Lee, Melisa
leemeli@oregonstate.edu
CS 162-400
Project 4: Design + Reflection

Program Design:

Game plan flow

container to hold losers
container for team 1
- linked list to keep track of each fighter

container for team 2
- linked list to keep track of each fighter

score for each team

Scoring Design
Each player starts off with 0 points
- winner get 1pt
- loser get 0 pts

Menu
1. play
2. exit

```
if user selects 1. play
User enter team information - store lineup in containers
ask user for number of fighters for BOTH teams
    - Enter number of fighters for team 1
    - Enter number of fighters for team 2
ask user for each fighter for team 1 (iterate until number of fighters are met)
    - Enter type of character
    - Enter name of character
ask user for each fighter for team 2 (iterate until number of fighters are met)
    - Enter type of character
    - Enter name of character
Note: order of lineup should be the same order of entrance

Start of tournament
head(fighter 1) of each time fight
    if fighter win
        restore some percentage of the damage (50%)
        put fighter at end of team line up
```

```
  else
    remove fighter from team line up
    put fighter in loser container
  display result of each combat
    Type of character name of character for each team
    winner
    i.e
      Fighters: Team 1 nameOfCharacter vs Team 2 nameOfCharacter
      Winner: Team x, nameOfCharacter won!

End of tournament
Display final score
  if team 1 score == team 2 score
    tie
  if team 1 score > team 2 score
    team 1 won
  if team 1 score < team 2 score
    team 2 won
  prompt user if the want to view loser container
    if yes
      print loser container
    if no

Display end Menu
1. play again
2. exit
```

Character class: (base class)

```
protected:(member variables)
  Int armor           //store character armor
  Int strengthPoint   //store character strength point
  String charName     // store name of each character
  Int startingSP      // store starting sp
  Bool Alive          // flag if character is alive
  Bool extraLife      // flag for harry potter if he has an extra life

public:(member functions)
 Character
    Description: set armor, strength point, startingSP and name to private member function
    Input: two integers representing armor and strength point, one string
    Output: nothing
  rollDie
    Description:  Randomly generate a number between two numbers
```

        Input: Integer representing min and max range
        Output: Random number between min and max range
    getArmor
        Description: return character’s armor
        Input: nothing
        Output: integer of character’s armor
    getSP
        Description: return character’s strength point
        Input: nothing
        Output: integer of character’s strength point
    getName
        Description: return character’’s name as a string
        Input: nothing
        Output: return character’s name as a string
    getAlive
        Description: return if char is alive
        Input: nothing
        Output: return a bool if char is alive
    randomNum
        Description: Random generate a number between two range
        Input: min, max
        Output: returns an integer of a randomly generated number
    recovery
        Description: Restore some percentage of damage for the winner of the combat.
        Input: nothing
        Output: nothing
        Note: call recovery func before going back in line. Set recovery of 50% of damage taken
Pure Virtual Functions
    Virtual attack() = 0
    virtual defense() = 0

QueueNode struct
            *Character
            *prev
            *next
// reuse code from lab 7
    QueueNode (constructor)
        Description:creates a character for the user type
        Input: character type and name
        Output: nothing
    ~QueueNode
        Description: delete character created
        Input: nothing

        Output: nothing

Queue class: reused most from lab 7
Private:
    QueueNode *head   // points to first node
Public:
    isEmpty
        Description: returns if bool if que is empty
        Input: nothing
        Output: returns bool
    addBack
        Description:create a node that place character type and name
        Input: int character type and string character name
        Output: nothing
    printQueue
        Description: print name of each character in queue
        Input: nothing
        Output: nothing
    getFront
        Description: returns character at the front of the queue
        Input: nothing
        Output: nothing
    addBackChar
        Description: place node at the end of queue
        Input: headnode
        Output: nothing
    removeNode
        Description: remove front node to be place in the back or in loser pile
        Input: nothing
        Output: nothing
    addNode
        Description: place node in the front
        Input: nothing
        Output: nothing

Barbarian class: (derived class of character)
Public function
    Barbarian
        Description: default constructor that sets barbarian armor =0 and strength point = 12
        Input: nothing
        Output: nothing
    attack
        Description: calculate Barbarian damage by rolling two die of six sides
        Input: nothing

Output: return Barbarian damage attack

defense

Description: Calculate actual damage inflicted and apply that damage to the defender's strength point.

Input: int damage attack

Output: nothing

Notes:

Calculation for actual damage inflicted

Damage = attacker's roll - defender's roll - defender's armor

Defender Roll: 2 die of six side

Apply damage to the defender's strength point

Defender S.P = Defender S.P - actual damage inflicted

Vampire class: (derived class of character)

Public function

Vampire

Description: default constructor that sets vampire armor = 1 and strength point = 18

Input: nothing

Output: nothing

attack

Description: calculate vampire damage by rolling 1 die of 12 sides

Input: nothing

Output: return vampire damage attack

defense

Description: Calculate actual damage inflicted and apply that damage to the defender's strength point.

Input: int damage attack

Output: nothing

Notes:

Calculation for actual damage inflicted

Damage = attacker's roll - defender's roll - defender's armor

Defender Roll: 1 die of six side

Charm: For a given attack there is 50% of opponent missing

Apply damage to the defender's strength point

Defender S.P = Defender S.P - actual damage inflicted

Blue Men class: (derived class of character)

Public function

BlueMen

Description: default constructor that sets BlueMen armor = 3 and strength point = 12

Input: nothing

Output: nothing

attack

Description: calculate BlueMen damage by rolling two die of ten sides

Input: nothing

Output: return BlueMen damage attack

defense

Description: Calculate actual damage inflicted and apply that damage to the defender's strength point.

Input: int damage attack

Output: nothing

Notes:

Calculation for actual damage inflicted

Damage = attacker's roll - defender's roll - defender's armor

Defender Roll: 3 die 6 side each

Mob: Defender Roll: For every 4 points of damage, they lose one defense die.

lostDef = dmg/4

numberOfDie -lostDef

Apply damage to the defender's strength point

Defender S.P = Defender S.P - actual damage inflicted

Medusa: (derived class of character)

Public function

Medusa

Description: default constructor that sets Medusa armor = 3 and strength point = 8

Input: nothing

Output: nothing

attack

Description: calculate Meusa damage by rolling two die of six sides

Input: nothing

Output: return Medusa damage attack

Note: If Medusa rolls a 12 target instantly get turned into stone and wins. If Medusa fights Harry Potter he loses his first life and he comes back alive.

defense

Description: Calculate actual damage inflicted and apply that damage to the defender's strength point.

Input: int damage attack

Output: nothing

Notes:

Calculation for actual damage inflicted

Damage = attacker's roll - defender's roll - defender's armor

Defender Roll: 1 die 6 side each

Apply damage to the defender's strength point

Defender S.P = Defender S.P - actual damage inflicted

Harry Potter: (derived class of character)

Public function

HarryPotter

Description: default constructor that sets HarryPotter armor = 0 and strength point = 10

Input: nothing

Output: nothing

attack

Description: calculate HarryPotter damage by rolling two die of six sides

Input: nothing

Output: return HarryPotter damage attack

defense

Description: Calculate actual damage inflicted and apply that damage to the defender's strength point.

Input: int damage attack

Output: nothing

Notes:

Calculation for actual damage inflicted

Damage = attacker's roll - defender's roll - defender's armor

Defender Roll: 2 die 6 side each

Hogwarts: Defender Roll: if strength point is less than 0 set strength to 20 once

Apply damage to the defender's strength point

Defender S.P = Defender S.P - actual damage inflicted

Combat class: reused most functions

Private member variables

Ptr Queue for team1

Ptr Queue for team2

Ptr Queue for losers

Int team1Score // store team 1 score

Int team2Score // store team 2 score

Int round // set round num

Bool firstTime, exitGame, playAgain // flags to manage menu/game option

Public Function

fillLineup

Description: create queue from user input

Input: nothing

Output: nothing

startTournament

Description: have head of each team fight until one team has no more

Input: nothing

Output: nothing

displayResult

Description: display scores to user

Input: nothing

Output: nothing

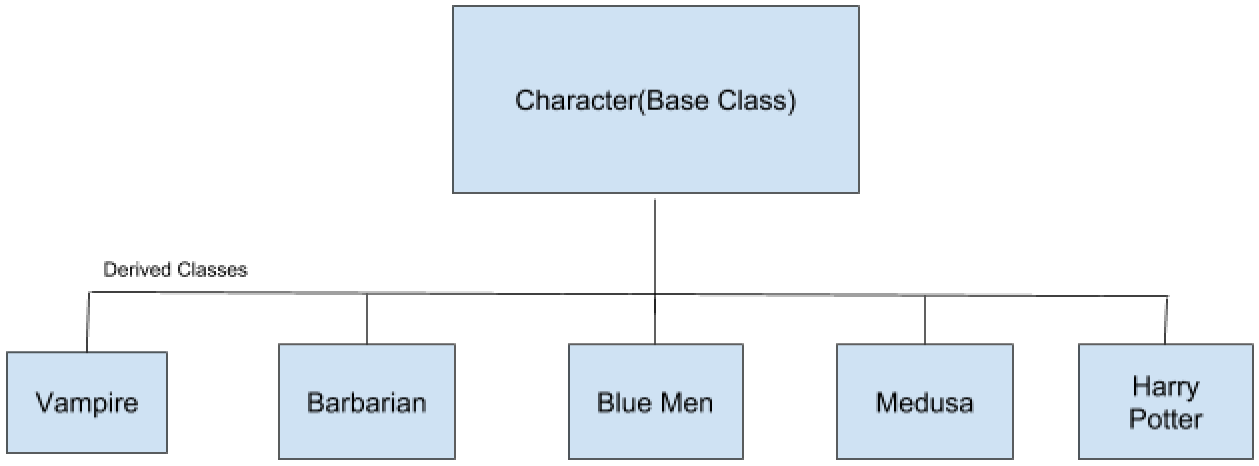
fight

Description: while character is alive, fight till death

Input: nothing

Output: nothing

Class Hierarchy Diagram



Test Case

Test Case: Lineup Order- should be the same as it was entered

Test Case	Input Values	Driver Func	Expected	Observed
Same number of characters in lineup	Team 1: barb, vamp Team 2: Harry, blue	fillLineup()	Team 1: barb, vamp Team 2: Harry, blue	As expected
Different number of characters in lineup	Team 1: barb, vamp, harry Team 2: Medusa, vamp	fillLineup()	Team 1: barb, vamp, harry Team 2: Medusa, vamp	As expected
Multiple of the same type of characters in line up	Team 1: Vamp, vamp Team 2: Vamp, vamp	fillLineup()	Team 1: Vamp, vamp Team 2: Vamp, vamp	As expected

Test Case: Lineup Order- should be the same as it was entered

Test Case	Input Values	Driver Func	Expected	Observed
Head of both team fight	-		Both fight	As expected
Winner is put in the back of the lineup	123	addBack(), remveFront()	231	As expected
Loser goes on top of the loser container	ab	addBack(), remveFront()	b	As expected

Reflection

I was able to build off project 3, I was able to reuse my code from lab 7 with slight adjustment to help with shifting the nodes around.

Some design changes I made was on my destructor. I ended up creating a function called cleanNodes that cleans up the queue. I decided to have a seperate function instead of clearing the node in the destructor specifically allowing me the to clear the queue for each Team lineup and the loser pile. Instead the destructor, simply calls the cleanNode function. A different approach I could of done is to create each line as a pointer and then deleting that pointer.

While testing for loser pile queue, I notice after adding a certain amount of characters to the loser pile and then displaying the contents; it would created an infinity loop. When I revisited my code I realized I was not linking my new head properly. I forgot to link the newHead prev and next pointer.
Code Added:

```
newHead->next = oldHead;
newHead->prev = last;
```

Going back the the rubric I realized I needed to print the name of each character during each round. To implement that functionality, I had to create a getter function instead of returning an integer; I returned a string representing the character's type. From there I was able to call my getter function to retrieve each character's type during each round.

Another small thing I realize was that I did not include in my test case was to test for white spaces when the user input a characters name. For an example, if the user inputted John Doe, I should ignore white spaces. This was an easy fix by using the cin.ignore and getline functions.

Overall, I liked how this project was able to be built off from project 3 and lab 7. I feel like I have a better understanding and practice on maintaining queues and containers.