Mariam Ben-Neticha
CS 162: Intro to Computer Science II
11.22.2015
Assignment 4: Containers

Program Design:

Problem(s) to be Solved:

- Create a double-linked, circular stack to hold creature*, user-created name, and creature type
    - Solution: store 3 data points in each node with the corresponding types
- Prompt user for number of creatures each player will use
- Prompt each player for type of creature and name of each creature they choose
    - should have the same number of creatures for each player
- Instantiate each players creatures and store names, types, etc.
    - use for loop to determine what creature each player entered and add creature to the list with the user-defined name.
    - add virtual function in Creature Class to return the creature's identity type.
        - add function in derived classes to overload virtual function
- Create function to replenish points of winning creature between rounds
    - add function to Combat Class
- Create a winning system
    - Last creature to win the last round is the first place winner. Following winners are the survivors of winning team (in order), then the creatures in the loser-pile in reverse order than the way they entered (aka, first loser = Last Place)
- Print 1st, 2nd, and 3rd Placers
- Create option to display remaining rankings

Inputs:

- How many creatures each player will choose to play in the tournament?
    - integer value
- The creature types and names that each player (P1 and P2) want to use for their teams.
    - integer value to indicate creature type
    - string to enter creature's name

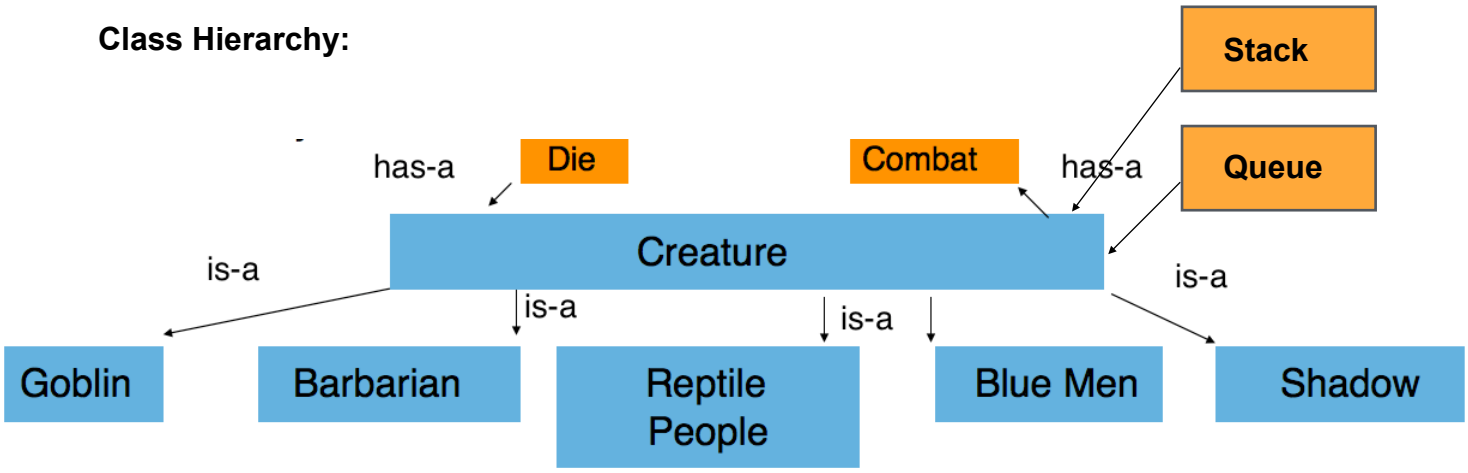
Outputs:

- Instructions for user to enter the number of creatures each player will use.
- Instructions for each player to enter a number of the type of creatures they choose and the names for those creatures.
- Display the winner and loser of each round in the tournament.
- Display which team won and how many points each team has.
- Display 1st, 2nd, and 3rd place winners.
- Display option to print out remaining rankings.

Algorithms:

- Calculate points for each team — use a counter.

Class Hierarchy:



Test Plan:

1. Does the 1st, 2nd, 3rd place rankings display properly?
   a. Yes: They are displayed properly with the correct Team #, creature type and creature name printed on-screen.
   b. No: They are displayed improperly with the incorrect Team #, creature type and creature name printed on-screen.
2. Is the User-Input received properly? And do the P1 and P2 lineup queues reflect the changes?
   c. Yes: The input is read properly into the appropriate queues. All actions reflect the appropriate changes.
   d. No: The input is not read properly. Incorrect changes are reflected in the round winner/loser displays and/or the ranking display.
3. Is the ranking list displayed as an option? Does it work properly?
   e. Yes: The option appears at the end of the tournament after the 1st, 2nd, 3rd place winners are displayed. This option functions properly.
   f. No: The option does not appear or this option functions improperly.
4. Does the tournament function properly with different types of creature combinations?
   g. Yes: The tournament functions properly regardless of the type of creatures fighting.
   h. No: The tournament malfunctions for certain creatures.
5. Does the winner of a round replenish it's points properly?
   i. Yes: The winner's strength points are incremented by 5 points.
   j. No: The winner's strength points are not incremented or are incremented by the wrong number of points.

Rankings:

Within this program, rankings are determined in the following way:

- First place is the last creature to play within a round and win.
- Second —> ? Place are the creatures remaining within the winning team's queue.
- ? —> Last Place are the creatures in the loser pile from last-in to first-in.

Test Results:

Test Number	Test Case	Input Values	Expected Outcome	Observed Outcome
1. Rankings Display (1, 2, 3)	N/A	N/A	Rankings displayed in order defined under 'Rankings' above.	Rankings displayed in order defined under 'Rankings' above.
2. User-Input Read-In	Input is user-input	Input for creature type is between 1-5 and input for creature name is one-word string.	Desired creatures are instantiated and placed into a p1 queue or a player 2 queue.	Desired creatures are instantiated and placed into a p1 queue or a player 2 queue.
3. Rankings Display Option	N/A	N/A	Rankings displayed in order defined under 'Rankings' above.	Rankings displayed in order defined under 'Rankings' above.
4. Tournament Functions Properly Regardless of Creature Type	User-Input any creature	User-Input any creature	Application functions properly. All displays are meaningful and clear.	Application functions properly. All displays are meaningful and clear.
5. Replenish Points Added	N/A	N/A	winner's strength points are replenished by incrementing creature's strength points by five.	winner's strength points are replenished by incrementing creature's strength points by five.

Tournament Test Results:

** **Definition of Tournament functioning properly:**

Proper instantiation of user-desired creatures — Proper storage of creature*, creature type, and creature name — Properly displayed winning and losing creature type and creature name after each round [indicating proper use of circular queue] — Proper display of first, second, and third place winners [in accordance to 'Rankings'-defined system] — Proper display of the option to show the remaining rankings — Proper display of the remaining rankings if chosen by user — Outcome of tournament is reasonable **

Test Number	Test Case	Input Values	Expected Outcome	Observed Outcome
1. Goblin vs Goblin	Input is an option Input is a string (user-defined)	Input = 1 && = 1	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.
2. Goblin vs Barbarian	Input is an option Input is a string (user-defined)	Input = 1 && = 2	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.
3. Goblin vs Reptile People	Input is an option Input is a string (user-defined)	Input = 1 && = 3	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.
4. Goblin vs Blue Men	Input is an option Input is a string (user-defined)	Input = 1 && = 4	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.
5. Goblin vs Shadow	Input is an option Input is a string (user-defined)	Input = 1 && = 5	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.
6. Barbarian vs Barbarian	Input is an option Input is a string (user-defined)	Input = 2 && = 2	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.
7. Barbarian vs Reptile People	Input is an option Input is a string (user-defined)	Input = 2 && = 3	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.
8. Barbarian vs Blue Men	Input is an option Input is a string (user-defined)	Input = 2 && = 4	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.

Test Number	Test Case	Input Values	Expected Outcome	Observed Outcome
9. Barbarian vs The Shadow	Input is an option Input is a string (user-defined)	Input = 2 && = 5	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.
10. Reptile People vs Reptile People	Input is an option Input is a string (user-defined)	Input = 3 && = 3	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.
Reptile People vs Blue Men	Input is an option Input is a string (user-defined)	Input = 3 && = 4	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.
Reptile People vs The Shadow	Input is an option Input is a string (user-defined)	Input = 3 && = 5	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.
Blue Men vs Blue Men	Input is an option Input is a string (user-defined)	Input = 4 && = 4	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.
Blue Men vs The Shadow	Input is an option Input is a string (user-defined)	Input = 4 && = 5	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.
The Shadow vs The Shadow	Input is an option Input is a string (user-defined)	Input = 5 && = 5	When tournament conducted, tournament functions properly.	When tournament conducted, tournament functions properly.

Reflection:

The first problems I needed to resolve and fix were edits to my Assignment 3 code. I had not used pointers to Creatures and as a result, the polymorphism was incomplete and the program encountered errors. I reviewed my assignment 3 code and edited the necessary functions within the Combat class in order to make the program work correctly and in order for the combat between two creatures to work properly. Once I got this part working as was necessitated by Assignment 3, I created a brand new main.cpp for Assignment 4 and linked necessary files.

At first, I had improperly read the directions and understood them to mean that the player's creatures needed to be stored dynamically. With this understanding, I created two dynamically allocated arrays to contain the creature pointers (Creature*)

instantiated based on the player's choices. However, It did not take me long before I re-read the assignment description and realized the creature pointers needed to be stored within a doubly-linked circular queue/linked structure. This was an early mistake, but I decided to begin anew once more.

On my second attempt, I first wrote down a list of requirements of the assignment on a piece of paper and created a checklist out of them. This ensured that no more simple mistakes would occur. I also outlined in human-speak a general idea as to how I could accomplish each requirement.

The first requirement was the simplest: Prompt the user for the number of creatures each player will have for the tournament. I wrote this cout statement with the user-input to be stored in a variable I created.

The second requirement was to prompt the 1st player to enter the creatures they want on their team along with the user-determined name for each creature. Before I began writing any code for this one, I edited my Queue class and determined what variables each node would need to store. Each node would need to store the creature pointer, the creature's type [i.e. Goblin vs Barbarian vs Shadow, etc.], and the creature's user-defined name. Once I inserted these into my Queue node, I instantiated the necessary queues within my main file. I created queues for player 1 and another set for player 2.

To read in the user-input for each player's creature types and creature names, I used a for-loop that compared the value input of the user with the matching creature value on the creature_display. Based on the user's input value, a specific creature would be instantiated using a creature*. Then, that creature*, the user-defined name, and the creature's type would be stored within a node inside that player's queue. The next creature would be read-in in the same manner until the user-defined number of creatures were entered. (To store the creature's type, I added a virtual identity() function within the Creature class and overloaded it in all the derived classes to return a string of the creature's type [i.e. The Shadow returns string "The Shadow"].)

At this point, I had completed the first three requirements of the assignment. I tested each by running the program and displaying each queue with a display() function from a previous lab.

The next requirement was to display the winner and loser of each round. To do this, I had to create a tournament. I created a tournament function that took two creature pointers as parameters and made them combat. This function was used in a do-while loop with creatures from both p1 and p2 lineups. First, the counter would increase for each round played. The round number would be printed on-screen and the winner and loser would be printed on-screen. Following this, the loser would be removed from their lineup and added to the loser_pile stack. Then, the loser team's lineup would be checked to make sure there were still creatures to play in the tournament. If there are still creatures available, the winning creature gets pushed to the back of it's lineup with

a move_back() function. Each time a creature wins a round, one point is added to their team's score. The next round occurs and the cycle repeats until one lineup is empty.

Once a lineup is empty, the winning team and the losing team are printed on-screen with their corresponding point-values. Teams win based on the number of round the creatures within their team have won. The team with the greatest number of wins is the overall winner.

Following this, I printed the first, second, and third place rankings based on my Ranking system described under 'Ranking' above. It states:

Within this program, rankings are determined in the following way:
- First place is the last creature to play within a round and win.
- Second —> # Place are the creatures remaining within the winning team's queue.
- # —> Last Place are the creatures in the loser pile from last-in to first-in.

This was fairly easy to code as I simply needed to repeatedly print the front of the queue and then delete that front in order to reach the next node. The last thing I needed to display was an option to allow the user to view the remaining rankings. I used a simple yes =1 and no = 0 in order to allow the user to indicate their preference with the least number of keystrokes. I used nested if-else statement to print the following rankings. The nested if-else statements and do-while loops checked whether the winning team's lineup was empty and printed-then-removed from that lineup if it was not empty. If that lineup was empty, then it printed all the creatures within the loser-pile in reverse order that they went into the pile [i.e. FILO].

I did run into a few problems when testing my code to make sure that it ran properly. One mistake I made was I had accidentally used a P1 variable to read-in a P2 name. However, it was done for only one creature. The result of this error was an error in the naming of one of the P2 creatures. This was a simple mistake and it was easy to fix.

When testing the code to make sure it ran properly, I entered five to six creatures at a time per player. I first entered creatures in the same order [i.e 1, 2, 3, 4, 5 where 1 = Goblin, 2 = Barbarian, 3 = Reptile People, 4 = Blue Men, 5 = The Shadow]. I allowed the program to run and, using paper and pen, I wrote down the names and types of each player. I then followed each round and determined the winner from the display. I ensured that the types and names of the creatures matched the user-defined types and names. I also ensured that the order the creatures fought corresponded to the assignment requirements [i.e. P1 first creature vs P2 first creature]. Finally, I double checked the points each team received and that the proper team was declared winner. Lastly, I checked the entire ranking list to make sure my Ranking system was being followed.

The next time I ran the program, I entered the creatures in reverse order [i.e. P1: 1, 2, 3, 4, 5 ; P2: 5, 4, 3, 2, 1]. This allowed a greater range of battles between different

types of creatures. I repeated the program with random creature types input. Each time, I checked the functioning of the program and determined that the program was functioning properly [as defined under ‘Tournament Test Results’ above].

The last piece I tested was the optional display of the rankings. Each time prior to this, I had entered 1(=YES) in order to view the rankings of the remaining creatures. I ran the program in order to ensure that it would function properly if I entered 0(=NO) and was able to finally declare that my program passed all the tests and completed all the requirements of the assignment with no errors.

Once I was sure that my program worked with no errors, I added a simple replenish() function within my Combat class that took a creature pointer as a parameter. My replenishing system was simple. I would add five strength points back to the winner of the round. This, to me, was fair across all creature types since all attacking dice could roll at least a 5 and so this did not create a disadvantage. At the same time, it rewards the winner for having survived the round. Finally, since it is impossible to determine the type of damage a certain creature will incur, I kept in mind that a creature could have maintained all their strength points within a round and simply added on extra points. I did not want this to occur and to create an invincible creature. I believe that five points was a good intermediary and that is what I implemented. I was able to test this by displaying the strength points of the creatures for my test. I went through each round and looked at each winner’s strength points prior to replenishment and after replenishment and made sure the value added was five points.