

CPP Problem Design

Subject: Creatures

Contributor: 謝宜杭, 謝公耀, 廖宣瑋

Main testing concept: Inheritance

Basics

- ☐ C++ BASICS
- ☐ FLOW OF CONTROL
- ☒ FUNCTION BASICS
- ☐ PARAMETERS AND OVERLOADING
- ☐ ARRAYS
- ☒ STRUCTURES AND CLASSES
- ☒ CONSTRUCTORS AND OTHER TOOLS
- ☐ OPERATOR OVERLOADING, FRIENDS, AND REFERENCES
- ☐ STRINGS
- ☐ POINTERS AND DYNAMIC ARRAYS

Functions

- ☐ SEPARATE COMPILATION AND NAMESPACES
- ☐ STREAMS AND FILE I/O
- ☐ RECURSION
- ☒ INHERITANCE
- ☒ POLYMORPHISM AND VIRTUAL FUNCTIONS
- ☐ TEMPLATES
- ☐ LINKED DATA STRUCTURES
- ☐ EXCEPTION HANDLING
- ☐ STANDARD TEMPLATE LIBRARY
- ☐ PATTERNS AND UML

Description:

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 we can inflict
    int hitpoints; // How much damage we can sustain
    string getSpecies(); // Returns type of species
public:
    Creature( );
    // Initialize to human, 10 strength, 10 hit points
    Creature(int newType, int newStrength, int newHit);
    // Initialize creature to new type, strength, hit points
    // Also add appropriate accessor and mutator functions
    // for type, strength, and hit points
    int getDamage();
    // Returns amount of damage this creature
    // inflicts in one round of combat
};
```

Input:

No Input. Given testing main().

Output:

See Sample.

Sample Input / Output :

| | |
|--------------|---------------|
| Sample Input | Sample Output |
| Main. in | Output1. txt |

- ☒ Easy, Only basic programming syntax and structure are required.
- ☐ Medium, Multiple programming grammars and structures are required.
- ☐ Hard, Need to use multiple program structures or more complex data types.

Expected solving time:

30 minutes

Other notes:

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 $0 < r \leq \text{strength}$.
- 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 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 so fast 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 unwieldy to add new creatures. Rewrite the class to use inheritance, which will eliminate the need for the variable type. The Creature class should be the base class. The classes Demon, Elf, and Human should be derived from Creature. The classes Cyberdemon and Balrog should 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. This will compute the basic damage that all creatures inflict, followed by the random 5% damage that demons inflict, followed by the double damage that balrogs inflict.

Also include mutator and accessor functions for the private variables. Write a main function as shown in the following that contains a driver to test your classes. It should create an object for each type of creature and repeatedly outputs the results of `getDamage()`.

// Note: the following main() is used to test your program
// You may have to run this program many times to encounter the special attacks.

```

int main()
{
srand(static_cast<int>(time(NULL)));
Human h(30,10);
h.getDamage();
cout << endl;
Elf e;
e.getDamage();
cout << endl;
Balrog b(50,50);
b.getDamage();
cout << endl;
}

```

```
Cyberdemon c(30,40);  
c.getDamage();  
cout << endl;  
}
```