Nicole Parker
Assignment 4

<u>Design:</u>

For this program I was thankfully able to use my assignment 3 program as a stating point and add a few things to meet the new requirements. For this design I wanted to keep as much of the new functionality as I could outside of the creature class. I thought that this would make the most sense and also keep things clean because most of the new features concerned tournament play and didn't really change the aspects of a creature. I decided to name the new class Battle, because it seemed appropriate for a tournament. After going over all of the necessary new functions I realized that although I was able to keep most of the new features outside of the creature class, I did have to add in a few things. Ultimately, it made more sense to add in a points variable to the creature class. By doing this I could make it so each creature would have their own points variable and when I wanted to determine the first, second and third place players there was an easy way to do so. I thought that points would only be achieved if a player wins a battle, and points would be correspond to how much damage a winner inflicted on their opponent. The other variables I added were name and teamName, it made sense that these would be apart of the creature class because the name of the creature directly corresponds and the team that the creature is a part so every creature object would have their specific team name. I also added a function for heal() to the creature class. This was meant to be used when a player wins a battle and some of their strength is restored. Again, being that all creatures would need a heal function, it made sense to add this to the creature class. I thought that the most fair way to restore health was to roll one six sided dice, and then the result of this roll would be added to the strength.

The main changes to the creature class were the name, teamName, points variables and the heal function.

```
1. class Creature
protected:
string name // for the creatures name entered by the user
string teamName
double points;

public:
void heal();
```

2. The battle class will contain the functions to implement the tournament play. It will contain an array of Creature queues of length two, which will hold the two teams. It will also contain a stack of Creature type, that will hold the losers of each fight. I will be able to use my fight function from assingment3, with the adjustment of changing the parameters to two Creature pointers. For making the two teams for the tournament, I thought that it would make sense to make a function that would contain all of the I/O and validation and then push the new Creature object onto the appropriate team. To actually implement a tournament I wanted to make a separate function that would handle this, it will call the fight function and then display which team has won the tournament. To show which players were the top three fighters, I also wanted to create a separate function that would handle this. I thought the best way to do this was to first combine the losers stack and teams queue (at this point the queue would only contain the winners) and then sort these in descending order by their points. Then I would be able to just take the first three players and cout those.

```
class Battle
```

```
  queue<Creature*> teams[2]
  stack<Creature*> losers
  void fight(Creature*, Creature*)

public:
  void doTournament()
  void makeTeams()
  void displayTopThree()
```

-The Dice class would not need to be changed in any way from assignment 3. The goblin, blue man, reptile, shadow and barbairan class also remained the same. With shadow keeping it's virtual applyDamage function, and Goblin keeping it's virtual attack function.

Testing:
For testing I thought it would be easier to create a function that would allow me to test the tournament, and input all of the user input, I called the function makeTestTeams(). This would be much quicker than writing everything in and I would be able to test more combinations easily. The main things I wanted to test, were that the makeTeams function did correctly make the teams. The tournament function ran appropriately and the displaytopthree function worked.

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes
Verify the makeTeams() works appropriately with valid data	Prompt: Enter number of players input: 3 players for each team input: player 1 and player 2 enter a team name input: appropriate player info for type and name.	None	- Teams are created with all input entered and in the correct order	- Teams are created with all input entered and in the correct order ** I used the makeTestTeams() to test this. I also input the info manually a few times to make sure the the prompts displayed correctly
Verify the makeTeams() validation works	Prompt: Enter number of players input: 1 player for each team input: player 1 and player 2 enter a team name input: invalid	None	- Validation works and notifies user when invalid type is entered. Prompts again to enter type. If invalid type is entered again will notify again of invalid type until a correct type is entered, then will	I had a minor error where I misspelled “Goblin” and instead wrote “Golin”, this broke the validation for goblin but once this was fixed the expected outcome was achieved.

	player type Input: invalid player type input: correct player type		prompt for name of player.	
Verify that the doTournament function runs the game appropriately	None	None	The fight function is called, cout information informs user of team, type and name of players fighting correctly. The points are added to the winner correctly, and loser is added to the stack.	I tested this with multiple combinations of players, and all combinations of creature types. The fight function is called, cout information informs user of team, type and name of players fighting correctly. The points are added to the winner correctly, and loser is added to the stack.
Verify displayTopThree() function determines correct players, and couts info. appropriately	None	None	-The top three players with the most points would be determined and displayed.	I was running into an issue with this. I kept having a segmentation fault error. I thought that it was the sorting function from the STL but after reading the code I had written many times I realized that when I had made the vector of allPlayers to sort I did not add the players to the vector correctly, and was repeating some of the data. This caused the

				segmentation fault when I then tried to sort the vector.

Reflection:

After making the heal function I realized that I needed to account for the max strength that each specific was allowed to have. I realized that the way I had it written, allowed for a player to go beyond their max strength, ie if a Goblin won, was added back in line and then heal rolled a 6 and this was added to their current strength after a battle of 5, their new strength would be 11. This would be greater than the max strength for a Goblin and would result in an unfair advantage for future fights. I added in an if condition which checked the combined strength plus the roll and if it exceeded the max strength then the strength would be set to max strength.

The other big change I had to my design was the way I sorted the player for the function to display the first second and third players. At first I was going to create a function that would implement a bubble sort, but I wanted to see if there was an easier way. After doing some research I came across a sort function in the STL. I was able to implement this and it worked nicely. This involved making a function that I called creatureSort() that would return if the points value for one creature was greater than those points of another creature.

** I have left the test teams functions in the program and commented them out.