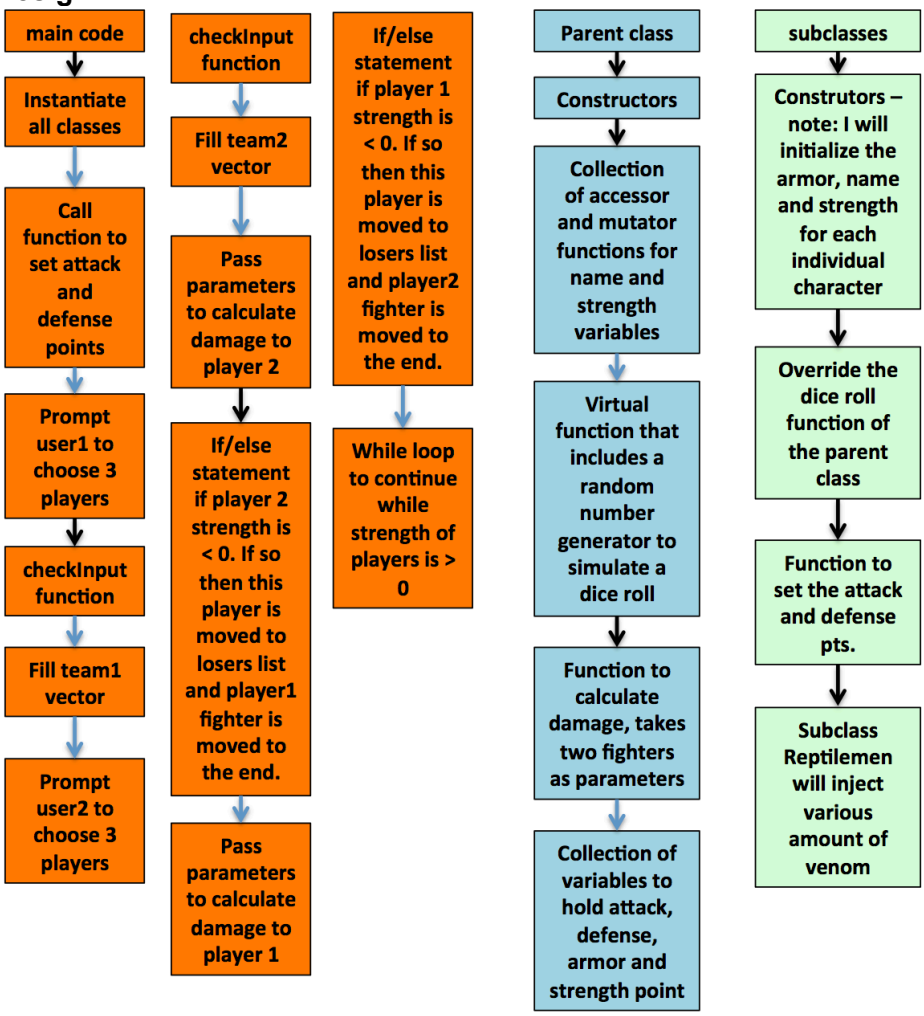
Jody Hansen
Final Project CS 162

**Understanding:**
For the final project we are modifying our last assignment 4 so that it is a tournament versus a single combat scene. Each user will choose three fighters, which will then be entered into a list to fight each other. Below I have included only the new requirements.

The requirements were as followed:

- Players pick multiple fighters to be entered into a tournament.
- The first picked will begin the combat.
    - The winner is moved to the back of the list.
    - The loser is moved to the standings list.
- Standings must include order of fighters, and from which player, along with their place (perhaps their strength points as well).
- Assumptions:
    - I removed a couple of areas of functionality from the last assignement, including the ability to dodge attacks and a choice of weapons for one of the subclasses.

**Design:**



**Implementation:**
The files are called Fighter.cpp, Fighter.h, subclassGoblin.cpp, subclassGoblin.h, subclassBarbarian.cpp, subclassBarbarian.h, subclassReptile.cpp, subclassReptile.h, subclassBluemen.cpp, subclassBluemen.h, subclassMaryLittle.cpp, subclassMaryLittle.h, and fantasyGame.cpp

**Testing:**

| Test Case | Input | function or flow tested | Driver function | Expected Outcomes | Observed outcome |
|---|---|---|---|---|---|
| Initialize collection of variables to 0 | NA | Default constructor parent class | Main() | Variables will be set to 0 | Variables will be set to 0 |
| Test if setupPts function calls correct diceroll function | Pass object | Set function attack and defense variables Parent class | Main() | Variables will be set to whatever specified | Variables will be set to whatever specified |
| Test if accessor functions return appropriate values | NA | Get function for attack, defense, armor and strength variables Parent class | Main() | Get functions return variable values | Get functions return variable values |
| Test if the diceroll function can return a randomly generated number for a dice roll | NA | Diceroll function Override in subclasses | Main() | Returns a number that is within the range for the times dice is rolled and how many sides it has | Returns a number that is within the range for the times dice is rolled and how many sides it has |
| Test if the subclass properly initialized the armor and strength variables in parent class | NA | Constructor subclasses | Main() Use parent get functions to return values | Armor and strength values are set according to each subclass requirements | Armor and strength values are set according to each subclass requirements |
| Test if the subclass can set the parents attack and defense point values with diceroll | NA | Diceroll subclasses | Main() Use parents get functions to return values | Attack and defense values are within the range for each specific class | Attack and defense values are within the range for each specific class |
| Test if reptilemen can vary the amount of "venom" they deliver | NA | setAttack value function need random number generator with high and low biases | Subclass reptilemen | Returns a damage value that is either biased high or low | Returns a damage value that is either biased high or low |
| Test calculate damage function | NA | CalculateDamage function() parent class | Main() | Calculates correct damage value | Calculates correct damage value |
| Test if the checkinput function only accepts integers 1-5 | 0<br>6<br>letters | checkInput function main | Main() | Users can only input integers 1-5 | Users can only input integers 1-5 |
| Test if the checkinput 2 function only accepts integers 1-2 | 0<br>3<br>letters | checkInput function main | Main() | Users can only input integers 1-5 | Users can only input integers 1-5 |

| Test Case | Input | function or flow tested | Driver function | Expected Outcomes | Observed outcome |
|---|---|---|---|---|---|
| Test changeAttack function | na | changeAttack function | Main() | Generates random number and matches user input choice. If match changes value to 0 | Generates random number and matches user input choice. If match changes value to 0 |
| If/else statement evaluates if strength is < 0 | NA | If/else statements | Main() | Breaks loop if < 0, continues if > 0 | Breaks loop if < 0, continues if > 0 |
| Runs game while players are "healthy" | NA | While loop | Main() | Continues to simulate a fight if strength is > 0 | Continues to simulate a fight if strength is > 0 |
| Test that each team vector holds appropriate fighters | 3 choices of fighters | | Main() | Team1 and team2 vector hold 3 players | Team1 and team2 vector hold 3 players |
| Test that loser of the round is added to loser list | NA | New vector container | Main() | Loser is added to new vector | Loser is added to new vector |
| Test that winner is added to back of list | NA | | Main() | Winner is added to back of team list | Winner is added to back of team list |
| Test that team with remaining players now fights itself | NA | | MAIN() | | |
| Test that loser of the round is added to loser list | NA | New vector container | Main() | Loser is added to new vector | Loser is added to new vector |
| Test that all players are removed from original vector | NA | | Main() | Team1 and team2 vector are empty | Team1 and team2 vector are empty |
| Test that the standings list is correct | NA | | Main() | Players are listed appropriately | Players are listed appropriately |

**Reflection:**

Considering that there were few requirements, I thought that the final project would be easy to finish. In fact, it didn't take me very long to write a program that implemented a tournament using vector containers and produced the correct standings. Despite testing the program and getting the desired result, I found myself wanting to emphatically repeat the phrase, "WHYYYY won't my program work." I started out with a slimmer version that used only two subclasses and everything appeared to work fine. However, when I added the other three subclasses, my program would occasionally freeze. At first, I did the usual. I walked back through my logic and sure enough I found a mistake where my program could access memory that it was not suppose to. However, the bug in my program persisted. As any one can imagine, I went through several rounds of thinking I found the problem until I knew I needed help. Some experienced programmers gave me some tips, but even after implementing their ideas to make my program better, I still had that nasty bug. I finally put in print statements everywhere! It turns out I made the rookie mistake of creating an infinite loop that was triggered by specific conditions. Even though I probably would of turned in my final project much earlier, I can't say that having the infinite loop in my code was a waste of time. My program evolved a lot throughout the process and I think it got better each time I tried to fix the bug. The bug forced me to keep working on it even though I had produced the desired functionality. The process drove home the concept of separate compilation. I was so relieved that I didn't need to test my parent and subclasses since I knew that for the most part they were already well tested. I at least knew that my problem was in my main function. Lastly, I learned the importance of a debugger, which had I known how to use I would of surely save time. So, in the future I would like to get comfortable with some type of debugger.

As for the programming part of the final project, I enjoyed learning how to use pointers and I now recognize how important they are. I wish I had started using them earlier. Perhaps future classes would benefit from having a few more requirements necessitating the use of pointers.