
}
archer::~archer()
{
}

int main()
{
    warrior w1("Arman Warrior", 25, 10, 100, "Male", "Sword Fighting", 80, 20,
15);
    w1.war_display();
    cout << endl;

    mages m1("Eman Mages", 30, 15, 80, "Female", "Fireball", "Ice Blast", 50);
    m1.mag_diaplay();
    cout << endl;

    archer a1("Umair Archer", 28, 12, 90, "Male", "Hawkeye", 95, 2);
    a1.arc_display();

    return 0;
}
```







Task 05

Advantages of Multiple Inheritance:

- **Code Reusability:** Multiple inheritance allows reusing code from multiple base classes, reducing duplication and promoting code reuse. For example, a car class can inherit specific features from sports, electric, and hybrid classes without rewriting common functionalities.

- **Flexibility and Extensibility:** Multiple inheritance enables easily extending a class's functionality by inheriting from multiple base classes. It allows creating complex class hierarchies and combining different features from various sources.

- **Precise Representation:** Multiple inheritance accurately represents real-world scenarios involving multiple traits or characteristics. It helps represent different vehicle models in the car class with specific features and common attributes.

Challenges by using Multiple Inheritance:

- **Name and Scope Clashes**: Multiple inheritance can lead to conflicts in member function or variable names across base classes. Explicit scoping is needed to access the desired member in such cases.

- **Complexity and Maintenance**: Multiple inheritance can result in complex class hierarchies, making the code harder to understand, debug, and maintain. Modifying one base class can have unforeseen effects on derived classes, increasing the risk of introducing bugs.

- **Diamond Inheritance Problem:** If multiple base classes share a common base class, and a class inherits from these base classes, the diamond inheritance problem may arise. It causes ambiguity in accessing inherited members due to multiple instances of the same base class.

- **Coupling and Dependency:** Multiple inheritance can introduce tight coupling and high dependency between classes, making the code less modular and challenging to modify or replace parts of the inheritance chain.

```
#include <iostream>
using namespace std;

class sports
{
private:
    int maxSpeed;
    int sec;      // 1-100 taken time

public:
    sports(int maxSpeed, int sec) : maxSpeed(maxSpeed), sec(sec) {}
    void spoDisplay();
    ~sports();
};

void sports::spoDisplay()
{
    cout << "  * Sports Car Max Speed: " << maxSpeed << " km/h" << endl;
    cout << "  * Sports Car 0-100 km/h Time: " << sec << " seconds" << endl;
}

sports::~sports() {}

class electric
{
private:
    int capacity; // Battery capacity
    int km;       // kilo meters cover

public:
    electric(int capacity, int km) : capacity(capacity), km(km) {}
    void eleDisplay();
    ~electric();
};

void electric::eleDisplay()
{
    cout << "  * Electric Car Battery Capacity: " << capacity << " kWh" << endl;
    cout << "  * Electric Car Range: " << km << " km" << endl;
}

electric::~electric() {}

class hybrid
{
private:
    int power;           // power in herdz
    string fuelType;
    int octanNO;        // fuel octan number
    int tankCapacity;
    int cylinder;       // number of cylinders

public:
    hybrid(int power, string fuelType, int octanNO, int tankCapacity, int
cylinder)
```

```
        : power(power), fuelType(fuelType), octanNO(octanNO),
tankCapacity(tankCapacity), cylinder(cylinder) {}
    void hybDisplay();
    ~hybrid();
};

void hybrid::hybDisplay()
{
    cout << "  * Hybrid Car Power: " << power << " herdz" << endl;
    cout << "  * Hybrid Car Fuel Type: " << fuelType << endl;
    cout << "  * Hybrid Car Octane Number: " << octanNO << endl;
    cout << "  * Hybrid Car Tank Capacity: " << tankCapacity << " liters" <<
endl;
    cout << "  * Hybrid Car Number of Cylinders: " << cylinder << endl;
}

hybrid::~hybrid() {}

class car : protected sports, protected electric, protected hybrid
{
private:
    int price;
    string offRoading;
    int year;
    string make;
    string model;
    string colour;

public:
    car(int price, string offRoading, int year, string make, string model, string
colour,
        int sportsMaxSpeed, int sportsSec, int electricCapacity, int electricKm,
        int hybridPower, string hybridFuelType, int hybridOctanNO, int
hybridTankCapacity, int hybridCylinder)
        : sports(sportsMaxSpeed, sportsSec), electric(electricCapacity,
electricKm),
          hybrid(hybridPower, hybridFuelType, hybridOctanNO, hybridTankCapacity,
hybridCylinder),
          price(price), offRoading(offRoading), year(year), make(make),
model(model), colour(colour) {}

    void carDisplay();
    ~car();
};

void car::carDisplay()
{
    cout << "  * Make: " << make << endl;
    cout << "  * Colour: " << colour << endl;
    cout << "  * Model: " << model << endl;
    cout << "  * Year: " << year << endl;
    cout << "  * Price: $" << price << endl;
    cout << "  * Off-Roading: " << offRoading << endl;
    spoDisplay();
    eleDisplay();
    hybDisplay();
}

car::~car() {}
```

```cpp
int main()
{
    car car01(50000, "No", 2023, "Toyota", "Camry", "Blue", 280, 5, 75, 400, 150,
"Electric", 91, 60, 4);
    cout << "<><><><><><><><< CAR 01 DETAILS >><><><><><><><><>\n";
    car01.carDisplay();
    cout << endl;

    car car02(60000, "Yes", 2022, "Ford", "Mustang", "Red", 320, 4, 80, 350, 180,
"Gasoline", 95, 70, 6);
    cout << "<><><><><><><><< CAR 02 DETAILS >><><><><><><><><>\n";
    car02.carDisplay();
    cout << endl;

    car car03(40000, "Yes", 2021, "Tesla", "Model S", "Black", 300, 6, 100, 500,
120, "Electric", 89, 55, 4);
    cout << "<><><><><><><><< CAR 03 DETAILS >><><><><><><><><>\n";
    car03.carDisplay();
    cout << endl;

    return 0;
}
```

```
<><><><><><><><< CAR 02 DETAILS >><><><><><><><><>
  * Make: Ford
  * Colour: Red
  * Model: Mustang
  * Year: 2022
  * Price: $60000
  * Off-Roading: Yes
  * Sports Car Max Speed: 320 km/h
  * Sports Car 0-100 km/h Time: 4 seconds
  * Electric Car Battery Capacity: 80 kWh
  * Electric Car Range: 350 km
  * Hybrid Car Power: 180 herdz
  * Hybrid Car Fuel Type: Gasoline
  * Hybrid Car Octane Number: 95
  * Hybrid Car Tank Capacity: 70 liters
  * Hybrid Car Number of Cylinders: 6
```

```
<><><><><><><><< CAR 01 DETAILS >><><><><><><><><>
  * Make: Toyota
  * Colour: Blue
  * Model: Camry
  * Year: 2023
  * Price: $50000
  * Off-Roading: No
  * Sports Car Max Speed: 280 km/h
  * Sports Car 0-100 km/h Time: 5 seconds
  * Electric Car Battery Capacity: 75 kWh
  * Electric Car Range: 400 km
  * Hybrid Car Power: 150 herdz
  * Hybrid Car Fuel Type: Electric
  * Hybrid Car Octane Number: 91
  * Hybrid Car Tank Capacity: 60 liters
  * Hybrid Car Number of Cylinders: 4
```

```
<><><><><><><><< CAR 03 DETAILS >><><><><><><><><>
  * Make: Tesla
  * Colour: Black
  * Model: Model S
  * Year: 2021
  * Price: $40000
  * Off-Roading: Yes
  * Sports Car Max Speed: 300 km/h
  * Sports Car 0-100 km/h Time: 6 seconds
  * Electric Car Battery Capacity: 100 kWh
  * Electric Car Range: 500 km
  * Hybrid Car Power: 120 herdz
  * Hybrid Car Fuel Type: Electric
  * Hybrid Car Octane Number: 89
  * Hybrid Car Tank Capacity: 55 liters
  * Hybrid Car Number of Cylinders: 4
```

Task 06

Benefits:

- **Avoiding duplication:** Virtual inheritance ensures that the shared base class, "employee" in this case, is inherited only once by the derived class "admin." This prevents duplication of member variables and functions from the shared base class, avoiding potential conflicts and reducing memory overhead.

- **Proper object slicing:** When objects of derived classes are treated as objects of the shared base class, virtual inheritance ensures that the derived objects are properly sliced and only the shared base class part is accessed. This allows objects of "admin" to be used interchangeably with objects of the shared base class or its other derived classes.

Considerations:

- **Design complexity:** Virtual inheritance introduces additional complexity to the class hierarchy and requires careful design and understanding of object relationships. It may not be necessary or appropriate for all situations, and its usage should be justified based on the specific requirements and constraints of the application.

- **Runtime overhead:** Virtual inheritance can introduce slight runtime overhead due to the additional level of indirection required to access virtual base class members. However, this overhead is typically negligible unless performance is a critical concern in the specific application.

Overall, virtual inheritance provides a mechanism to handle multiple inheritance situations with a shared base class, ensuring proper object slicing and avoiding duplication. However, its usage should be carefully considered and justified based on the specific requirements and complexity of the application.

```cpp
#include <iostream>
using namespace std;

class employee
{
private:
    string name;
    int age;
    int id;
    string department;
    float salary;

public:
    employee(string name, int age, int id, string department, float salary);
    void work();
    void empDisplay();
    ~employee();
};

employee::employee(string name, int age, int id, string department, float salary)
{
    this->name = name;
    this->age = age;
    this->id = id;
    this->department = department;
    this->salary = salary;
}
```

```
void employee::work()
{
    cout << "  -> Employee's Working in Hospital:\n";
    cout << "  ->    1. Doctors\n   2. Nurses\n   3. Technicians\n";
}

void employee::empDisplay()
{
    cout << "  -> Name: " << name << "\n";
    cout << "  -> Age: " << age << " years\n";
    cout << "  -> Employee ID: " << id << "\n";
    cout << "  -> Salary: " << salary << "\n";
    cout << "  -> Department: " << department << "\n";
}

employee::~employee()
{
}

class doctor : virtual public employee
{
protected:
    int hours;    // Duty hours
    int patients; // Diagnosed patients per day
    string specialization;

public:
    doctor(string name, int age, int id, string department, float salary, int
hours, int patients, string specialization);
    void diagnose();
    void docDisplay();
    ~doctor();
};

doctor::doctor(string name, int age, int id, string department, float salary, int
hours, int patients, string specialization)
    : employee(name, age, id, department, salary)
{
    this->hours = hours;
    this->patients = patients;
    this->specialization = specialization;
}

void doctor::diagnose()
{
    cout << "  -> Doctor Diagnosing Patients.\n";
}

void doctor::docDisplay()
{
    cout << "  -> Duty Hours Per Day: " << hours << "\n";
    cout << "  -> Diagnosed Patients Per Day: " << patients << "\n";
    cout << "  -> Doctor's Specialization: " << specialization << "\n";
}

doctor::~doctor()
{
}

class nurse : virtual public employee
```

```
{
private:
    string qualification;
    string shift; // Day or Night Duty

public:
    nurse(string name, int age, int id, string department, float salary, string
qualification, string shift);
    void assist();
    void nursDisplay();
    ~nurse();
};

nurse::nurse(string name, int age, int id, string department, float salary,
string qualification, string shift)
    : employee(name, age, id, department, salary)
{
    this->qualification = qualification;
    this->shift = shift;
}

void nurse::assist()
{
    cout << "  -> Nurse Assisting Patients.\n";
}

void nurse::nursDisplay()
{
    cout << "  -> Qualification: " << qualification << "\n";
    cout << "  -> Duty Shift: " << shift << "\n";
}

nurse::~nurse()
{
}

class technician : virtual public employee
{
private:
    int experience; // working since
    int no_of_tec;
    float budget; // budget given to technician

public:
    technician(string name, int age, int id, string department, float salary, int
experience, int no_of_tec, float budget);
    void tecDisplay();
    ~technician();
};

technician::technician(string name, int age, int id, string department, float
salary, int experience, int no_of_tec, float budget)
    : employee(name, age, id, department, salary)
{
    this->experience = experience;
    this->no_of_tec = no_of_tec;
    this->budget = budget;
}

void technician::tecDisplay()
```

```
{
    cout << "  -> Working Experience: " << experience << " years\n";
    cout << "  -> No of Technicians in Hospital: " << no_of_tec << "\n";
    cout << "  -> Budget Given to Technician: " << budget << "\n";
}

technician::~technician()
{
}

class admin : public doctor, public nurse, public technician
{
private:
    int staff; // total staff members in admin

public:
    admin(string name, int age, int id, string department, float salary, int
staff, int experience, int no_of_tec, float budget, string qualification, string
shift);
    void role();
    void admDisplay();
    ~admin();
};

admin::admin(string name, int age, int id, string department, float salary, int
staff, int experience, int no_of_tec, float budget, string qualification, string
shift)
    : employee(name, age, id, department, salary),
    doctor(name, age, id, department, salary, 0, 0, ""),
    nurse(name, age, id, department, salary, "", ""),
    technician(name, age, id, department, salary, 0, 0, 0.0)
{
    this->staff = staff;
}

void admin::role()
{
    cout << "  -> Admins are able to manage all employees.\n";
}

void admin::admDisplay()
{
    cout << "  -> Admin Staff Members: " << staff << "\n";
}

admin::~admin()
{
}

int main()
{
    admin a1("Arman Tahir", 35, 1, "ICU", 5000, 10, 5, 3, 10000, "BSc Nursing",
"Day Shift");
    cout << "******* DOCTOR DETAILS *******\n";
    a1.empDisplay();
    a1.docDisplay();
    a1.diagnose();
    cout << endl;
```

```
    admin a2("Umair Inayat", 40, 2, "Administration", 6000, 15, 7, 4, 12000,
"MBA", "Night Shift");
    cout << "******* ADMIN DETAILS *******\n";
    a2.empDisplay();
    a2.admDisplay();
    a2.role();
    cout << endl;

    admin a3("Eman Murtaza", 25, 3, "Nursing", 3000, 2, 8, 2, 8000, "Diploma in
Nursing", "Day Shift");
    cout << "******* NURSE DETAILS *******\n";
    a3.empDisplay();
    a3.nursDisplay();
    a3.assist();
    cout << endl;

    admin a4("Abdul Hadi", 30, 4, "Technician", 4000, 5, 10, 5, 15000, "Diploma
in Electronics", "Night Shift");
    cout << "******* TECHNICIAN DETAILS *******\n";
    a4.empDisplay();
    a4.tecDisplay();
    a4.admDisplay();
    cout << endl;

    return 0;
}
```

```
******* NURSE DETAILS *******
  -> Name: Eman Murtaza
  -> Age: 25 years
  -> Employee ID: 3
  -> Salary: 3000
  -> Department: Nursing
  -> Qualification:
  -> Duty Shift:
  -> Nurse Assisting Patients.

******* TECHNICIAN DETAILS *******
  -> Name: Abdul Hadi
  -> Age: 30 years
  -> Employee ID: 4
  -> Salary: 4000
  -> Department: Technician
  -> Working Experience: 0 years
  -> No of Technicians in Hospital: 0
  -> Budget Given to Technician: 0
  -> Admin Staff Members: 5
```

```
******* DOCTOR DETAILS *******
  -> Name: Arman Tahir
  -> Age: 35 years
  -> Employee ID: 1
  -> Salary: 5000
  -> Department: ICU
  -> Duty Hours Per Day: 0
  -> Diagnosed Patients Per Day: 0
  -> Doctor's Specialization:
  -> Doctor Diagnosing Patients.

******* ADMIN DETAILS *******
  -> Name: Umair Inayat
  -> Age: 40 years
  -> Employee ID: 2
  -> Salary: 6000
  -> Department: Administration
  -> Admin Staff Members: 15
  -> Admins are able to manage all e
```

Task 07

```cpp
#include <iostream>
using namespace std;

class employee
{
private:
    int id;
    string name;
    float salary;
    int grade;

public:
    employee(int id, string name, float salary, int grade);
    void empDisplay();
    float getsalary();
    ~employee();
};

employee::employee(int id, string name, float salary, int grade)
{
    this->name = name;
    this->id = id;
    this->salary=salary;
    this->grade = grade;
}
void employee ::empDisplay()
{
    cout << "  -> ID: " << id << "\n";
    cout << "  -> Name: " << name << "\n";
    cout << "  -> Grade: " << grade << "\n";
    cout << "  -> Salary: " << salary << "\n";
}
float employee::getsalary()
{
    return salary;
}
employee::~employee()
{
}

class professors : public employee
{
private:
    /* data */
    int total_classes; // classes in ine day
    int class_time;    // time of each class
    int extra_lecture; // extra_lecture if any
    string subject;

public:
    professors(int id, string name, float salary, int grade, int total_classes,
int class_time, int extra_lecture, string subject);
    void proDisplay();
    ~professors();
};
```

```
professors::professors(int id, string name, float salary, int grade, int
total_classes, int class_time, int extra_lecture, string subject) : employee(id,
name, salary, grade)
{
    this->total_classes = total_classes;
    this->class_time = class_time;
    this->extra_lecture = extra_lecture;
    this->subject = subject;
}
void professors::proDisplay()
{
    cout << "  -> Subject: " << subject << "\n";
    cout << "  -> Total Class in Day: " << total_classes << "\n";
    cout << "  -> Time of Each Class: " << class_time << "\n";
    cout << "  -> Total Net Salary: " << (getsalary() + (extra_lecture * 2000))
<< "\n"; // if their is any extra lecture 2000 increase per lecture
}
professors::~professors()
{
}

class staff : public employee
{
private:
    /* data */
    int num_of_cordinaters;
    int gurds;
    int swipers;

public:
    staff(int id, string name, float salary, int grade, int total_classes, int
class_time, int extra_lecture, int num_of_cordinaters, int gurds, int swipers);
    void staffDisplay();
    ~staff();
};

staff::staff(int id, string name, float salary, int grade, int total_classes, int
class_time, int extra_lecture, int num_of_cordinaters, int gurds, int swipers) :
employee(id, name, salary, grade)
{
    this->num_of_cordinaters = num_of_cordinaters;
    this->gurds = gurds;
    this->swipers = swipers;
}
void staff::staffDisplay()
{
    cout << "  -> Total # of Cordinater: " << num_of_cordinaters << "\n";
    cout << "  -> Total # of Gurds: " << gurds << "\n";
    cout << "  -> Total # of Swipers: " << swipers << "\n";
}
staff::~staff()
{
}

class admin : public employee
{
private:
    /* data */
    int employment_years;
    string department;
```

```
public:
    admin(int id, string name, float salary, int grade, int employment_years,
string department);
    void adminDisplay();
    ~admin();
};

admin::admin(int id, string name, float salary, int grade, int employment_years,
string department) : employee(id, name, salary, grade)
{
    this->employment_years = employment_years;
    this->department = department;
}
void admin::adminDisplay()
{
    cout << "  -> Employeed Since: " << employment_years << "\n";
    cout << "  -> Department: " << department << "\n";
}
admin::~admin()
{
}

int main()
{
    cout << "========== PROFESSORS DETAILS ==========\n";
    professors p1(123, "Eman Mutaza", 56000, 1, 4, 60, 2, "Mathematics");
    p1.empDisplay();
    p1.proDisplay();
    cout << endl;

    cout << "============= STAFF DETAILS =============\n";
    staff s1(456, "Ahmed", 33000, 2, 5, 40, 2, 3, 2, 4);
    s1.empDisplay();
    s1.staffDisplay();
    cout << endl;

    cout << "============= ADMIN DETAILS =============\n";
    admin a1(789, "Arman Tahir", 49000, 3, 10, "Human Resources");
    a1.empDisplay();
    a1.adminDisplay();
    cout << endl;

    return 0;
}
```

```
========== PROFESSORS DETAILS ==========
  -> ID: 123
  -> Name: Eman Mutaza
  -> Grade: 1
  -> Salary: 56000
  -> Subject: Mathematics
  -> Total Class in Day: 4
  -> Time of Each Class: 60
  -> Total Net Salary: 60000
```

```
============= STAFF DETAILS =============
  -> ID: 456
  -> Name: Ahmed
  -> Grade: 2
  -> Salary: 33000
  -> Total # of Cordinater: 3
  -> Total # of Gurds: 2
  -> Total # of Swipers: 4
```

```
============= ADMIN DETAILS =============
  -> ID: 789
  -> Name: Arman Tahir
  -> Grade: 3
  -> Salary: 49000
  -> Employeed Since: 10
  -> Department: Human Resources
```

Task 08

```cpp
#include <iostream>

using namespace std;

class animal
{
private:
    /* data */
    string name;
    int age;
    string habitat;
    float weight;
    string gender;

public:
    animal(string name, int age, string habitat, float weight, string gender);
    void aniDisplay();
    ~animal();
};

animal::animal(string name, int age, string habitat, float weight, string gender)
{
    this->name = name;
    this->age = age;
    this->habitat = habitat;
    this->weight = weight;
}
void animal::aniDisplay()
{
    cout << "  -> Name of Animal: "<<name << "\n";
    cout << "  -> Age of " << name << ": " << age << "\n";
    cout << "  -> Habitat of " << name << ": " << habitat << "\n";
    cout << "  -> Weight: " << weight << "\n";
}
animal::~animal()
{
}

class lions : protected animal
{
private:
    /* data */
    string food_source;
    string feeding_frequency;

public:
    lions(string name, int age, string habitat, float weight, string gender, string
food_source, string feeding_frequency);
    void lionDisplay();
    void getfun();
    ~lions();
};

lions::lions(string name, int age, string habitat, float weight, string gender, string
food_source, string feeding_frequency)
```

```
    : animal(name, age, habitat, weight, gender)
{
}
void lions::lionDisplay()
{
    cout << "  -> Food Source: " << food_source << "\n";
    cout << "  -> Feeding Frequency: " << feeding_frequency << "\n";
}
void lions::getfun()
{
    aniDisplay();
}
lions::~lions()
{
}

class tigers : protected animal
{
private:
    /* data */
    string special_ability;
    int life_span;

public:
    tigers(string name, int age, string habitat, float weight, string gender, string
special_ability, int life_span);
    void tigDisplay();
    void getfun();
    ~tigers();
};

tigers::tigers(string name, int age, string habitat, float weight, string gender, string
special_ability, int life_span)
    : animal(name, age, habitat, weight, gender)
{
    this->special_ability = special_ability;
    this->life_span = life_span;
}
void tigers ::tigDisplay()
{
    cout << "  -> Life Span: " << life_span << "\n";
    cout << "  -> Special Abitity: " << special_ability << "\n";
}
void tigers::getfun()
{
    aniDisplay();
}
tigers::~tigers()
{
}

class elephant : protected animal
{
private:
    /* data */
    float average_weight;
    int trunk_length;

public:
    elephant(string name, int age, string habitat, float weight, string gender, float
average_weight, int trunk_length);
    void eleDisplay();
    void getfun();
    ~elephant();
};

elephant::elephant(string name, int age, string habitat, float weight, string gender, float
average_weight, int trunk_length)
    : animal(name, age, habitat, weight, gender)
```

```
{
    this->average_weight = average_weight;
    this->trunk_length = trunk_length;
}
void elephant::eleDisplay()
{
    cout << "  -> Average Weight: " << average_weight << "\n";
    cout << "  -> Elephant Trunk Length: " << trunk_length << "\n";
}
void elephant::getfun()
{
    aniDisplay();
}
elephant::~elephant()
{
}

int main()
{
    cout << "============ LION DETAILS ============\n";
    lions l1("Simba", 5, "Savannah", 150.5, "Male", "Zebras", "Twice a day");
    l1.getfun();
    l1.lionDisplay();
    cout << endl;

    cout << "=========== TIGER DETAILS ===========\n";
    tigers t1("Rajah", 7, "Jungle", 200.3, "Male", "Camouflage", 15);
    t1.getfun();
    t1.tigDisplay();
    cout << endl;

    cout << "========== ELEPHANT DETAILS ==========\n";
    elephant e1("Dumbo", 10, "Forest", 3000.7, "Male", 5000.2, 200);
    e1.getfun();
    e1.eleDisplay();
    cout << endl;

    return 0;
}
```

```
============ LION DETAILS ============
  -> Name of Animal: Simba
  -> Age of Simba: 5
  -> Habitat of Simba: Savannah
  -> Weight: 150.5
  -> Food Source:
  -> Feeding Frequency:
```

```
=========== TIGER DETAILS ===========
  -> Name of Animal: Rajah
  -> Age of Rajah: 7
  -> Habitat of Rajah: Jungle
  -> Weight: 200.3
  -> Life Span: 15
  -> Special Abitity: Camouflage
```

```
========== ELEPHANT DETAILS ==========
  -> Name of Animal: Dumbo
  -> Age of Dumbo: 10
  -> Habitat of Dumbo: Forest
  -> Weight: 3000.7
  -> Average Weight: 5000.2
  -> Elephant Trunk Length: 200
```

Task 09

```
#include <iostream>
```

```cpp
#include <string>

using namespace std;

class Account
{
protected:
    int acc_no;
    double balance;

public:
    Account(int account_number, double balance);
    int getaccountno();
    int getbalace();
    void deposit(double amount);
    void withdraw(double amount);
    void display();
};
Account::Account(int account_number, double balance)
{
    this->acc_no = account_number;
    this->balance = balance;
}
int Account::getaccountno()
{
    return acc_no;
}
int Account::getbalace()
{
    return balance;
}
void Account::deposit(double amount)
{
    balance += amount;
}
void Account::withdraw(double amount)
{
    if (balance < amount)
    {
        cout << "======= Insufficient balance ======="
             << "\n";
        return;
    }
    balance -= amount;
}
void Account::display()
{
    cout << "   Account Number: " << acc_no << "\n";
    cout << "   Balance: $" << balance << "\n";
}

class saving_account : public Account
{
protected:
    double interest_rate;
    double interest;

public:
    saving_account(int acc_no, double balance, double interest_rate);
    double calculateInterest();
    void displaysaving();
};
saving_account::saving_account(int acc_no, double balance, double interest_rate)
    : Account(acc_no, balance)
{
    this->interest_rate = interest_rate;
}

double saving_account::calculateInterest()
```

```
{
    interest = balance * interest_rate / 100;
    balance += interest;
    return interest;
}
void saving_account::displaysaving()
{
    display();
    cout << "   Interest: " << calculateInterest() << "\n";
    cout << "   Interest Rate: " << interest_rate << "\n";
}

class checking_account : public Account
{
protected:
    int monthly_withdrawals;
    int max_withdrawals;
    double withdrawal_fee;

public:
    checking_account(int acc_no, double balance, int max_withdrawals, double withdrawal_fee,
int monthly_withdrawals);
    bool withdraw_Amount();
    void displaycheck();
};
checking_account::checking_account(int acc_no, double balance, int max_withdrawals, double
withdrawal_fee, int monthly_withdrawals)
    : Account(acc_no, balance)
{
    this->max_withdrawals = max_withdrawals;
    this->withdrawal_fee = withdrawal_fee;
    this->monthly_withdrawals = monthly_withdrawals;
}
bool checking_account::withdraw_Amount()
{
    if (monthly_withdrawals >= max_withdrawals)
    {
        return false;
    }
    else
    {
        balance -= withdrawal_fee;
        monthly_withdrawals++;
    }
}
void checking_account::displaycheck()
{
    display();
    cout << "   Monthly Withdrawals Balance: " << monthly_withdrawals << "\n";
    cout << "   Maximun Withdrawals: " << max_withdrawals << "\n";
    cout << "   Withdrawal Fee: " << withdrawal_fee << "\n";
}
class fixed_deposit_account : public Account
{
protected:
    double interest_rate;
    int term; // In months
    int monthsPassed;
    double interest;

public:
    fixed_deposit_account(int acc_no, double balance, double interest_rate, int term);
    bool withdraw();
    double cal_interest();
    void displayfix();
};
fixed_deposit_account::fixed_deposit_account(int acc_no, double balance, double interest_rate,
int term)
    : Account(acc_no, balance)
```

```
{
    this->interest_rate = interest_rate;
    this->term = term;
}

bool fixed_deposit_account::withdraw()
{
    if (monthsPassed < term)
        return false;
}

double fixed_deposit_account::cal_interest()
{

    interest = balance * interest_rate / 100;
    balance += interest;
    return interest;
}
void fixed_deposit_account::displayfix()
{
    display();
    cout << "   Interest: " << cal_interest() << "\n";
    cout << "   Interest Rate: " << interest_rate << "\n";
    cout << "   Term in Month: " << term << "\n";
}
int main()
{
    int option;
    int acc_no;
    double balance;
    double interest_rate;
    int max_withdrawals;
    double withdrawal_fee;
    int term;
    int monthly_withdrawals;

    cout << "===== Welcome to the Banking System! =====\n\n";
    cout << "   Please choose an account type:\n";
    cout << "     1. Savings Account\n";
    cout << "     2. Checking Account\n";
    cout << "     3. Fixed Deposit Account\n";
    cout << "   Enter your choice (1-3): ";
    cin >> option;

    switch (option)
    {
    case 1:
    {
        cout << "   Enter Account Number: ";
        cin >> acc_no;
        cout << "   Enter Initial Balance: ";
        cin >> balance;
        cout << "   Enter Interest Rate: ";
        cin >> interest_rate;

        saving_account savingsAccount(acc_no, balance, interest_rate);
        savingsAccount.displaysaving();

        double depositAmount;
        cout << "   Enter the amount to deposit: ";
        cin >> depositAmount;
        savingsAccount.deposit(depositAmount);

        double withdrawAmount;
        cout << "   Enter the amount to withdraw: ";
        cin >> withdrawAmount;
        savingsAccount.withdraw(withdrawAmount);

        savingsAccount.calculateInterest();
```

```
        savingsAccount.displaysaving();
        break;
    }

    case 2:
    {
        cout << "Enter Account Number: ";
        cin >> acc_no;
        cout << "Enter Initial Balance: ";
        cin >> balance;
        cout << "Enter Max Withdrawals: ";
        cin >> max_withdrawals;
        cout << "Enter Withdrawal Fee: ";
        cin >> withdrawal_fee;
        cout << "Enter Monthly Withdrawals: ";
        cin >> monthly_withdrawals;

        checking_account checkingAccount(acc_no, balance, max_withdrawals, withdrawal_fee,
monthly_withdrawals);
        checkingAccount.displaycheck();

        double depositAmount;
        cout << "Enter the amount to deposit: ";
        cin >> depositAmount;
        checkingAccount.deposit(depositAmount);

        checkingAccount.withdraw_Amount();

        checkingAccount.displaycheck();
        break;
    }

    case 3:
    {
        cout << "Enter Account Number: ";
        cin >> acc_no;
        cout << "Enter Initial Balance: ";
        cin >> balance;
        cout << "Enter Interest Rate: ";
        cin >> interest_rate;
        cout << "Enter Term (in months): ";
        cin >> term;

        fixed_deposit_account fixedDepositAccount(acc_no, balance, interest_rate, term);
        fixedDepositAccount.displayfix();

        double depositAmount;
        cout << "Enter the amount to deposit: ";
        cin >> depositAmount;
        fixedDepositAccount.deposit(depositAmount);

        fixedDepositAccount.cal_interest();
        fixedDepositAccount.displayfix();
        break;
    }

    default:
        cout << "Invalid choice. Exiting the program."
             << "\n";
        return 0;
    }

    return 0;
}
```

```
===== Welcome to the Banking System! =====

   Please choose an account type:
     1. Savings Account
     2. Checking Account
     3. Fixed Deposit Account
   Enter your choice (1-3): 1
   Enter Account Number: 12000
   Enter Initial Balance: 3000
   Enter Interest Rate: 6
   Account Number: 12000
   Balance: $3000
   Interest: 180
   Interest Rate: 6
   Enter the amount to deposit: 35000
   Enter the amount to withdraw: 13000
   Account Number: 12000
   Balance: $26690.8
   Interest: 1601.45
   Interest Rate: 6
```

Task 10

```cpp
#include <iostream>
#include <cstring>
using namespace std;

class entity
{
protected:
    string category;
    int time_of_departure;
public:
    entity(string category, int time_of_departure);
    ~entity();
};
entity::entity(string category, int time_of_departure)
{
    this->category = category;
    this->time_of_departure = time_of_departure;
}
entity::~entity()
{
}
class flight : public entity
{
protected:
    string plane;
    string pilot_name;
public:
    void time_of_departure();
    void availability();
    flight(string plane, string pilot_name, string category, int
time_of_departure);
    ~flight();
};
void flight::time_of_departure()
{
    cout << "Flight has departured\n";
}
void flight::availability()
{
```

```
    cout << "Seats are available in flight\n";
}
flight::flight(string plane, string pilot_name, string category, int
time_of_departure) : entity(category, time_of_departure)
{
    this->plane = plane;
    this->pilot_name = pilot_name;
}
flight::~flight()
{
}
class passengers : public entity
{
protected:
    string name;
    int phone_number;
    int seat_num;

public:
    void cancel();
    void ticket_info();
    passengers(string name, int phone_number, int seat_num, string category, int
time_of_departure);
    ~passengers();
};
void passengers::cancel()
{
    cout << "Passenger has canceled the ticket\n";
}
void passengers::ticket_info()
{
    cout << "   Name: " << name << "\n";
    cout << "   Class: " << category << "\n";
    cout << "   Seat: " << seat_num << "\n";
    cout << "   Departure Time: " << time_of_departure << "\n";
    cout << "   Contact: 03" << phone_number << "\n";
    cout << "   Pilot Name: "<< "Ali Abbas\n";
}
passengers::passengers(string name, int phone_number, int seat_num, string
category, int time_of_departure) : entity(category, time_of_departure)
{
    this->name = name;
    this->phone_number = phone_number;
    this->seat_num = seat_num;
}
passengers::~passengers()
{
}
class ticket : public entity
{
protected:
    string method;
    double price;
    int seat_num;

public:
    void cancel();
    void upgrade_cate(string cat);
    ticket(string method, double price, int seat_num, string category, int
time_of_departure);
```

```
    ~ticket();
};
void ticket::cancel()
{
    cout << "Ticket has been cancelled\n";
}
void ticket::upgrade_cate(string cat)
{
    cout << "Ticket " << seat_num << " has been upgraded to " << cat << "\n";
    category = cat;
}
ticket::ticket(string method, double price, int seat_num, string category, int
time_of_departure) : entity(category, time_of_departure)
{
    this->method = method;
    this->price = price;
    this->seat_num = seat_num;
}
ticket::~ticket()
{
}
int main()
{

    cout << " *********** Ticket Booking ***********\n";

    string name, category;
    int phone_number, seat_num, time_of_departure;
    double price = 15000;
    cout << "\n   Input Your full Name: ";
    getline(cin, name);
    cout << "   Input Phone Number: +92";
    cin >> phone_number;
    cout << "   Input Departure Time in 0000 hour format: ";
    cin >> time_of_departure;
    cout << "   Input seat number to Book: ";
    cin >> seat_num;
    cout << "   Input Class for Booking: ";
    cin.ignore();
    getline(cin, category);

    char confirm;
    cout << "Confirm Booking for Seat Number " << seat_num << "(y/n): ";
    cin >> confirm;
    if (confirm == 'y')
    {
        passengers p1(name, phone_number, seat_num, category, time_of_departure);
        ticket t1("Online", price, seat_num, category, time_of_departure);
        cout << "\n *********** Ticket Confirmed ***********\n";
        p1.ticket_info();
    }
    else
        cout << "Booking Cancel\n";

    return 0;
}
```

```
 ********** Ticket Booking **********

   Input Your full Name: Eman Murtaza
   Input Phone Number: +925844868
   Input Departure Time in 0000 hour format: 16000
   Input seat number to Book: 34
   Input Class for Booking: Standard
Confirm Booking for Seat Number 34(y/n): y

 ********** Ticket Confirmed **********
   Name: Eman Murtaza
   Class: Standard
   Seat: 34
   Departure Time: 16000
   Contact: 035844868
   Pilot Name:    Ali Abbas
```