Outlines

Apply OOP concepts to define a hierarchy of animals (mammals, birds, and fish)

- Define the parent class.
- Use inheritance to define child classes.
- Enable polymorphism where necessary.
- Use the STL vector to store pointers to animals.

To be returned

Create a compressed archive containing:

- A short report in PDF explaining what you did during this TP. The report should include:
    - Your names.
    - A brief introduction and conclusion.
    - Some screenshots of your code validation.
    - For each question, explain the choices made. For example: public or protected parts, include a graph of the class hierarchy, etc.
- Your source files (.h or .hpp and .cpp), where you have added comments.

Q1: Create the parent class `Animal`

We want to create a `Animal` class with two member variables: `name` and `weight`, and four member functions:

- `void print()`: prints member variable values.
- `void move()`: prints how the animal moves.
- `void type()`: prints what type of animal it is.
- `void info_displacement()`: prints information about what the animal is and how it moves.

The body of the last function is:

```
void Animal::info_displacement()
{
  cout << "The ";
  type();
  cout << " (" << name << ") ";
  move();
  cout << endl;
}
```

For this question, you need to properly encapsulate your member functions using the `public protected` access specifiers, and define a class constructor. You can also add access functions (getter and setter) to use or modify member variables if needed.

Q2: Define a class hierarchy using inheritance

Define 3 derived classes from `Animal`: `Mammal`, `Bird`, and `Fish`. For each of them, you can add some new member variables (such as the number of legs, nose length, or whether they live in saltwater or freshwater, etc.).

For all these child classes, you will need to redefine some functions from the parent class and the constructor.

(Bonus) If you have enough time, you can also define some exception cases by adding the derived classes: `Penguin`, `Ostrich` and `Dolphin`.

Q3: Enable polymorphism

Introduce the `virtual` and `override` keywords to enable polymorphism in your hierarchy. Doing so, you should be able to call the correct object function using a pointer to an `Animal`.

```
Animal* pA = new Bird("Coco", 1.5, 3);
pA->print();
```

Q4: Create a collection of `Animal`

Use the STL `vector` container to store pointers to `Animal`. Insert into it some instances of `Bird`, `Mammal`, `Fish` (`Penguin`, `Ostrich`, `Dolphin`) using:

```
  vector<Animal*> Zoo;
  Zoo.push_back(new Bird("Coco", 1.5, 3));
  ...
```

Then, in the file Zoo.cpp, implement the function `total_weight` that calculates the total weight of all the animals created. You should be able to execute this part of the code:

```
info_displacement(Zoo);
cout << "Total weight of the animals in the zoo is " << total_weight(Zoo)
<< endl;
```

This should produce output similar to:

```
The Bird (Coco) is flying
The Fish (Dory) is swimming
The Mammal (Wide_Ethelbert) is running
The Bird (Eiscue) is swimming
The Bird (Bipbip) is running
Total weight of the animals in the zoo is 108.8
```

Zoo.hpp:

```
#ifndef ZOO_HPP
#define ZOO_HPP

#include "animal.hpp"
#include <vector>
using namespace std;

double total_weight(const vector<Animal*>& zoo); // to implement
void info_displacement(const vector<Animal*>& zoo);
void invatory(const vector<Animal*>& zoo);

#endif
```

Zoo.cpp:

```
#include "zoo.hpp"

void info_displacement(const vector<Animal*>& zoo)
{
  for (int i = 0; i < zoo.size(); i++)
  {
     zoo[i]->info_displacement();
  }
}

void inventory(const vector<Animal*>& zoo)
{
  for (int i = 0; i < zoo.size(); i++)
  {
     zoo[i]->print();
  }
}
```

Before exiting the program, you need to free the memory allocated using the keyword `new` when inserting `Animal` into the vector. For that, use `delete` as follows:

```
for (int i = 0; i < Zoo.size(); i++)
  delete Zoo[i];
```