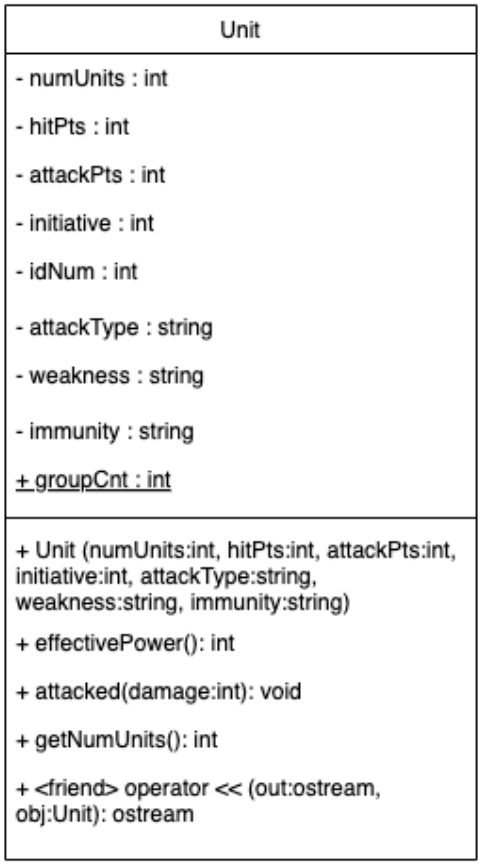
UML to Code – Immune System Simulation Part 2

In this assignment you will implement a design that is specified using a UML class diagram.

In completing this assignment, you will:

- Gain more experience understanding how a class design is represented in UML
- Apply what you have learned about converting a UML class diagram to C++ code

Background



You might recall the UML class diagram to the right which represents a program that stores details about Units in an immune system simulation.

The given code (unit.h and unit.cpp) implements this original diagram. For the final project you be adding a few methods and two subclasses, Immune and Infection.

You will be adding:

`int getId()` which returns the idNum.

`int getInitiative()` which returns the initiative.

`int getDamage(Unit& obj)` which returns the damage which could be one of three numbers:

If the attackType is equal to the obj's immunity then the damage is zero because the object is immune to that attack.

Else if the attackType is equal to the obj's weakness then the damage is double the effectivePower because the object is weak to that attack.

Else it is a regular attack so return the effectivePower.

You will be overloading the < and > operators:

`bool operator>(const Unit &obj) const` if the effective power is equal to the obj's effectivePower then return true if the initiative is greater than the obj's initiative. Else return true if the effectivePower is greater than the obj's effectivePower.

`bool operator<(const Unit &obj) const` returns true if the initiative is less than the obj's initiative.

Immune

The Immune class has a constructor and overrides `string getTypeOf()` to return "immune"

Infection

The Infection class has a constructor and overrides `string getTypeOf()` to return "infection"

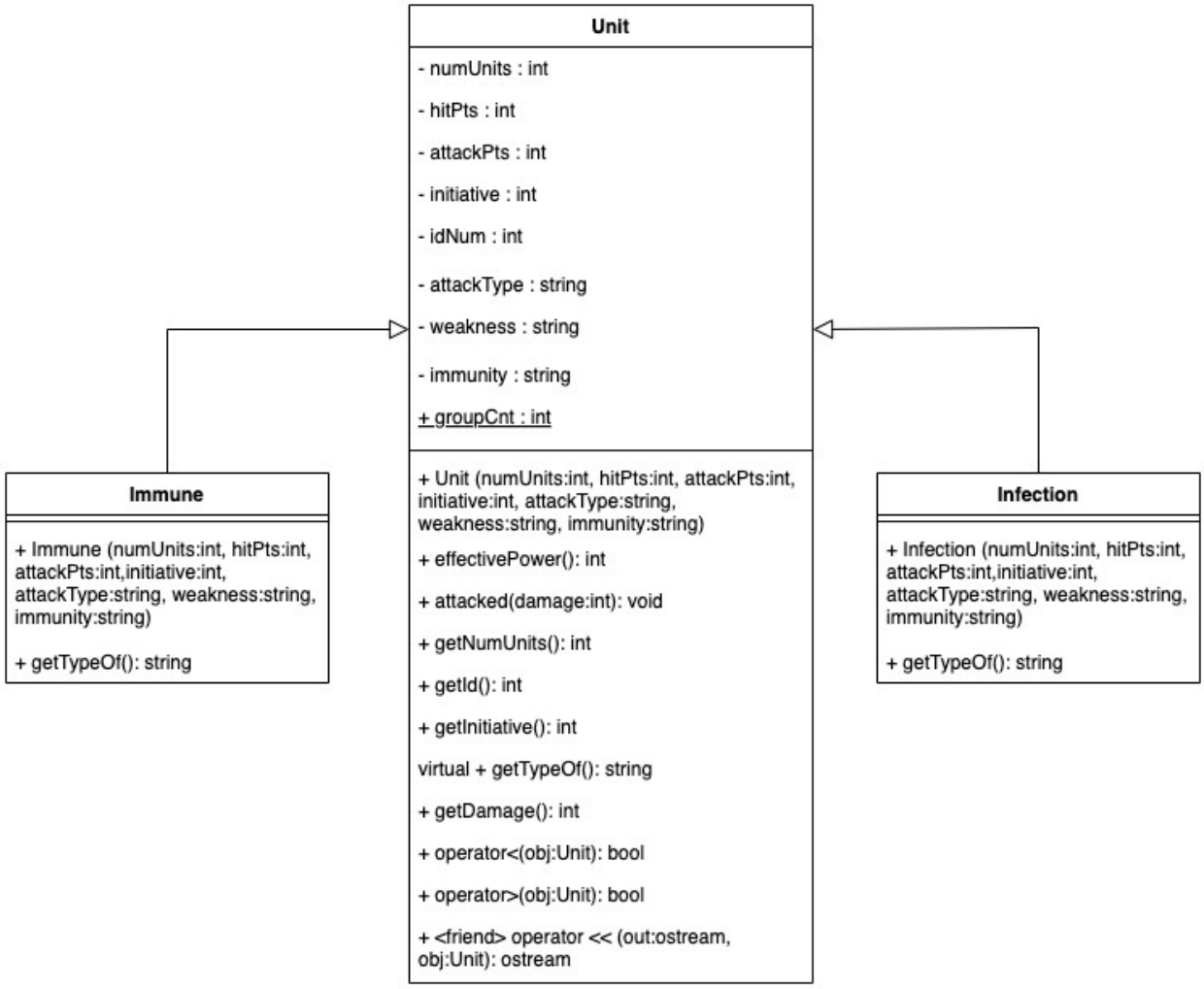
Adjustments:

You will be editing `void attacked(int damage)` to throw an error if the numUnits is less than or equal to zero after an attack.

You will also be editing the overloaded output friend function (<<) that outputs the current state of the Unit. For the an immune type, it should print exactly like below (Hint: call getTypeOf()):

```
Type:immune
Units:18
HitPts:729
Weakness:fire
Immunities:cold
AttackPts:8
AttackType:radiation
Initiative:10
ID:0
```

The new UML looks like this:



Activity

Implement the design in C++ by creating classes and methods according to the class diagram and following these guidelines:

- You should not have a main or other classes/methods/functions; just represent the class as shown above. Though you should create a main in a separate file for testing, you will not be submitting it.
- All fields and methods' privacy should follow the diagram (that's what the little plus-sign and minus signs next to the method names indicate).
- All of your classes should be in the same file for this assignment (`unit.h` and `unit.cpp`).

Helpful Hints

- Underlined pieces indicate a static member.
- In addition to setting all of the incoming variables, the constructor should set idNum equal to groupCnt and then increment groupCnt.
- The header file should have only the class definition given in the UML.

Before You Submit

Please be sure that:

- your code compiles and outputs the required text.

Assessment

Your submission will be assessed using automatic grading scripts that will check that your implementation matches the specified design, specifically that each class:

- implements the correct interfaces
- has the correct number of fields and methods
- has fields of the correct name, type, and multiplicity
- has methods of the correct name, parameter type(s), and return type

Your score is determined by the percentage of these tests that "pass," i.e. that correctly implement the various aspects of the design.

Before submitting your solution, you can run these grading scripts locally on your computer by following the steps below. These instructions assume you are running them on the server, but should be applicable to other environments as well:

1. Download the makefile `makefile`
2. Download the main file `main.cpp`
3. Upload the files into the same director as your cpp and header files for this assignment (eg in `cisp400/hw15`)
4. You can compile your file using `make`
5. Now run the tests by typing `make grade`

Note: your score is only an approximation of your actual score, there may be other, hidden, test cases that I will run when grading this officially.

Turn-in Procedure

Submit the class you created and implemented (`unit.h` and `unit.cpp`) on Canvas as an attachment.

Adapted from: https://prod-edx-prospectus.edx.org/course/data-structures-and-software-design and https://adventofcode.com/2018/day/24