CMSC 202 — Fall 2015 — Prof. Marron

Project 4: Zombie Apocalypse

Due: Monday, November 23rd, 9:00PM

Objective
The objectives of this project are 1) to practice designing C++ derived classes, and 2) to practice using polymorphism in C++.

Background
The program for this project is a game/simulation that involves a bunch of "characters" in a dungeon/maze. The dungeon consists of rooms. Each room is connected to some of the other rooms, as specified by a map. Characters that are in the same room interact with each other. Characters also wander from room to room.

During each *turn* of the simulation, the following will happen, in this order:

1. Dead characters are removed.
2. Each character interacts with one randomly selected character who is in the same room. A character will not be asked to interact with himself/herself. If there is only 1 character in a room, then no interaction take place.
3. Each character migrates to a neighboring room — one that is connected to its current room, as specified by the map. Some characters do not migrate and stay in the same room.
4. Each character might transform into another character at the end of a turn. For example, a human who has been bitten by a zombie, might transform into a zombie.

These simulations are handled by the member functions in a `Game` class that is provided for you. This `Game` class only works with a base class called `Character`. To make the simulation interesting, you will derive classes from `Character` that must conform to some behaviors specified below. The character types that you will derive are: Zombie, Human, Doctor, Buffy, Vampire and RedShirt.

Here is the declaration for `Character` (comments removed):

```
class Character {


public:

   Character() ;

   virtual ~Character() ;

   virtual void encounter(Character *ptr) = 0 ;
```

```
    virtual void biteMe(Character *ptr) = 0 ;

    virtual void hitMe() = 0 ;

    virtual void cureMe() = 0 ;

    virtual Character *morph() = 0 ;

    virtual bool migrates() ;

    virtual string identify() = 0 ;

    unsigned int getId() ;

    bool isDead() ;


protected:

    bool m_isDead ;


private:

    unsigned int m_id ;

    static unsigned int m_counter ;
} ;
```

The `Character` class has many pure virtual functions. This allows code in the `Game` class to call member functions defined by the derived classes that you will write.

Here's the intended use for each function:

- `Character()`
  The base class constructor takes care of storing a unique number in `m_id`.
- `~Character()`
  The destructor is virtual and does not do anything.
- `void encounter(Character *ptr)`
  The `Game` class calls `A.encounter(B)` to make `A` do something with `B`. For example, if `A` is a zombie, it should bite `B` when `A.encounter(B)` is called.
- `void biteMe(Character *ptr)`
  This function specifies the behavior of the Character that is bitten (not the one that does the biting). If zombie A bites a human B, then somewhere in the code for `Zombie::encounter()`, there should be a call to `B.biteMe()`.
- `void hitMe()`
  Similar to `biteMe`, this member function should determine the Character's behavior after it is "hit".
- `cureMe()`
  This function determines what happens when the host character is cured. Note: dead zombies should not be cured.
- `Character *morph()`
  This function returns a pointer to a new Character if the host is ready to change into a different creature. **If the host does**

not change, then `morph()` should return `NULL`. Note that if morph() returns a non-NULL value, the original character will be deleted by the Game class. So, do not return `this` to indicate that the character did not change.

- `bool migrates()`
  This functions says whether the Character moves around.
- `string identify()`
  This function returns a string that describes the host (e.g., `"Human #6"`, `"Vampire #7"`,...)
- `unsigned int getId()`
  This function returns `m_id`, the unique ID number of the host.
- `bool isDead()`
  This function returns `m_isDead`. Note: vampires and zombies are not dead — they are undead.

Assignment

Your assignment is to write/design classes derived from the `Character` class. Your derived objects must conform to the behavior described below.

These are the classes you have to implement:

- Zombie
  Zombies bite, they do not hit. If they are hit 3 times, then they die. Zombies migrate. Zombies do not cure. If a zombie is bitten, nothing happens. (Zombies do not become vampires.) If a zombie is cured, it turns into a human.
- Human
  Humans do not bite, they do not hit. If they are bitten by a zombie or a vampire, then they turn into the corresponding creature after 3 turns. Humans do not cure. Only good guys hit, so if a Human is hit, nothing happens. (We'll assume that Buffy would not really punch a Human.) If a Human is cured after they were bitten, then the countdown clock for turning into a Vampire or a Zombie is reset to zero (and the countdown stops). A cured Human is still vulnerable to bites in the future. I.e., curing does not convey any immunity. Humans do migrate.
- Doctor
  Doctors are like humans, except they cure. Doctors who are bitten cannot cure themselves, but they can be cured by other Doctors. If a Doctor becomes a Zombie and is subsequently cured, that Doctor/Zombie turns into a regular Human, not a Doctor.
- Buffy
  Buffy hits. She does not bite. She is too quick for Zombies and Vampires, so she cannot be bitten. She also cannot be hit. Curing has no effect on Buffy since she doesn't get bitten. Buffy does not cure others. She does migrate.
- Vampire
  Vampires have the same attributes as Zombies except that they cannot be cured. They do die after being hit three times (same as Zombies).
- RedShirt
  (Think Star Trek.) A RedShirt does not migrate. RedShirts stay in the same room and hit characters that they encounter. RedShirts die immediately if they are bitten, thus they cannot be cured (and they die quickly).

Implementation Issues

1. The files provided to you have been placed here:

   `/afs/umbc.edu/users/c/h/chang/pub/cs202stuff/proj4/`
   Except for `main.cpp`, you should not change the content of these files.

   `character.cpp`

```
character.h
game.cpp
game.h
ghost.cpp
ghost.h
linked.cpp
linked.h
main.cpp
node.cpp
node.h
```

2. Here is an example of the output from the program: proj4-output.txt. You do not have to make your output look exactly like this.
3. Note that the Character class is an abstract class because it has pure virtual member functions. (That what those `=0`'s at the end of the member function declarations mean.) So, you cannot create a Character object. You can only create objects of derived classes that have every pure virtual member function defined.
4. Your implementation should not rely on runtime type checking. In particular, your implementation should allow for future class derivations. For example, we should be able to introduce a new Werewolf class that bites. Then, without changing your code, your Humans should turn into Werewolves after 3 turns if they are bitten by a Werewolf.
5. How does a Human know to morph into a Zombie or a Vampire? When a Zombie invokes the `biteMe()` function, it has to pass a new Zombie object to `biteMe()`. The `biteMe()` function must store a pointer to this Zombie object. After 3 turns, when the Human morphs, this pointer must be returned to the Game class which will replace the Human with a Zombie. Similarly, when a Vampire bites a Human, it should pass in a new Vampire object. Thus, a Human bitten by a Vampire becomes a Vampire and a Human bitten by a Zombie becomes a Zombie.
6. Watch out when a Human who was bitten gets cured. You have to delete the pointer to the Vampire or Zombie that you stored somewhere. If you do not, then you will have a memory leak. Use `valgrind` on GL to check your program for memory leaks. The syntax for invoking `valgrind` is:

   ```
   valgrind ./a.out
   ```

   assuming that your executable file was named `a.out`.
7. You should not use any global variables.
8. Note that the phases of each turn take place in lock step. For example, all the characters interact before any of them morph. This means it is possible for a Human who was just cured by a Doctor to be bitten again in the same turn. Also, a Zombie who was just cured by a Doctor, might suffer some hits from Buffy (and die) during the same turn because the transformation to human takes place in the last phase.
9. The `Ghost` class implementation in `ghost.h` and `ghost.cpp` should help you get started with the C++ syntax for class derivations.
10. If you copy `ghost.h` to make a header file for another class, make sure you change the guards at the top of the file.

Submitting your program

Extra Credit: If you submit your project by the original due date of Thursday, November 19, 9:00PM, you will receive 5 points of extra credit on this project. To submit early, copy your files to `proj4-early` instead of `proj4`.

You should submit these files to the `proj4` subdirectory: `zombie.h`, `zombie.cpp`, `human.h`, `human.cpp`, `buffy.h`, `buffy.cpp`, `doctor.h`, `doctor.cpp`, `vampire.h`, `vampire.cpp`, `redshirt.h`, `redshirt.cpp`.

You should also submit a main program that works for you. This allows you to demonstrate what you have accomplished if you are not able to implement all of the Character classes. Call this file `proj4.cpp`.