

NET2006: Object-Oriented Programming in C++

Lab 11: More Derived Classes

Introduction

This lab is intended to be easy. But you can undertake this lab only after you complete Lab 10. With a working Lab 10, it's possible to implement the expanded capabilities of this lab in a matter of minutes – not hours or days.

Using inheritance and virtual functions, you can achieve the expanded capabilities with almost no change to the existing code. You merely add the new variations.

You will start with your solution to Lab 10; derive two new classes from the **Actor** class; and implement the associated virtual functions for those derived classes.

The two new classes can be of your choosing: **Orc**, **Elf**, **Dwarf** or some other type of **Actor**.

The Details: Adding the Classes

Step 1: Ensure Lab 10 Code is Stable

The lab 10 code implemented two derived classes: **Hobbit** and **Wizard**, and the associated virtual functions.

Step 2: Decide on the New Types

Currently, the **Hobbit** and **Wizard** classes add one new data member (an **int** for the **Hobbit**'s stealth, and a **bool** for whether the **Wizard** is holding a staff).

When inventing your new types, add more than one data member to each (to practice implementing classes that result in objects of differing size).

The data members will be used in the next lab to determine the motion of each **Actor** object during the run of the simulator.

Step 3: Copy the **Hobbit** Class and Modify

The **Hobbit** class has the essential derivation pattern. Look to the existing **virtual** functions for a model for your newly derived classes. You'll need to implement the following for both classes:

- **virtual char GetSymbol()**
- **virtual bool Read(istream& rifsInput)**

- **virtual void Write(ofstream& rofsOutput)**
- **virtual void Display()**

Step 4: Modify the **CreateNewActor()** Function

Add your two new classes to the **switch-case** statement. Here, I show several new types, though you need only add two.

```
Actor* Actor::CreateNewActor(char cActorType)
{
    switch(cActorType) {
        case 'h': return new Hobbit;
        case 'w': return new Wizard;
        case 'o': return new Orc;
        case 'e': return new Elf;
        case 'd': return new Dwarf;
        case 'n': return new Nazgul;
        // other Actor types to be added later
        default: return NULL;
    }
}
```

Step 5: Modify the Data File

Add enough records to test your newly implemented classes.

Step 6: Compile and Test

If the preceding steps have been completed correctly, the program should compile and run much as before.

Submission

Include the following in your submission:

- **Memory Map:** Create a sample memory map that shows memory after adding at least two objects for each of your **Actor** derived classes. The diagram should clearly show the difference between the data members.
- **Test Plan and Test Files:** These two things work hand-in-hand. The *test plan* describes how you will test. The *test files* implement much of the testing.
- I also want a printout of your source code.