Algorithmic Adventures: Into the Recursive Realms

Project 4 - The Tainted Stew

Now that we have our full-fledged Characters and a game world, let's implement some game functionality. In this project you will modify the classes you have implemented thus far to allow Characters to eat **`Tainted Stew`**. Eating tainted stew will affect each `Character` subclass differently. Have your `Barbarians`, `Mages`, `Rangers` and `Scoundrels` feast in the `Tavern` by eating Tainted Stew and see what happens to them!!!

The link to accept the GitHub Classroom assignment can be found on Blackboard

This project consists of two parts:

- ***You will modify the `Character` class and its subclasses*** to enable them to eat Tainted Stew and display their current status.
- ***You will modify the `Tavern`*** to store `Character` subclasses polymorphically and access each subclass-specific tainted stew actions.

Some additional resources

- ***Abstract Classes and Polymorphism:***
  - Pure Virtual and Abstract Classes
  - Early vs Late binding
- ***Reading from input file:***
  - ifstream
  - stringstream
  - getline
- ***String manipulation:***
  - C++ Substring of a given string
  - C++ find position of a string inside another string
- ***Dynamic memory allocation:***
  - `new` and `delete` key words

Implementation

Work through the tasks sequentially (implement and test). Only move on to a task when you are positive that the previous one has been completed correctly. Remember that the names of classes and methods must exactly match those in this specification **(FUNCTION NAMES, PARAMETER TYPES, RETURNS, PRE AND POST CONDITIONS MUST MATCH EXACTLY)**.

Remember, you must thoroughly document your code!!!

Task 1: Modify the `Character` class and its Subclasses

Modify the Character class as follows:

- Make the `display()` function pure virtual
- Add the following public member function and also make it pure virtual

```
/**
    @post: Modifies the character's private member variables (the exact modifications will be subclass specific)
*/
  eatTaintedStew
```

Modify the Barbarian class as follows:

- Add the following public member functions

```
/**
    @post     : displays Barbarian data in the form:
    "[NAME] is a Level [LEVEL] [RACE] BARBARIAN.
    \nVitality: [VITALITY]
    \nArmor: [ARMOR]
    \nThey are [an enemy/not an enemy].
```

```
    \nMain Weapon: [MAINWEAPON]
    \nOffhand Weapon: [OFFHANDWEAPON]
    \nEnraged: [TRUE/FALSE]
    \n\n"

    Example:
    BONK is a Level 5 HUMAN BARBARIAN.
    Vitality: 11
    Armor: 5
    They are an enemy.
    Main Weapon: MACE
    Offhand Weapon: ANOTHERMACE
    Enraged: TRUE
*/
    display



/**
    @post:
    If the character is UNDEAD, gain 3 Vitality points. Nothing else happens.

    If the character is NOT UNDEAD, Vitality is set to 1.
    In addition, as a Barbarian:
    Become enraged if the character was not enraged, and not enraged if they were already enraged.
    If they have now become enraged, the offhand weapon is replaced with "TABLE".
    If they are now not enraged, the main weapon is replaced with "BUCKET".
*/
    eatTaintedStew
```

Modify the Mage class as follows:

- Add the following public member functions

```
/**
    @post     : displays Mage data in the form:
    "[NAME] is a Level [LEVEL] [RACE] MAGE.
    \nVitality: [VITALITY]
    \nArmor: [ARMOR]
    \nThey are [an enemy/not an enemy].
    \nSchool of Magic: [SCHOOL]
    \nWeapon: [WEAPON]
    \nThey [can/cannot] summon an Incarnate.
    \n\n"

    Example:
    SPYNACH is a Level 4 ELF MAGE.
    Vitality: 6
    Armor: 4
    They are not an enemy.
    School of Magic: ILLUSION
    Weapon: WAND
    They can summon an Incarnate.
*/
    display
```

```
/**
    @post:
    If the character is UNDEAD, gain 3 Vitality points. Nothing else happens.

    If the character is NOT UNDEAD, Vitality is set to 1.
    In addition, as a Mage:
    If the character is equipped with a wand or staff, they cast a healing ritual and recover vitality points – 2 points
with a wand, 3 with a staff.
    If they can summon an incarnate, the emotional support allows the character to recover 1 Vitality point.
*/
  eatTaintedStew
```

Modify the Ranger class as follows:

- Add the following public member functions

```
/**
    @post     : displays Ranger data in the form:
    "[NAME] is a Level [LEVEL] [RACE] RANGER.
    \nVitality: [VITALITY]
    \nArmor: [ARMOR]
    \nThey are [an enemy/not an enemy].
    \nAnimal Companion: [TRUE/FALSE]
    \nArrows:
    \n[ARROW TYPE]: [QUANTITY]
    \nAffinities: [AFFINITY1], [AFFINITY2],..., [AFFINITYN]
    \n\n"

    Example:
    MARROW is a Level 6 UNDEAD RANGER.
    Vitality: 9
    Armor: 4
    They are not an enemy.
    Animal Companion: TRUE
    Arrows:
    WOOD: 30
    FIRE: 5
    Affinities: FIRE, POISON
*/
    display



/**
    @post:
    If the character is UNDEAD, gain 3 Vitality points. Nothing else happens.

    If the character is NOT UNDEAD and, as a Ranger has POISON affinity, their Vitality is halved (round down to the
nearest integer).
    Otherwise (not UNDEAD and not POISON affinity), their Vitality is set to 1.
    In either case, if they have an animal companion, the emotional support allows the character to recover 1 Vitality
point.
*/
```

```
  eatTaintedStew
```

Modify the Scoundrel class as follows:

- Add the following public member functions

```
/**
    @post     : displays Scoundrel data in the form:
    "[NAME] is a Level [LEVEL] [RACE] SCOUNDREL.
    \nVitality: [VITALITY]
    \nArmor: [ARMOR]
    \nThey are [an enemy/not an enemy].
    \nDagger: [DAGGER]
    \nFaction: [FACTION]
    \nDisguise: [TRUE/FALSE]
    \n\n"

    Example:
    FLEA is a Level 4 DWARF SCOUNDREL.
    Vitality: 6
    Armor: 4
    They are an enemy.
    Dagger: ADAMANT
    Faction: CUTPURSE
    Disguise: TRUE
*/
    display



/**
    @post:
    If the character is UNDEAD, gain 3 Vitality points. Nothing else happens.

    If the character is NOT UNDEAD, their Vitality is set to 1.
    In addition, as a Scoundrel: If the character is of the CUTPURSE faction, they steal a health potion and recover 3
Vitality points.
    If they are of the SILVERTONGUE faction, they talk the cook into redoing their stew as follows: they have a 70%
chance of recovering 4 Vitality points, but a 30% chance of resetting their Vitality to 1, and they lose their daggers,
which are replaced with WOOD daggers. (If their daggers were already WOOD, nothing happens to the daggers).
*/
  eatTaintedStew
```

Task 2: Modify the `Tavern` class

- Modify the `Tavern` to now store `pointers to Characters`, rather than Character objects

- Implement a parameterized constructor

```
/**
    @param: the name of an input file
    @pre: Formatting of the csv file is as follows (each numbered item appears separated by comma, only one value for each numbered item):
1. Name: An uppercase string
2. Race: An uppercase string [HUMAN, ELF, DWARF, LIZARD, UNDEAD]
3. Subclass: An uppercase string [BARBARIAN, MAGE, SCOUNDREL, RANGER]
4. Level/Vitality/Armor: A positive integer
5. Enemy: 0 (False) or 1 (True)
6. Main: Uppercase string or strings representing the main weapon (Barbarian and Mage), Dagger type (Scoundrel), or arrows (Ranger). A ranger's arrows are of the form [TYPE] [QUANTITY];[TYPE] [QUANTITY], where each arrow type is separated by a semicolon, and the type and its quantity are separated with a space.
```

```
7. Offhand: An uppercase string that is only applicable to Barbarians, and may be NONE if the Barbarian does not have an offhand weapon, or if the character is of a different subclass.
8. School/Faction: Uppercase strings that represent a Mage's school of magic: [ELEMENTAL, NECROMANCY, ILLUSION] or a Scoundrel's faction: [CUTPURSE, SHADOWBLADE, SILVERTONGUE], and NONE where not applicable
9. Summoning: 0 (False) or 1 (True), only applicable to Mages (summoning an Incarnate) and Rangers (Having an Animal Companion)
10. Affinity: Only applicable to Rangers. Affinities are of the form [AFFINITY1];[AFFINITY2] where multiple affinities are separated by a semicolon. Th value may be NONE for a Ranger with no affinities, or characters of other subclasses.
11. Disguise: 0 (False) or 1 (True), only applicable to Scoundrels, representing if they have a disguise.
12. Enraged: 0 (False) or 1 (True), only applicable to Barbarians, representing if they are enraged.
    @post: Each line of the input file corresponds to a Character subclass and dynamically allocates Character derived objects, adding them to the Tavern.
*/
```

- Add the following public member functions

```
/**
    @post: For every character in the tavern, displays each character's information
*/
    displayCharacters



/**
    @param: a string reference to a race
    @post: For every character in the tavern of the given race (only exact matches to the input string), displays each character's information

*/
    displayRace




/**
    @post: Every character in the tavern eats a tainted stew.
*/
    taintedStew
```

Testing

How to compile with your **Makefile**:
In terminal, in the same directory as your *Makefile and your source files*, use the following command

```
make rebuild
```

This assumes you did not rename the `Makefile` and that it is the only one in the current directory.

You must always implement and test your programs **INCREMENTALLY!!!**
What does this mean? Implement and **TEST one method at a time**.
For each class:

- Implement one function/method and **test it thoroughly** (write a main file with multiple test cases + edge cases if applicable).
- Only when you are certain that function works correctly and matches the specification, move on to the next.
- Implement the next function/method and test in the same fashion.
  How do you do this? Write your own `main()` function to test your classes. Choose the order in which you implement your methods so that you can test incrementally: i.e. implement constructors then accessor functions, then mutator functions. Thoroughly test with valid and invalid input to check thet your function behaves as expected in each case. Pay special attention to edge cases. Sometimes functions depend on one another. If you need to use a function you have not yet implemented, you can **use stubs**: a dummy implementation that always returns a single value for testing. Don't forget to go back and implement the stub!!! If you put the word STUB in a comment, some editors will make it more visible.

Grading Rubric

Correctness 80% (distributed across unit testing of your submission)
Documentation 15%
Style and Design 5% (proper naming, modularity, and organization)

Important: You must start working on the projects as soon as they are assigned to detect any problems with submitting your code and to address them with us **well before** the deadline so that we have time to get back to you **before** the deadline.

There will be no negotiation about project grades after the submission deadline.

Submission:

We will grade the following :

```
Character.hpp
Character.cpp
Barbarian.hpp
Barbarian.cpp
Mage.hpp
Mage.cpp
Ranger.hpp
Ranger.cpp
Scoundrel.hpp
Scoundrel.cpp
Tavern.hpp
Tavern.cpp
```

Although Gradescope allows multiple submissions, it is not a platform for testing and/or debugging and it should not be used for that. You MUST test and debug your program locally. To help you not rely too much on Gradescope for testing, we will only allow **5 submissions per day**. Before submitting to Gradescope you MUST ensure that your program compiles using the provided `Makefile` and runs correctly on the Linux machines in the labs at Hunter (see detailed instructions on how to upload, compile and run your files in the "Programming Guidelines" document). That is your baseline, if it runs correctly there it will run correctly on Gradescope, and if it does not, you will have the necessary feedback (compiler error messages, debugger or program output) to guide you in debugging, which you don't have through Gradescope. **"But it ran on my machine!" is not a valid argument for a submission that does not compile.** Once you have done all the above you submit it to Gradescope.

Due date:

This project is **due on October 27 at 11pm.**
No late submissions will be accepted.

Important

You must **start working on the projects as soon as they are assigned** to detect any problems and to address them with us **well before** the deadline so that we have time to get back to you **before** the deadline.
There will be no extensions and no negotiation about project grades after the submission deadline.

Help

Help is available via drop-in tutoring in Lab 1001B (see website for schedule). You will be able to get help if you start early and go to the lab early. We only have 2 UTAs in the lab, **the days leading up to the due date will be crowded and you will not be able to get much help then**.

Authors: Georgina Woo, Tiziana Ligorio