Benjamin Tate
2/14/16
CS 162 – Section 400
Assignment 3 Design and Reflection

Design:
(In blue are aspects of my design that were carried over from assignment 3, black lines are new)

Creature Base Class:
Protected data members: ints for the following: Number of sides on attack dice (attSides), number of attack dice (attCount), number of sides on defense dice (defSides), number of defense dice (defCount), attack and defense for the turn (attack and defense), armor, and strength. A string to hold creature's name (creatureName), ints to hold the creature's id number from 1-5 (id), score (score), starting strength (startingStr), and total damage (totalDamage).

Public member functions:

- Creature(int attSides, int attCount, int defSides, int defCount, int armor, int strength)
    - Initialize corresponding data members from parameter values.
- Virtual void attack() = 0
- Virtual void defend(int enemyAtt) = 0
- Int getAtt()
    - Return att
- Int getDef()
    - Return def + armor
- Int getStrength()
    - Return strength
- Int getId()
    - Return id
- String getName()
    - Return creatureName
- Int getScore()
    - Return score
- Void victory()
    - totalDamage = startingStr – strength
    - Initialize int variables for roll and heal amount
    - Roll between 1 and 20
    - Heal = roll * totalDamage * 0.05
        - (This way, there is some chance to how much gets healed after winning a battle)
    - Increment score

Barbarian Class:
No new data members

Member functions:

- Barbarian() : Creature(6, 2, 6, 2, 0, 12)
    - All members are shared with Creature, so nothing is needed in this constructor.
    - Set id to 1
- Barbarian(string name) : Creature(/*stats*/)
    - Does the same thing as default, but also assigns name to creatureName
- Virtual void attack()
    - Attack = 0
    - For(iterate up to attCount)
        - attack += random value from 1 to attSides
- Virtual void defend(int enemyAtt)
    - Int damage = enemyAtt - armor
    - Defense = 0
    - For(iterate up to defCount)
        - defense += random value from 1 to defSides
    - Damage -= defense
    - If(damage >= 0)
        - Strength -= damage

<u>Medusa Class:</u>
No new data members

Member functions:

- Medusa() : Creature(6, 2, 6, 1, 3, 8)
    - Set id to 2
- Medusa(string name) : Creature(/*stats*/)
    - Does the same thing as default, but also assigns name to creatureName
- Virtual void attack()
    - Attack = 0
    - For(iterate up to attCount)
        - attack += random value from 1 to attSides
    - if(attack == 12) ←Glare
        - attack = 50
- Virtual void defend(int enemyAtt) ←SAME AS BARBARIAN
    - Int damage = enemyAtt - armor
    - Defense = 0
    - For(iterate up to defCount)
        - defense += random value from 1 to defSides
    - Damage -= defense
    - If(damage >= 0)
        - Strength -= damage

<u>Vampire Class:</u>
Private data members: int charm (charm will be set randomly between 0 and 1 in defend(), if charm == 1, enemy will not attack)

Member functions:

- Vampire() : Creature(12, 1, 6, 1, 1, 18)
    - Charm will be set in defend(), so nothing is needed in this constructor.
    - Set id to 3
- Vampire(string name) : Creature(/*stats*/)
    - Does the same thing as default, but also assigns name to creatureName
- Virtual void attack() ←SAME AS BARBARIAN
    - Attack = 0
    - For(iterate up to attCount)
        - attack += random value from 1 to attSides
- Virtual void defend(int enemyAtt)
    - Defense = 0
    - Charm = random value from 0 to 1
    - If(charm == 0)
        - Int damage = enemyAtt - armor
        - For(iterate up to defCount)
            - defense += random value from 1 to defSides
        - Damage -= defense
        - If(damage >= 0)
            - Strength -= damage

<u>Potter Class:</u>

Private data members: int hogwarts

Member functions:

- Potter() : Creature(6, 2, 6, 2, 0, 10)
    - Initialize hogwarts to 1
    - Set id to 4
- Potter(string name) : Creature(/*stats*/)
    - Does the same thing as default, but also assigns name to creatureName
- Virtual void attack() ←SAME AS BARBARIAN
    - Attack = 0
    - For(iterate up to attCount)
        - attack += random value from 1 to attSides
- Virtual void defend(int enemyAtt)
    - Defense = 0
    - Int damage = enemyAtt - armor
    - For(iterate up to defCount)
        - defense += random value from 1 to defSides
    - Damage -= defense
    - If(damage >= 0)
        - Strength -= damage
    - If(strength <= 0 and Hogwarts > 0)
        - Strength = 20

        - Hogwarts -= 1

<u>BlueMen Class:</u>
No new data members

Member functions:

- BlueMen() : Creature(10, 2, 6, 3, 3, 12)
    - All members are shared with Creature, so nothing is needed in this constructor.
    - Set id to 5
- BlueMen(string name) : Creature(/*stats*/)
    - Does the same thing as default, but also assigns name to creatureName
- Virtual void attack()
    - Attack = 0
    - For(iterate up to attCount)
        - attack += random value from 1 to attSides
- Virtual void defend(int enemyAtt)
    - Int damage = enemyAtt - armor
    - Defense = 0
    - For(iterate up to defCount)
        - Defense += random value from 1 to defSides
    - Damage -= defense
    - If(damage >= 0)
        - Strength -= damage
    - If(strength <= 4)
        - defCount = 1
    - else if(strength <= 8)
        - defCount = 2

<u>Tournament Class:</u>
Private data members: int to keep track of current battle number (battleCount), two Queue objects to hold the teams (t1 and t2), and a Stack object to hold the loser pile (losers).

Member functions:

- Tournament(Queue team1, Queue team2)
    - Initialize battleCount to 1
    - Assign team1 and team2 to t1 and t2, respectively
- Battle()
    - Assign first member of t1 and t2 to Creature pointers c1 and c2 (using .remove())
    - Initialize round count to 1
    - While both Creatures' strength > 0
        - C1 attacks, c2 defends, print attack and defense power, and c2's remaining strength
        - C2 attacks, c1 defends, print attack and defense power, and c1's remaining strength

        - Increment round counter
    - If c1 wins
        - C1.victory()
        - T1.add(c1)
        - Losers.add(c2)
    - If c2 wins
        - C2.victory()
        - T2.add(c2)
        - Losers.add(c1)
    - Otherwise (draw)
        - Add both c1 and c2 to losers (if you aren’t first, you’re last)
    - Increment battleCount
- Results()
    - Create creature pointer temp
    - Determine and print winner
    - Remove remaining creatures from winning team and add to losers for sorting (using temp as proxy)
    - Create first, second, and third creature pointers.
    - Repeat until losers.getSize() is reached:
        - Pop top of stack into temp
        - If temp.getScore() > first.getScore()
            - Shuffle first, second and third back
            - First = temp
        - Else if temp.getScore() > second.getScore()
            - Shuffle second and third back
            - Second = temp
        - Else if temp.getScore() > third.getScore()
            - Third = temp
- getT1()
    - return t1 Queue
- getT2()
    - return t2Queue

<u>Queue Class:</u>
I will use the Queue class from lab6, but I will alter it to use pointers to Creature objects instead of ints. I will also need to add a private int to keep track of the size of the Queue (queueSize), and a public getter for queueSize (getSize()).

<u>Stack Class:</u>
I will use the Stack class from lab6, but I will alter it to use pointers to Creature objects instead of ints. I will also need to add a private int to keep track of the size of the Stack (stackSize), and a public getter for stackSize (getSize()).

Testing plan and results:

Since there are too many team combinations to possibly test, I will do my best to get a wide spread of team sizes and composition. I will include print statements in new functions to ensure that they are behaving properly (e.g. print heal amount and strength after heal in heal function). In each test, I will be checking to make sure that everything went well throughout each battle. Of primary interest will be the heal functions and the add functions during battles, as well as the determination of first-third place Creatures. I will check the heal function through print statements, as mentioned above (looking for heal values that exceed maximum allowed, are negative, etc.), the Queue add function by checking to see if the winning creature battles again at the right time, and the first second and third place winners by tallying wins myself.

Test 1:
Team size = 2
Team 1 = Barbarian a and barbarian b
Team 2 = Barbarian c and barbarian d
Observations: Heal and add seem to work correctly, but top three contain a repeat (2nd and 3rd are the same).
Altered top 3 functionality of results() as outlined in reflection

Test 2: (excluding multiple tests to get top 3 to work properly)
Same as test 1
Observations: everything works right.

Test 3:
This test was repeated a few times, because team 2 kept winning every battle, making it impossible to assure that add() was working right for the Queues
Team size = 3
Team 1 = Barbarian b1, barbarian b1.1, medusa m1
Team 2 = Vampire v2, Potter p2, BlueMen x2
Observations: All random rolls seem well within range, heal, add, etc. seem to work well.

Test4: (And onward)
For test 4 onward, I set up a new file to replace my main, omitting prompts so that I could more quickly run the same test over and over, I then included an Example command in my makefile to compile with the test file instead of my main
Team size = 4
Team 1 = BlueMen x1, Medusa m1, medusa m1.1, vampire v1
Team 2 = vampire v2, potter p2, potter p2.1, barbarian b2
Observations: Ran the test upwards of a dozen times to view as many result combinations as possible, observed everything as mentioned above with each test, found that everything was well within reasonable expectations.

Reflection:

Once I got everything compiling (had some trouble with pointers in my Stack and Queue classes that needed resolving), most of my program worked as planned. The only thing that I couldn't get to work right was the top 3, as mentioned above. I ended up going an entirely different route for determining the top 3 deciding that going by who survived the longest would be a better way of determining it than score anyway, since there were always so many ties. To do this, all I had to do was pop the top 3 off of the losers stack (after adding the surviving winners to the stack) and store them in first, second, and third. I kept the score and getScore() members in my program so that I could still announce the number of wins each of the top 3 had, because I thought that would be cool to see too.

My creature class still has a member variable int id and a member function getId(), and each creature type has a unique id set in the constructors from 1-5. My original idea with this was to be able to output winner names and types (e.g. "Borris, the Barbarian, wins the battle!"), but that turned out to be a pain to implement correctly, so I scrapped that, since it wasn't required. However, I kept the id-related stuff in my program, in case I wanted to improve my program later.

Once I got over the top 3 hurdle, testing was pretty smooth. It was something of a pain keeping track of all of the things that could go wrong, but I approached it with a systematic tally system on paper to make sure I observed everything that I needed to in each test.