Project 3 Design + Reflection
Tay, Serena
Student #: 933588509

Document (Design + Reflection)

You need to create your program design **BEFORE** you start coding. Your design needs to be included in your document.

Simply stated, **program design** is identifying the problem to be solved, modularizing and describing the structure of program, and making a test table to that fulfills the requirements of the program.

For test table, please include test plan, expected output, and actual output. The test plan should be robust. The expected output is based on the requirement. If your actual output deviates from your expected output, that usually means the program behavior is deviating from the requirements. Following is a Examples Test Plan.pdf

In your **reflection**, you should explain what design changes you made, how you solved some problems you encountered, and what you learned for this project. The TA will grade on how well your implementation matches your design.

Implemented the extra credit option #2:
Status messages: (5pts)

Design

- Create a master parent class with the following member variables and functions to be inherited to all the character classes.
- Five classes which are Barbarian, BlueMen, HarryPotter, Medusa and Vampire are derived from the character class.

Character Class
Contain the following member variables to store character data as outlined in the project requirements:

```
protected:
  int characterType,
      attack,
      defense,
      armor,
      strengthPoints,
      attackDieSides,
      defenseDieSides;
      string characterName;
```

Project 3 Design + Reflection
Tay, Serena
Student #: 933588509

Contain the following class functions:

```
public:
  Character();
  int getCharacterType ();
  string getCharacterName();
  virtual int getAttack();
  virtual int getDefense();
  virtual int getArmor();
  virtual int getStrengthPoints();
  virtual void setStrengthPoints(int);
};
```

The virtual functions are to be overridden as necessary in the derived class.

Data validation/integrity consideration for Animal class:

- Nothing as this is the parent class that does not get any user inputs

Barbarian Class

```
  public:
    Barbarian();
    int getAttack();
    int getDefense();
    int getArmor();
    int getStrengthPoints();
```

BlueMen Class

```
  public:
    BlueMen();
    int getAttack();
    int getDefense();
    int getArmor();
    int getStrengthPoints();
    void mob();
```

HarryPotter Class

```
private:
  int lifeCount; //for hogwarts
public:
  HarryPotter();
  int getAttack();
  int getDefense();
  int getArmor();
  int getStrengthPoints();
  void hogwarts();
```

Project 3 Design + Reflection
Tay, Serena
Student #: 933588509

Medusa Class

```
  public:
    Medusa();
    int getAttack();
    int getDefense();
    int getArmor();
    int getStrengthPoints();
```

Vampire Class

```
private:
  bool charmStatus;

public:
  Vampire();
  int getAttack();
  int getDefense();
  int getArmor();
  int getStrengthPoints();
  bool charm();
  bool getCharmStatus();
  void setStrengthPoints(int); //virtual function from character override
```

- Constructors in the derived character classes are used to initialize object with the data on strength Points, defence and attack die sides
- Virtual getAttack and getDefense functions are overrode to include the specification of each characters # of die sides
- getArmor and getStrengthPoints, in hindsight should be normal functions as no override is needed
- setStrengthPoints – same for all characters except for Vampire (see below)

Character Abilities Functions

BlueMen – created a mob function to keep track of strength points and whenever it hits below 8 and 4, to decrease die rolls but only decrease once in each range
HarryPotter – created a hogwart function to revive character one and give him a strengthpoints of 20
Medusa – glare ability is embedded in the attack function where if it hits 12, a big attack 50 points will be returned which is greater than any strengthpoints and defence combination in the game
Vampire – a charm function that creates a bool to determine 50% chance of deflecting attacks. setStrengthPoints function overrode to take in bool to determine if the damage will be registered

Menu Class

Project 3 Design + Reflection
Tay, Serena
Student #: 933588509

Contain the following member variables:

- countPlay – This is used to determine if the “new run” menu should be used or “play again” menu should be used per requirements
- menuChoice – used to store choice entered by user

Contain the following member functions:

- Menu () – default constructor to initialize member variables used
- displayMenu() – to display menu (contained 2 menus, one for the new game, one for the ‘play again’ scenario once the first Game run occurred
- prompt(string, int, int) – a generic function that can be used to prompt user for data inputs. Contain 3 parameters, one string to tell user which member variable the program is asking for, one integer for the minimum allowable input and one for the maximum allowable input. **Please note that this function was discarded and not used**
- int promptChoice(string, string, string, string, string) – prompt user to select character type
- getMenuChoice() – function to be used in Game class to obtain user’s input of what the user whats to do

Data validation/integrity consideration for Menu class:

- make sure user gets an error message when entering an invalid choice
- create a max and min for a range of allowable input (create an error message when user entered an invalid number outside the range)
- make sure non-ints are guarded from creating errors in the program

Game class

Contain the following member variables:

```
private:
  int round,
      playChoice,
      characterChoice1,
      characterChoice2,
      damage1,
      damage2,
      attackRoll1,
      defenderRoll1,
      defenderArmor1,
      attackRoll2,
      defenderRoll2,
      defenderArmor2;
  Character *character1;
  Character *character2;
```

Project 3 Design + Reflection
Tay, Serena
Student #: 933588509

characterChoice, damage, attack and defender and character pointers were created to get data from character classes for both players.

Contain the following member functions:

```
Game();
void runGame();
void createCharacter(int, int);
void runRound();
bool processDamage(int, int);
void printResult(int);
void deallocatePointers();
```

- Game() – default constructor used to initialize all the member variables listed above
- createCharacter – To dynamically create characters based on user choice
- runGame() – used to run the Game and calling the menu and character classes in sequential order (display menu, ask user for input, run program, and repeat until user want to quit)
- runRound – initiate the combat between both characters
- processDamage – to run calculation of attacks and defence and determine if a damage resulted from that combat
- printResults - to print result for the round for testing purposes
- deallocatePointers – to deallocate pointers to avoid memory leaks

Data validation/integrity consideration for Game class:

- Use menu functions to ensure user inputs are valid for daily purchase, first purchase and choice to move to next day or quit

Project 3 Design + Reflection
Tay, Serena
Student #: 933588509

Testing plan and results

Menu class

- displayMenu() – to display menu (contained 2 menus, one for the new game, one for the 'play again' scenario once the first Game run occurred

Test Case	Input Values	Expected Outcomes	Observed Outcome
Display 1st menu when program initialize	Menu counter	Display first menu	Success – Achieved expected outcomes
Display "play again menu" after first run	Menu counter	Display play again menu	Success – Achieved expected outcomes

- prompt(choice) – a generic function that can be used to prompt user for choice of which characters they want to use

Test Case	Input Values	Expected Outcomes	Observed Outcome
Program select correct character based on user choice	User input 1,2,3,4 or 5	Display correct character names when combat starts based on user inputs	Success – Achieved expected outcomes for all inputs prompted for
Display error messages if input does not fall between the min and max set	Game class calls	Display appropriate error message when user input is outside of valid range set and reprompt user until valid input is provided for valid character choices (1 to 5)	Success – Achieved expected error message outcomes for all invalid inputs and reprompt user for valid input

Game class

- Game() – default constructor used to initialize all the member variables listed above

Test Case	Input Values	Expected Outcomes	Observed Outcome
Ensure constructor initialize to default values	N/A – set in program	Member variables initialize to the values stated	Member variables initialized to the values stated

- runGame() – used to run the Game and calling the menu and Animal class in sequential order (display menu, ask user for input, run program, ask user what they want and repeat until user want to quit)

Test Case	Input Values	Expected Outcomes	Observed Outcome

Project 3 Design + Reflection
Tay, Serena
Student #: 933588509

Ensure function is run the function calls in sequence and keep looping until user decides to quit	N/A – set in program	1. display menu 2. ask user for input (characters) 3. run program/combat 4. ask user if they want to play again with menu display 5. Repeat loop until user want to quit	Success – Achieved expected error message outcomes for all invalid inputs and re-prompt user for valid input

- createCharacter – To dynamically create characters based on user choice

Test Case	Input Values	Expected Outcomes	Observed Outcome
Ensure createCharacter function creates the correct characters	User input	Function dynamically create characters as prompted	Success – expected outcome achieved

- runRound – initiate the combat between both characters

Test Case	Input Values	Expected Outcomes	Observed Outcome
Ensure rounds are executed in sequential order (Player 1 attack first) and runs until one player dies	Character class inputs	Rounds are executed in order. Player 1 attack, then player 2, then repeat until a player dies	Success – expected outcome achieved

- processDamage – to run calculation of attacks and defence and determine if a damage resulted from that combat

Test Case	Input Values	Expected Outcomes	Observed Outcome
Ensure rounds are executed in sequential order (Player 1 attack first) and function calculate damage accurately	Character class inputs	Rounds are executed in order. Player 1 attack, then player 2, then repeat until a player dies. Damage only applied when greater than zero and player dies when no strength points is left.	Success – expected outcome achieved

- printResults - to print result for the round for testing purposes

Test Case	Input Values	Expected Outcomes	Observed Outcome

Project 3 Design + Reflection
Tay, Serena
Student #: 933588509

Print results of each round of attack by both players depending on situation	Character class inputs	Rounds are executed in order. Player 1 attack, then player 2, then repeat until a player dies. Damage only applied when greater than zero and player dies when no strength points is left. Display any abilities used by special character for testing purposes	Success – expected outcome achieved

- deallocatePointers – to deallocate pointers to avoid memory leaks

Test Case	Input Values	Expected Outcomes	Observed Outcome
Memory leak	N/A	Run program with Valgrind to ensure no memory leak	Success – expected outcome achieved

Project 3 Design + Reflection
Tay, Serena
Student #: 933588509

Reflection

Inheritance

- virtual functions are used heavily in this program which allow characters with different attack and defence functions to be implemented with each
- setting up the member variables and functions in the character class carefully enable all derived characters to inherited the functions and reduce the time and effort significantly to re-create these member variables in each of the derived class

Design

- I initially made most of the functions virtual but soon realised some functions such as get characterName does not require to be virtual as it would be getting the name of the characters regardless of the character class
- It was hard to keep track of the results and ensuring the damage is applied accurately in the strengthPoints of the characters. I initially forgot to create a guard to set negative strengthPoints which created somewhat of an infinite loop as the characters strength points would increase if a negative damage was calculate. This was fixed with a guard to ignore damage with negative points
- Incorporating the character special abilities was difficult. Initially I wanted to create if functions to check if the characters are of a certain type but realized I can embed these in the respective character classes by creating additional special ability functions that get triggered when certain events occur
- HarryPotter hogwart spell was initially designed to be triggered when strengthPoint = 0, I realized that harryPotter can die when receive a damage that would result in a negative strengthPoint balance, and decided to change the logical statement to <= 0 instead to capture that range.
- The mob function also required a count in the class to ensure it is only deducting one die for each of the range (4 points). Initially I had the test to ==4 and ==8 but realized it could be below 8/4 and not equal 8/4 depending on the damage inflicted

Testing Phase

- Using the print results two-fold (both for the program design and test) helped ensured my program is working properly
- Testing strategy included the following as outlined in the testing table above
    - Out of bounds, invalid user inputs to prompt error message
    - Accuracy of characters chosen
    - Accuracy of damage calculation
    - Program handling events/situation properly
    - Program prompting function (mob and hogwart) when criterias are met
    - Memory leaks