CS 303
Summer 2013
Program 1 - Inheritance

The first programming project of the summer is an ecosystem simulation. We will use an object oriented approach to take advantage of similarities in behavior across several objects.

An ecosystem can be modeled something like this:

Organism
|
- Plant
|
- Animal
|
- Herbivore
|
- Carnivore

At the highest (most abstract) level, we have an organism. All organisms are born, grow, and die; they have a mass. This is the base class for the entire ecosystem.

Plants are a specific type of organism. In addition to the behavior of all organisms, plants can be nibbled on by herbivores. For purposes of this assignment, all plants are identical.

Animals are another type of organism; Animal is an abstract class. In addition to the traits common to all organisms, animals have a maximum size, beyond which they will not grow. In general, animals that are below the maximum weight that get enough to eat will grow an amount proportional to the amount eaten. Animals can also reproduce. (Plants also reproduce, of course, but we're not going to model that in our program.) For our program, there are two types of animals:

Herbivores eat plants. In fact, herbivores must eat twice their own weight in plants every week, or they will starve to death.

Carnivores eat Herbivores. Carnivores must eat their own weight in Herbivores each week, or they starve.

Our program models a simple field with a fixed number of plants. Initially we have a certain number of herbivores and a smaller number of carnivores. This is a *discrete-time* simulation; that is, time advances in discrete 'ticks,' with all events happening within one tick to be considered more-or-less simultaneous. Your program will model a 50 week 'season' (we're not going to worry about actual seasonal changes). For each week, your program will report the total plant mass (in tons), the number of herbivores and their average weight, and the number of carnivores and their average weight.

Details:

- Implement the following classes. Think carefully about what should be public, private, or protected:
  - Organism. An Organism has public methods Birth(StartMass), Die(), GetMass(), SetMass(),

IsAlive(), and Grow(). An Organism has a boolean variable Alive, and a float or double called Mass. IsAlive() is a const method returning the current value of IsAlive. Birth() sets Alive to true and Mass to the starting mass (if the starting mass is > 0; otherwise it's set to 1). Die() sets Alive to false and Mass to 0. GetMass() reports the current value of Mass. SetMass() assigns a new value to Mass. Passing a value less than 0 to SetMass() throws a EcosystemError exception (an exception class defined for this program).

- Plant. A Plant is a public child of Organism. A Plant has a default starting size of 10. A Plant also has the methods:
  - Sprout(), which sets Alive to true and Mass to 1;
  - GetNibbled(). The amount nibbled by a herbivore is 25% of the mass of the plant. If the plant is alive, then its Mass is reduced by the amount eaten, and the amount eaten is returned; if the Mass of the plant afterwards is less than 0.5, the plant dies. If the plant is already dead when this method is called, it returns 0.
  - Plants grow as follows: If the Mass is less than or equal to 5, the Mass doubles; if the Mass is greater than 5 but less than or equal to 10, the Mass increases by 50%; otherwise the Mass increases by 10%.
- Animal. This is also a public child of Organism. An Animal has public methods to Eat(float Amount), a void method called StartWeek() described below, and a const method GetAmountEaten() that reports the amount eaten that week. An Animal also has a maximum size it can grow (unlike plants, which under this simple model can be arbitrarily large), and protected methods to set and get the maximum size. An Animal also has a private float to record the amount eaten each week. The StartWeek() method sets the amount eaten to 0. Each call to Eat(amount) increases the amount eaten. An Animal will continue eating until the amount eaten equals or exceeds the amount required for that week.
- Herbivore. This is a public child of Animal. The only additional method a Herbivore has is called Graze. It takes a reference to a Plant as its only parameter. If the amount eaten this week is less than the amount required, Graze() calls the GetNibbled() method of the plant, and passes the amount nibbled to the Eat method. A Herbivore has a maximum size of 80.
- Carnivore. This is another public child of Animal. The only additional method a Carnivore has is Hunt(), which takes a reference to a Herbivore as its only parameter. The logic is similar to a Herbivore's grazing: If the amount eaten is less than required, the Herbivore's mass is added to the amount eaten, and the Herbivore dies.

- The main program works as follows:
  - Our ecosystem starts with 30,000 plants. Use a vector to store these.
  - The ecosystem starts with 100 herbivores (rabbits), and 5 carnivores (hawks). Again, use a vector to store these.
  - Repeat for 50 weeks:
    - Plants grow
    - Herbivores graze. Each herbivore will graze on 50 randomly selected plants. If the Herbivore has enough to eat before 50 plants have been nibbled, then the Herbivore stops grazing. This can be done by ending the loop early, or by writing the Graze() method to return immediately if the Herbivore has eaten enough. If the Herbivore has grazed on 50 plants and not got enough to eat, the Herbivore dies and is removed from the vector.
    - Carnivores hunt. Each Carnivore gets 15 chances to hunt. The probability of any hunt being successful is the number of Herbivores divided by 5. If successful, the Carnivore eats a randomly-selected Herbivore, which is removed from the Herbivores vector. If the Carnivore does not get enough to eat, the Carnivore dies and is removed from the

Carnivore vector.
    - All surviving Animals grow. (For those not yet at their maximum size, mass increases by 75% of the amount eaten.)
    - Reproduction. The number of Herbivores increases by 20% each week. A fractional result should be taken as a probability. For example, if there are 53 Herbivores, 53*0.2 = 10.6. There will be 10 Herbivores born, and a 60% probability of an 11th. The number of Carnivores increases by 10% each week, using the same logic. If the number of Carnivores is 0, it is reset to 1 (something wanders in from the surrounding area).
    - Add to output data: Week number, number of herbivores, number of carnivores, total plant mass in pounds.
- Your program will be graded not just for correctness, but also for the design choices you make, which you should discuss in the program documentation. You should use C++ abstraction tools where possible: vectors rather than static arrays, iterators rather than an integer index, etc.

Submit:

- Header files for Organism, Animal, Herbivore, Carnivore classes, along with any other classes you write for this program. Note: You MUST name your classes and public methods as specified in the assignment!
- Main program file.
- Output from a typical program run. Your output file should be formatted for readability—line after line of numbers will NOT be adequate.

Further directions:

- Modify the program so there are no predators. What happens to the Herbivore population?