CS 11 Data Structures and Algorithms

Assignment 7.2: Inheritance 2

Skip to Main Content

Learning Objectives

After the successful completion of this learning unit, you will be able to:

- Define syntactically correct classes that use inheritance in accordance with good programming practice
- Explain and implement polymorphism, virtual functions, abstract base classes, and pure virtual functions.

Note: You may omit documentation on all parts of this assignment. Just a comment with your name in it in the client program, please.

Assignment 7.2 [45 points]

It's messy that in the Balrog class's getDamage() function and the Cyberdemon class's getDamage() function we have to write the name of the species before calling the demon class's getDamage() function. It would be better if the demon class's getDamage() function could print the name of the species. Taking this a step further, it would be even better if we didn't have to repeat the cout statement "The <whatever> attacks for ?? points!" in every class's getDamage() function. It would be better if that cout statement could occur just once, in the Creature class's getDamage() function.

1. In the Creature class's getDamage() function, insert the following statement:

   ```
   cout << "The " << getSpecies() << " attacks for " << damage << " points!" << endl;
   ```

2. Delete (or, if you prefer, comment out) the similar cout statements that appear in the getDamage() function of each of the 5 derived classes. (There will be one such cout statement to delete in each of the 5 getDamage() functions.)

3. Try executing the program. The results won't be quite what we were hoping for.

4. Now make the getSpecies() function in the Creature class a virtual function, and execute the program again. The results will now be correct.

5. We can now simplify our derived classes even further. Two of the five derived classes have getDamage() functions that do nothing more than call their parent class's getDamage() function. Delete these two functions. (Don't forget to delete both the prototype in the class declaration and the definition.) We don't need them, because they can just inherit the getDamage() function from the Creature class.

6. You may have noticed that the Creature class's getSpecies() function never gets called. However, it is absolutely critical that any class that is derived from the Creature class define a getSpecies() function since that function is called from the Creature class's getDamage() function. The best way to implement this is to make the Creature class's getSpecies() function a pure virtual function, so that every class that is derived from the Creature class will be required to implement a getSpecies() function. Make the Creature class's getSpecies() function a pure virtual function.

7. Comment out the getSpecies() function in the Human class and try compiling the program to see what happens. then uncomment it (i.e., return it to it's previous state).

8. Make getDamage() a virtual function. This will be important so that in your "battleArena" function (see below) you can say "Creature1.getDamage()" and the damage will automatically be calculated for the

correct Creature. Note that the parameters for "battleArena" will be of type "Creature" and they will need to be pass-by-reference. (You might try making them pass-by-value to see what happens.)

9. Make a function in your client program that is called from your main function, battleArena(Creature &Creature1, Creature& Creature2), that takes two Creature objects as parameters. The function should calculate the damage done by Creature1, subtract that amount from Creature2's hitpoints, and vice versa. (When I say "subtract that amount from Creature2's hitpoints, I mean that the actual hitpoints data member of the Creature2 object will be modified. Also note that this means that both attacks are happening simultaneously; that is, if Creature2 dies because of Creature1's attack, Creature2 still gets a chance to attack back.) If both Creatures end up with 0 or fewer hitpoints, then the battle results in a tie. Otherwise, at the end of a round, if one Creature has positive hitpoints but the other does not, the battle is over. The function should loop until either a tie or over. Since the getDamage() function is virtual it should invoke the getDamage() function defined for the appropriate Creature. Test your program with several battles involving different Creatures. I've provided a sample main function below. Your only remaining task is to write the "battleArena" function and expand the main function so that the "battleArena" function is tested with a variety of different Creatures.

```
int main()
{
        srand((time(0)));

        Elf e(50,50);
        Balrog b(50,50);;

        battleArena(e, b);
}
```

Make sure that when you test your classes you see examples of the Elf doing a magical attack and the Balrog doing a demonic attack and also a speed attack.

Don't forget you need to #include <ctime> and #include <cstdlib>

10. All of the classes should still be in the cs_creature namespace

Assignment 7 Reflection [0 points]

You won't submit anything or be graded for this but I think it is an important part of the assignment so I made it a separate part.

Take some time to reflect on the program you just completed from the perspective of the base class, Creature. With respect to inheritance, you'll see 3 categories of member functions illustrated (not counting constructors, since they are not inherited).

1. The accessors and mutators are typical member functions that are inherited and never redefined. For example, the balrog class's getDamage() function calls getStrength(), which it can do because it has been inherited from the Creature class (via the Demon class).

2. getSpecies() is a pure virtual function. It is not defined in the Creature class, but it must be defined in every class that is derived from the Creature class, or the Creature class won't work, because the Creature class's getDamage() function depends on the fact that getSpecies() exists in the derived class.

3. getDamage() is a virtual member function that is inherited and is sometimes redefined.

Submit Your Work

Name your client program a7.cpp. Name each class file according to the class it contains (for example, elf.h, creature.cpp). There will be 13 files. Execute your client program and copy/paste the output into the bottom of your client file, making it into a comment. Zip the files and use the Assignment Submission link to submit the zip file. When you submit your assignment there will be a text field in which you can add a note to me

(called a "comment", but don't confuse it with a C++ comment). In this "comments" section of the submission page let me know whether the program works as required.