Programming Assignment 5: *The Creature Feature*

COP 3330 - Fall Term 2007
Point Value: 100 points
Project Due Date: Wednesday November 14, 2007

Learning Objectives

- To utilize inheritance to set up and work with a group of related classes
- To implement a simple fantasy role-playing game

Discussion: Creatures

Suppose that you are creating a fantasy role-playing game. In this game we have four different types of creatures: humans, cyberdemons, balrogs and elves. To represent one of these creatures we might define a Creature class as follows:

```
class Creature
{
    private:
        int type;               // 0 human, 1 cyberdemon, 2 balrog, 3 elf
        int strength;           // how much damage it can inflict
        int hitpoints;          // how much damage it can sustain
        string getSpecies();    // returns type of species
    public:
        // initialize to human, strength 10, hit points 10
        Creature();

        // initialize to new type new, strength, and new hit
        Creature(int newType, int newStrength, int newHit);

        // also add appropriate accessor and mutator functions
        // for type, strength, and hit points

        // returns amount of damage this creature inflicts in one round of combat
        int getDamage();
}
```

Here is an implementation of the getSpecies() function:

```
string Creature::getSpecies()
{
    switch(type)
    {
        case 0: return "Human";
        case 1: return "Cyberdemon";
        case 2: return "Balrog";
        case 3: return "Elf";
    }
    return "Unknown";
```

```
}
```

The getDamage() function outputs and returns the damage this creature can inflict in one round of combat. The rules for calculating the damage are as follows:

- Every creature inflicts damage that is a random number r, where 0 < r <= strength.
- Demons have a 5% chance of inflicting a demonic attack which is an additional 50 damage points. Balrogs and Cyberdemons are demons.
- With a 10% chance, Elves inflict a magical attack that doubles the normal amount of damage.
- Balrogs are very fast, so they get to attack twice.

An implementation of getDamage() is given below:

```
int Creature::getDamage()
{
    int damage;

    // all creatures inflict damage which is a random number up to their strength
    damage = (rand() % strength) + 1;
    cout << getSpecies() << " attacks for " << damage << " points!" << endl;

    // demons can inflict damage of 50 with a 5% chance
    if ((type == 2) || (type == 1))
        if ((rand() % 100) < 5)
        {
            damage = damage + 50;
            cout << "Demonic attack inflicts 50 additional damage points!"
                 << endl;
        }

    // elves can inflict double magical damage with a 10% chance
    if (type == 3)
    {
        if (rand() % 10) == 0)
        {
            cout << "Magical attack inflicts " << damage
                 << " additional damage points!" << endl;
            damage = damage * 2;
        }
    }

    // balrogs are fast so they get to attack twice
    if (type == 2)
    {
        int damage2 = (rand() % strength) + 1;
        cout << "Balrog speed attack inflicts " << damage2
             << " additional damage points!" << endl;
        damage = damage + damage2;
    }
    return damage;
}
```

One problem with this implementation is that it is unwieldly to add new creatures. Rewrite the class to use inheritance, which will eliminate the need for variable *type*. The Creature class will be the base class. The classes Demon, Elf and Human will be derived from Creature. The classes Cyberdemon and Balrog will be derived from Demon. You will need to rewrite the getSpecies() and getDamage() functions so they are appropriate for each class.

For example, the getDamage() function in each class should only compute the damage appropriate for that object. The total damage is then calculated by combining the results of getDamage() at each level of the inheritance hierarchy. As an example, invoking getDamage() for a Balrog object should invoke getDamage() for the Demon object which should invoke getDamage() for the Creature object. Ths will compute the basic damage that all creatures inflict, followed by the random 5% damage that demons inflict, followed the by double damage that balrogs inflict.

Also include the mutator and accessor functions for the private variables. You will write an application program that contains a driver to test your classes. It must create an object for each type of creature and repeatedly output the results of getDamage(). See below for a discussion of the driver.

This program is adapted from assignment #9 on page 644 in chapter 14 of the required course textbook.

Driver/Application Game and Extra Credit Points

You must write the classes as described, and you must also write a simple game to demonstrate that the classes work correctly.

As a minimum you must implement a simple game as follows. You can receive full credit (100 points) for implementing this simple version of a game.

Read in a file which records the initial state of the game. This file will contain the current state of first 4 "good guy" creatures and then 4 "bad guy" creatures.

Engage the good guys and the bad guys in a battle. The bad guys are first the attackers. The good guys are first the defenders.

Start by having the first bad guy you read in attack the first good guy you read in. Then have the second bad guy attack the second good guy, and so on, until 4 attacks have been processed. This constitutes one battle.

Repeat this process until one "team" has run out of creatures (that is, all creatures have been killed). After each battle, swap who attacks and who defends (for example, in battle 2, the good guys attack and the bad guys defend).

A creature dies when the damage to it exceeds it hit points. If that occurs, set the hit points for that creature to zero.

When an attack occurs, the defender does not do anything (except, get attacked!). The damage inflicted is subtracted from the defender's hit points. Thus damage does accumulate over time.

If you do not reach a point where one team has no more creatures left, then terminate the game after 20 battles.

Extra credit:

We will allow up to 10 extra credit points on this program. This will be at the discretion of the graders. If you implement a really nifty game, above and beyond the basics described above, you may receive from 1 to 10 extra credit points on this program. For example, you could create an interactive game where you play against the computer. No matter what, you must fully demonstrate and test your classes.

If you do implement a game other than the basic one described here, but sure to COMMENT

THOROUGHLY and explain completely how your game works!!! The graders must be able to tell very clearly and easily, what you are implementing.

Input

We will provide one sample data file on the class web site. You should definitely also design your own data files for testing. It will require more than one file to test your program to see if it works correctly.

Data files will contain a single line for each creature. For example

1 20 80

represents a cyberdemon (type 1) with strength 20 and hitpoints 80.

Output

As always, follow the course style guidelines, echoprint all input and confirm all operations.

Overview of the module structure

The finished project will consist of the following modules, which must be implemented in separate source files:

creatureApp.cpp

This module contains the main function and its subordinate functions, and is responsible for all aspects of the task which are not specified in the other modules.

Creature (creature.h and creature.cpp) interface and implementation of *creature* class

Demon (demon.h and demon.cpp) interface and implementation of *demon* class

Elf (elf.h and elf.cpp) interface and implementation of *elf* class

Human (human.h and human.cpp) interface and implementation of *human* class

Cyberdemon (cyberdemon.h and cyberdemon.cpp) interface and implementation of *cyberdemon* class

Balrog (balrog.h and balrog.cpp) interface and implementation of *balrog* class

Other Requirements and Some Notes

You can complete this project using the information in *chapters 1 through 14* of the textbook and of course your knowledge from pre and co-requisite coursework.

To fully test your program, use the principles of program testing, test cases and test plans, as discussed in class.

You may find it convenient to use virtual functions and polymorphism from chapter 15 in functions such as getDamage() and getSpecies().

The Demon class should be abstract, since there will be no actual instances of it.

You may add members to the classes as described here, but you may not delete anything from the classes or change the functionality of the class.

What To Turn In and How to Turn in Your Work Using Blackboard

You must turn in thirteen C++ program files, which must be named as described above . You must tar these files into a **SINGLE .tar file** for your submission to Blackboard. Name the single file that you turn in to Blackboard ***creature.tar***.

Refer to the course handout on basic UNIX commands for instructions on how to generate a tar file.

Refer to the handout entitled "Submitting Your Program Assignments Electronically Using Blackboard" for complete instructions on how to submit and how to check your submission after you submit it. This handout is available on the course web site under "Handouts."

Last Update: A. Ford Tyson 11:11 AM 10/26/2007
