CSE 3150 — C++ Essentials — Yufeng Wu — Fall 2022
Programming Assignment Week 8: Life Simulation. Due: 11 pm @ 11/3/2022.

---

**Note**: this assignment is due on Thursday Nov 3. I am giving you one more week since we will have an exam during the 9th week. **Note**: this assignment can be more complex than previous ones. So please start early.

Imagine you are working for a software maker for a cartoon-based simulation game production. You are to design several classes for simulating living organisms. Life is complex. As you can see from the following specification, there are quite a bit of details to take care. To make things manageable, we have to apply some simplifications in the following. **Caution**: you must read the document very carefully. Otherwise, your code won't pass the test cases. This assignment is about implementing an eco-system with certain rules. These rules are fully specified and deterministic. That is, correct code should produce exactly the same results. If you spot some ambiguities somewhere, please post in Piazza.

To let you practice C++ design, I am only going to provide the high-level requirements. Different from previous assignments, the starter files are essentially empty. You need to design the class headers yourselves. We will release some test cases so that you know what interfaces you have to support. This way, you can practice to build something with complexity from ground up.

## What is the point of this assignment?

The main reason I want you to work on this assignment is, object orientation or OO (e.g., polymorphism and inheritance) in C++ works well to deal with complex situations like life simulation. As we know, there are many forms of life (biological diversity or polymorphism). That fits well for the polymorphism we use in C++. I hope you carefully plan your class hierarchy. Try to make your code simple and easy to maintain. **Note**: try to reduce the duplicate code as much as you can! This will make it easier for both writing the code and also debugging.

This assignment may look long. But once you understand it and know how to do OO design, it should not be too difficult...

## 1 ECOrganism: the base class

ECOrganism is the base class for all organisms (bugs, fox, you name it) that you are to simulate. This class should have the following functionalities. Each activity is performed at a certain time (represented by hours of the day).

1. Vitality. Each entity has a name (in string) and more importantly *vitality* level (between 0 to 100, in floating points), Initially vitality is at 100 (maximum). Often activities performed by the entity will consume energy, while "Recharge" can increase energy level. Different entities have different activities to perform (see below). As for recharge, the only way to increase vitality is by "eating". If vitality reaches zero, the organism dies. When the organism is not eating or working or sleeping, it will juse doze (which doesn't consume vitality).

2. Eat. An organism can eat to increase vitality. Note: there is a maximum number of times for the organism to eat to recharge. When that number is reached, it can no longer recharge. Also, if vitality is zero, it cannot eat. Also, an organism can eat even when vitality is at 100 (this is like do nothing but waste its precious life). The Eat method takes the time (hours of the day). Eat is instantaneous.

3. Sleep: do nothing and there is *no* loss of vitality during sleep. For simplicity, an organism of certain kind (say spider) has the same sleeping period. For example, spider sleeps from 9 to

17 (i.e., 5 pm) during the day. When sleeping, the organism won't respond to anything (i.e., won't initialize any activity). However, it can still get attacked (see below).

4. Work. Each organism has something it wants to do. Exactly what it does depends on its kind (see below). As long as vitality is not zero, it can work. If after the work, vitality is exhausted, it dies. A few details about work.
   (a) Work is instantaneous.
   (b) The Work method takes the time (hours of the day).
   (c) Often there are multiple kind of work to do. We use a work type (an integer) to refer to the kind of work to do.
   (d) It is possible that requested work cannot be done (e.g., the needed target wasn't set properly or the target is already dead). In this case, the work request is voided with no loss of vitality.
   (e) Sometimes the organism tries to do something out of its ability (e.g., spider trying to trap a fox). This won't affect the target but still cost vitality of the organism( e.g. spider).
5. Organism-specific activity. An organism can have its own things to do, which involves another organism. In the interface, such activity has interface like: *void DoSomething(ECOrganism &rhs);* . Here, *rhs* is the subject of this activity. See below for more details.

We now describe specific organisms.

# 2 Arthropods

Arthropods refer to species such as insects or spiders. The following lists three kinds of arthropods to simulate. There are some common features of all arthropods.

1. Sleep: all arthropods sleep from 9 to 17 of the day.

## 2.1 Spider

Spider is simulated in ECSpider class.

1. Eat: gain 20 vitality each time. Eating limit: 5.
2. Work (type 0): weave the web and cost 10 vitality each time.
3. Work (type 1): trap a prey; use its web to catch its prey. Note: any (and only) arthropod caught by spider is dead instantly. For other organism, this has no effect at all. The prey will be passed in constructor of ECSpider, and won't change. If a NULL pointer is passed, then the spider is not hunting. Cost: 30 vitality.

## 2.2 Grasshopper

Grasshopper is simulated in ECGrasshopper class.

1. Eat: gain 25 vitality each time. Eating limit: 10.
2. Work (type 0): hop around and cost 5 vitality each time.
3. Work (type 1): bite; use its jaws to hurt the prey. Note: any (and only) arthropod bitten by grasshopper is dead instantly. If a mouse is bitten by a grasshopper, the mouse loses vitality by 5. Other than this, bitting has no effects. The prey will be passed in constructor, and won't change. If a NULL pointer is passed, then it is not biting. Cost: 40 vitality.

### 2.3 Caterpillar

Caterpillar is simulated in ECCaterpillar class.

1. Eat: gain 15 vitality each time. Eating limit: 5.
2. Work: climb the tree and cost 10 vitality each time. This is the only type of work.

## 3 Mammal

There are some common features of all mammals.

1. Eat: all mammals gains 50 vitality each time.
2. Work: all mammals "run" as their type-0 work, which costs 10 each time.

### 3.1 Mouse

Mouse is implemented in ECMouse class.

1. Limit of eat: 10 times.
2. Sleep: from 10 to 18 hours of the day.
3. Work: (type 1): catch a prey. Mouse is only interested in grasshopper. For other organism, it has no appetite at all. The prey will be passed in constructor, and won't change. If a NULL pointer is passed, then it is not hunting. Cost: 20 vitality.

### 3.2 Fox

Fox is implemented in ECFox class.

1. Limit of eat: 8 times.
2. Sleep: from 7 to 14 hours of the day.
3. Work: (type 1): catch a prey. Fox is only interested in mouse. For other organism, it has no appetite at all. The prey will be passed in constructor, and won't change. If a NULL pointer is passed, then it is not hunting. Cost: 40 vitality.

### 3.3 Panda

Panda is implemented in ECPanda class.

1. Limit of eat: 12 times.
2. Sleep: from 15 to 10 hours (of the next day).
3. No other work: Panda is lazy!

## 4 Simulator

At last, you need to implement a simple simulator called ECLifeSimulator. Here are the basic functionalities of this simulator. For the ease of testing, we will release this interface in the starter code.

1. Initialize. Clear out all previous user settings to start a fresh simulation.
2. Add an organism. Note: each organism will be later referred by its index (the zero-based position index based on the order of insertion).
3. Add a daily event: a specific organism will do some activities, either work (along with the type of work) or eat, at certain time of the day. For simplicity, an organism performs the same thing every day.
4. Collect information on the organisms being simulated. See the header file for what are to be implemented. This part is to be used in testing.
5. Simulation. Run the simulation, starting at certain time (in hours unit), until no alive organisms or reaching the specified time limit. Return the length of the simulation. You should run the simulation through multiple days if needed. Note: the scheduled activities are daily based, which means an activity can perform multiple times. Some more explanations on how simulation should be performed to pass the test cases.
   (a) Simulation is to continue day after day. That is, hour 24 of the previous day is the same as hour 0 of the current day.
   (b) If multiple activities are at exactly the same time, we perform these activities in the order of the position index (i.e., which organisms added earlier get preference).
   (c) Recall each activity is assumed to occur instantaneously.
   (d) Also recall that each activity is to be performed at the same time each day.

## 5 Instructions for implementation and submission

I know the above seems to be a little complex. I recommend you to apply **divide and conquer** scheme: start with the generic organism and one concrete organism (say spider); once you have these two classes, implement the simulator and add some test cases to ensure it can work; then move on to implement other classes. When working on a class, work on the most fundamental functionalities first before working the others.

We provide starter files for each class as described above. These files are essentially empty. Don't create new files. You can (and often should) create new classes. In that case, place the new classes in ECOrganism (header and source).