Project 1: OOP - a very simple class

Project1 is trivial and simply a starter project to establish basic ideas of **OOP** and ensuring you can successfully submit to Gradescope.

You will write 1 very simple class: **`Animal.`** Every Animal has a name, it may be domestic (or not) and it may be a predator (or not).

Implementation:

You must separate interface from implementation (.hpp and .cpp files) and implement the class based on the following specification (FUNCTION PROTOTYPES AND MEMBER VARIABLE NAMES MUST MATCH EXACTLY). This 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.
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_;
```

Testing:

This project is very basic, but it is never too early to set in good debugging and testing habits. 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.

Grading Rubric:

- **Correctness 80%** (distributed across unit testing of your submission)
- **Documentation 10%**
- **Style and Design 10%** (proper naming, modularity and organization)

Submission:

You will submit **2 files**: `Animal.hpp` and `Animal.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 Friday September 6 by 5pm. No late submissions will be accepted.

Have Fun!!!!!

Image credits: https://www.iconfinder.com