

Project 1: OOP and Inheritance

Aim: The aim of this project is to reinforce the concept of OOP and inheritance. There are two parts that need to be completed.

Part 1:

Write a class: **Animal**. Every Animal has a name, it may be domestic (or not) and it may be a predator (or not).

Part 2:

Write the following 3 classes that derive from Animal but **also have other attributes specific to their type**:

- Mammal
- Bird
- Fish

Implementation:

You must separate interface from implementation (both .hpp and .cpp files **for each class**) and implement the class based on the following specification. Function prototypes and member variable names must match exactly.

The Animal class has only accessor and mutator functions for its private data members. Recall that accessor functions (e.g. getName()) are used to access the private data members (e.g. all getName() will do is return name_), while mutator functions give a value to the data members.

As you implement the three classes for part 2, think about what is inherited and what is not (e.g. constructors are not inherited!!!). Also think about the order in which constructors are called, and how/where must you explicitly call the base class constructor. Constructors must initialize data members to 0 or false unless an argument is passed for that data member. You must write **Animal** to support inheritance and provide direct access from the derived classes to its private members.

Remember, **you must thoroughly document your code!!!**

class Animal public members:

```
Animal();  
Animal(std::string name, bool domestic = false, bool predator = false);  
std::string getName() const;  
bool isDomestic() const;  
bool isPredator() const;  
void setName(std::string name);  
void setDomestic();  
void setPredator();
```

class Animal private members:

```
std::string name_;  
bool domestic_;  
bool predator_;
```

class Mammal public members:

```
Mammal();  
Mammal(std::string name, bool domestic = false, bool predator = false);  
bool hasHair() const;  
bool isAirborne() const;  
bool isAquatic() const;  
bool isToothed() const;  
bool hasFins() const;  
bool hasTail() const;  
int legs() const;  
void setHair();  
void setAirborne();  
void setAquatic();  
void setToothed();  
void setFins();  
void setTail();  
void setLegs(int legs);
```

class Mammal private members:

```
bool hair_;  
bool airborne_;  
bool aquatic_;  
bool toothed_;  
bool fins_;  
bool tail_;  
int legs_;
```

class Bird public members:

```
Bird();  
Bird(std::string name, bool domestic = false, bool predator = false);  
bool isAirborne() const;  
bool isAquatic() const;
```

```

    void setAirborne();
    void setAquatic();

class Bird private members:
    bool airborne_;
    bool aquatic_;



---



class Fish public members:
    Fish();
    Fish(std::string name, bool domestic = false, bool predator = false);
    bool isVenomous() const;
    void setVenomous();

class Fish private members:
    bool venomous_;

```

Testing:

You must always implement and test your programs **INCREMENTALLY!!!**

What does this mean?

- Implement and test one class at a time!!!
- For each class:
 - Implement one function/method and test it thoroughly (multiple test cases + edge cases if applicable)
 - Implement the next function/method and test it ...

How do you do this?

Write your own **main()** function to test your classes. In this course you will never submit your test program but you must always write one to test your classes. As you implement each Animal method, instantiate objects of type Animal in main and call the method to test it is correct. Start from the constructor(s), then move on to the other functions. You may output the return values to inspect your program's behavior. Choose the order in which you implement your methods so that you can test incrementally (i.e. implement mutator functions before accessor functions). Sometimes functions depend on one another. If you need to use a function you have not yet implemented, you can use **stubs**: a dummy implementation that always returns a single

value for testing (don't forget to go back and implement the stub!!! If you put the word STUB in a comment, some editors will make it more visible so you will remember to implement it later)

For example:

```
//***** STUB *****/
bool Animal::isPredator() const
{
    return false;
}
```

Note: this will make much more sense as your programs become more complex, but it is very important to understand the fundamental concepts and develop good implement/test/debug habits from the very beginning.

Once you are done with the Mammal class, you can move on to implementing Bird, then Fish.

In your main function you also want to test for inheritance. Think about:

- **Can you access members of the base class from the derived class? Test it!!!**
- **Test calling a member function of the base class via an object to type derived. Make sure it works!**

Grading Rubric:

- **Correctness 80%** (distributed across unit testing of your submission)
 - A submission that implements all required classes and/or functions but does not compile will receive 40 points total (including documentation and design).
- **Documentation 10%**
- **Style and Design 10%** (proper naming, modularity and organization)

Submission:

You will submit **the following files**:

- The Animal class (Animal.hpp and Animal.cpp)
- The Mammal class (Mammal.hpp and Mammal.cpp)
- The Bird class (Bird.hpp and Bird.cpp)

- The Fish class (`Fish.hpp` and `Fish.cpp`)

Your project must be submitted on Gradescope.

Although Gradescope allows multiple submissions, it is not a platform for testing and/or debugging and it should not be used for that. You **MUST** test and debug your program locally. Before submitting to Gradescope you **MUST** ensure that your program compiles (with g++) and runs correctly on the Linux machines in the labs at Hunter (see detailed instructions on how to upload, compile and run your files in the "Programming Rules" document). That is your baseline, if it runs correctly there it will run correctly on Gradescope, and if it does not, you will have the necessary feedback (compiler error messages, debugger or program output) to guide you in debugging, which you don't have through Gradescope. "But it ran on my machine!" is not a valid argument for a submission that does not compile.

Once you have done all the above you submit it to Gradescope.

The due date is Tuesday February 18 by 5pm. No late submissions will be accepted.