Program name: Project3 Fantasy Combat Game
Author: Xiaoying Li
Date: 8/3/2019
Description: Design Description & Class Hierarchy Diagram & Test Table & Reflection of Project3 Fantasy Combat Game

- **Design**

1. Main

    The main function controls the entire program.
    Ask user whether he/she wants to start the game or not.
        If the user decided not to start the game, quit the program.
        Otherwise, start the game.
            Display five characters by their names.
            Prompt user to select two characters to fight one another.
            Create two characters correspondingly.
            Implement the combat between them.
    After the combat ends, ask the user whether he/she wants to play again.
        If the user decided to play again, start the game again.
        Otherwise, quit the program.
    Loop the game until the user decided to quit.

2. Menu

The menu function prints menu options to the screen for the user, and after the user makes a choice, verify the user's input, and return the value back to the program.

(1) startMenu:
    When the project starts, ask user whether he/she wants to start the game or not.
    If the user decided to start the game, return true.
    Otherwise, return false.

(2) endMenu:
    When the game ends, ask the user whether he/she wants to play the game again.
    If the user decided to play the game again, return true.
    Otherwise, return false.

3. Character

The Character class is an abstract base class, will be inherited by all the characters' own subclass.

    It has following protected data members:
        Characters' type<string>, attack point, defense point, armor, strength<int>.
    It has following public member functions:
        Set function for characters' type;
        Function implements characters' attack;
        Function implements characters' defense;

Function determines whether a character is dead or not.

The first function is virtual function. The other three functions are pure virtual functions.

4. Barbarian

The Barbarian class inherits from Character class, and contains all information of Barbarian.

Its private data members are inherited from Character class.

It has following public member functions:

Default constructor:

initializes all data members when a Barbarian object is created.

Override attack() function that takes no parameter, calculates and returns Barbarian's attack point.

Override defense() function that takes in the attacker's attack point, calculates and prints Barbarian's defense point and strength.

Override die() function that takes no parameter, determins whether Barbarian is dead or not. If Barbarian is dead, return true. Otherwise, return false.

5. BlueMen

The BlueMen class inherits from Character class, and contains all information of Blue Men.

Its private data members are inherited from Character class.

It has following public member functions:

Default constructor:

initializes all data members when a BlueMen object is created.

Override attack() function that takes no parameter, calculates and returns Blue Men's attack point.

Override defense() function that takes in the attacker's attack point, calculates and prints Blue Men's defense point and strength.

Implement Blue Men's special ability.

Determin whether Blue Men lose defense dice.

If Blue Men lost 4-7 points of strength, they lost one defense dice.

If Blue Men lost 8-11 points of strength, they lost two defense dices.

Override die() function that takes no parameter, determins whether Blue Men are dead or not. If Blue Men are dead, return true. Otherwise, return false.

6. HarryPotter

The HarryPotter class inherits from Character class, and contains all information of Harry Potter.

Its private data members are inherited from Character class, and an integer counting Harry Potter’s death time.

It has following public member functions:

Default constructor:

initializes all data members when a HarryPotter object is created.

Override attack() function that takes no parameter, calculates and returns Harry Potter 's attack point.

Override defense() function that takes in the attacker's attack point, calculates and prints Harry Potter 's defense point and strength.

Override die() function that takes no parameter, determins whether Harry Potter is dead or not. If Harry Potter is dead, return true. Otherwise, return false.

Implement Harry Potter's special ability.

If it's Harry Potter's first death, he recovers and his total strength becomes 20.

If it's Harry Potter's second death, then he is dead.

7. Medusa

The Medusa class inherits from Character class, and contains all information of Medusa.

Its private data members are inherited from Character class.

It has following public member functions:

Default constructor:

initializes all data members when a Medusa object is created.

Override attack() function that takes no parameter, calculates and returns Medusa's attack point.

Implement Medusa's special ability.

If Medusa's attack dices roll 12, set her attack point to 50, which is big enough to let all other characters' strength less than 0, which means they are dead instantly.

Override defense() function that takes in the attacker's attack point, calculates and prints Medusa's defense point and strength.

Override die() function that takes no parameter, determins whether Medusa is dead or not. If Medusa is dead, return true. Otherwise, return false.

8. Vampire

The Vampire class inherits from Character class, and contains all information of Vampire.

Its private data members are inherited from Character class.

It has following public member functions:

Default constructor:

initializes all data members when a Vampire object is created.

Override attack() function that takes no parameter, calculates and returns Vampire's attack point.

Override defense() function that takes in the attacker's attack point, calculates and prints Vampire's defense point and strength.

Implement Vampire's special ability.

```
        Declare an integer charm that is randomly 0 or 1.
        When charm==1, Vampire's charm ability is activated, so there is a 50%
        chance that the damage is 0.
    Override die() function that takes no parameter, determins whether Vampire
    is dead or not. If Vampire is dead, return true. Otherwise, return false.
```

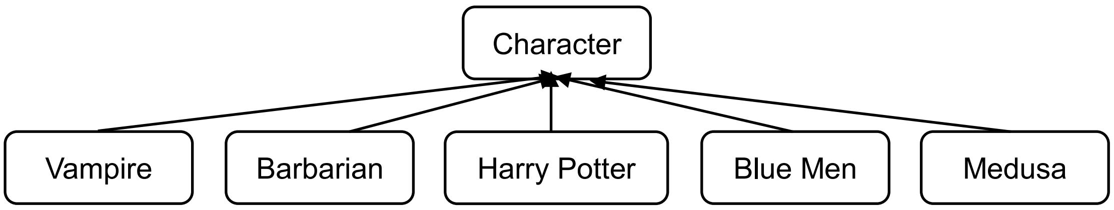
9. Validation

The validation function takes in a string and pass it by reference. The function makes sure the user's input is an integer. Afterwards, it returns the integer the user inputted.

```
Loop until the user inputs an integer.
    When a string inputted, detect whether an extraction has failed and fix it.
        If an extraction has failed, ask the user to enter again.
    If extraction succeeds, then make sure no extraneous input.
        Then detect if every character of the string is digit.
            If every character of the string is digit, then the string is an integer.
            Otherwise, ask the user to enter again.
    If the input string is an integer, transfer it to an integer, and return the integer.
```

The validation of the range of every integer returned should be validated after the integer returns to the program.

- **Class Hierarchy Diagram**



- **Test Plan**

This test plan does not include test cases for input validation, startMunu() and endMenu(), since they have been tested at Project1. This test plan only verifies if the program meets all given specifications and works well at all cases.

Test Scope	Description	Expected Results
main.combat()	Select to start Fantasy Combat Game.	Display five characters by their names, and prompt the user to select two characters to fight one another. Account for two characters of the same type.
main.combat()	Pick up 2 characters to start combat.	Combat will successfully proceed between 2 characters selected by user.

class Vampire	Select 2 Vampires to start combat.	Combat between 2 Vampires successfully proceeds. Results of each round will be displayed correctly, including all the required contents, and all the data should be correct. When the strength of one of the Vampires reaches 0, output this Vampire is dead and game over. Each Vampire should have about 50% chance to activate special ability Charm when defensing, and the damage to this Vampire should be 0 in that round.
class Barbarian	Select 2 Barbarian to start combat.	Combat between 2 Barbarians successfully proceeds. Results of each round will be displayed correctly, including all the required contents, and all the data should be correct. When the strength of one of the Barbarians reaches 0, output this Barbarians is dead and game over.
class BlueMen	Select 2 BlueMen to start combat.	Combat between 2 BlueMen successfully proceeds. Results of each round will be displayed correctly, including all the required contents, and all the data should be correct. When the strength of one of the BlueMen reaches 0, output this BlueMan is dead and game over. When the strength of a BlueMan <=8 and >4, he loses 1 defense dice; When the strength of a BlueMan <=4 and >0, he loses 2 defense dices.
class Medusa	Select 2 Medusas to start combat. Keep testing until one of the Medusas' attack point is 12.	Combat between 2 Medusas successfully proceeds. Results of each round will be displayed correctly, including all the required contents, and all the data should

		be correct. When the strength of one of the Medusas reaches 0, output this Medusa is dead and game over. When one of the Medusas' attack point is 12, her special ability Glare is activated, the other Medusa is dead and game over.
class HarryPotter	Select 2 HarryPotter to start combat.	Combat between 2 HarryPotter successfully proceeds. Results of each round will be displayed correctly, including all the required contents, and all the data should be correct. When HarryPotter's strength <=0, his special ability Hogwords will be activated, this HarryPotter will immediately recovers and his total strength becomes 20. But if his strength becomes <=0 again, he's dead and game over.
Tests listed above ensure every character can function properly. Tests listed below ensure program runs properly when the special ability of two different characters activate together.		
Medusa's Glare vs Harry Potter's Hogwards	Select Medusa and HarryPotter to combat. Situations that Medusa and HarryPotter starts first will both be tested.	If Medusa uses "glare" on Harry Potter on his first life, then Harry Potter comes back to life after using "hogwarts"
Medusa's Glare vs Vampire's Charm	Select Medusa and Vampire to combat. Situations that Medusa and Vampire starts first will both be tested.	If the Vampire's "charm" ability activates when Medusa uses "glare", the Vampire's charm trumps Medusa's glare.
Test all the possible combinations not mentioned above. Results of each round should be displayed correctly, including all the required contents, and all the data should be correct. When the strength of one of the characters reaches 0, output this character is dead and game over. Each character's special ability should be activated correctly.		

- **Reflection**

1. Design changes I have made:

In the design of this project, the only change I made was that the original return type of attack() is void, and I had getter functions for every data member in class Character. I set a getter function for every data member because I wanted to output required contents for each round in menu, and get attacker’s attack point to calculate defender’s damage. But when I coding, I noticed the data for each round can be output directly in attack() and defense(), using getter is redundant. Moreover, I changed the return type of attack() to int, then defender’s defense() could use attack point data by returning attacker, which made my code more concise

2. How I solved problems encountered:

The first-time test after I finished coding showed error messages that every character’s sub class didn’t include their base class Character’s pure virtual function. Apparently, my understanding of pure virtual function was off. I learned knowledges related to pure virtual function online, and I found pure virtual function needs to include their declaration in the header file of sub class, which is different from common inheritance. My code ran successfully after I fixed this issue.

3. What I learned for this project:

This project deepened my understanding of polymorphism, helped me practice the application of inheritance between classes, and gave me an opportunity to learn knowledges about abstract base class. This project strengthened my ability of convert project requirement into code as well as coding more complex program. During test phase, my skills of locating errors through error message in compiler has also been sharpened.