Kyle Karthauser
Project 3
May 13, 2018

Inheritance Chart
BASE -> DERIVED
Character -> Barbarian (Unmodified inheritance of attack and defend functions)
Character -> Vampire (Modified attack function)
Character -> Medusa (Modified attack function)
Character -> Bluemen (Modified defense function)
Character -> Harry Potter (Modified defense function)

Program Design
Making the Characters Fight
--Vector of Character* pointers
--Menu asks user who will fight
--Populate the vectors
--Iterate over the vectors
--Attack and defend values are mediated by another vector or int which will also allow me to regulate charm and glare and other effects

Program Implementation Issues
Problem 1
At first, I implemented a simple routine to handle the matches:

1. Fighter 1 attacks. Attack function returns damage dealt.
2. Fighter 2 calls defend(damage_dealt) function. That calculates strength loss (or no change) depending on the armor and defense rolls.

But the special abilities proved too complicated for this setup. If, for example, a Vampire cast charm and it was fighter 2, then charm had no effect--there was no way for me to transfer that back to fighter 1attack.

To solve this, I added a data member to character: bool getsAttack. I thought I could flip this true and false depending on the situation. But in my if and switch statements, I couldn't come up with anything satisfactory. It all seemed too messy. Checking and setting and this all the time inside a loop seemed like a mess.

Next, I decided to add a parameter to my attack function. If the character receives a -1 value in its attack call, it won't attack. This would work for Medusa as well, as it would simply preempt her ability to call glare, and therefore the vampire would always win.

However, another wrinkle: If the vampire casts charm, then in my setup, it couldn't also deal damage. I could have the damage rolled specially inside each character class's "defense" function. But that seemed assinine. So I needed a way to pass two ints to the match function. I could do this with a string like "10, -5", where 10 would be damage dealt and -5 a special ability--or, I could pass a struct which would be "opened up" and read by the function.

Then I smacked my forehead with my palm, because I remembered how easy it is to pass an array. So I changed my returns to pointers to arrays and my defend functions to accept arrays.

Problem 2
Again and again I'm running into the problem of sequence: Player 1 attacks and maybe uses a special ability; player 2 defends, then attacks (and maybe uses a special ability). But if Player 2 has already been attacked, the special ability is meaningless. It's cast "too late" to register on the attacker and, for example, nullify their move (or kill them).

Kyle Karthauser
Project 3
May 13, 2018

So I came up with the Match class & function set. Match holds the two fighters in a vector (vector<Character*> fighter {2}) and acts as an arbiter between the two. Both attack simultaneously; each class can handle the output; but match is there to organize data, track data, and return “false” if someone dies.

The “false” return value ties into the menu function set.
Menu is called to set up the match, including naming the characters. Once the characters have names, they are created. Once they are created, a Match is created, as the Match constructor requires two Characters.

This is much more complicated than my initial design, which simply had two characters exchanging attack information!

Problem 3
I wanted to give myself the ability (in my test driver program) to re-run the match with the same characters, different characters, or stop.
If I ran it with the same characters, it wasn’t resetting their strength and special attributes.
If I ran it with different characters, it wasn’t resetting the round count.
It turns out that there’s always something more to learn about clearing memory!
Because the program wasn’t going out of scope, destructors weren’t automatically being called. I added an explicit call to destruct “match” and that cleared up the issue.

Test Case	Input Values	Driver Function	Expected Outcome	Observed Outcome
User asks for a Barbarian/Vampire /Medusa/Harry Potter/Blue Man	name	setupMatch, Character::charTyp e constructor();	Appropriate character type is created with chosen name	Appropriate character type is created with chosen name
User asks to re-run program with different chars/same chars	“2”	Match::~Match(); Match::(Match(); setupMatch();	New match is created; no old information hangs around -- OR, same match is run, same characters have refreshed strength and so on	Old match information persisted (see Problem 3 above for fix)
Vampire uses special ability	none	Vampire::attack();	Opponent attack nullified	Opponent attack nullified
Medusa uses special ability	none	Medusa::attack();	Opponent strength set to 0	Opponent strength set to 0
Vampire uses charm again Medusa’s glare	none	Vampire::attack(); Medusa::attack(); Match::OneRound()	Neither fighter lands a hit; both survive to fight	Neither fighter lands a hit; both survive to fight

		;	another round	another round
Fighter 1 against fighter 2	none	setupMatch(); Character::attack(); Character::defend(); Match::oneRound(); ;	Die is rolled appropriate # of times; proper attack values calculated and displayed; proper char attributes displayed; attack and special attacks are handled; strength is modified appropriately; death check occurs	A-OK