CLASSES

- Die
    - Public:
        - Number of sides
    - Private:
        Constructor
        - Set sides
- Loaded Die (inherits Die)
    - Public:
        - Number of sides
        - “Weight” factor to load the die
    - Private:
        - Constructor
        - Set sides
        - Set weight

1. Menu prompts user for options
2. When die is created, the default constructor handles user interaction and setting parameters
3. When loaded die is created, the same as above
4. Game displays characteristics of dice
5. Game rolls dice
6. Game records and displays all rolls, judges individual round winners, and lastly overall winner

ACTUAL

1. I ended up changing part 2 & 3. The object constructors do not handle initialization. I decided to use a vector and vector.emplace_back in order to build it out. The problem (as I saw it) lay in giving the user choice to select the type of dice. If we did two normal dice, or two loaded, or a mix, how would I then pass this on to program and make it “play” with these dice? I didn’t like the idea of hardcoding it and I couldn’t think of any other options.
2. I thought I could initialize each die with “new Die/new LoadedDie” and create a customized default constructor. But when I call new LoadedDie, I get Die()’s constructor too. I knew that this would happen in principle because of the reading I did on the topic, but sometimes you don’t really learn until the rubber meets the road!
3. Initially I had “n” (number of sides) set to “private” in my “Die” class. But it turns out that it was then hidden from my derived “LoadedDie” class. I switched “private” to “protected.”
4. I waffled between using the “friend class” distinction and “virtual” functions in my class definitions. I eventually settled on virtual (for the roll() function) as that seemed less complicated and more parsimonious.
5. I beefed up my input checking functions for this lab. They’re contained in “yomu.hpp/cpp” (yomu is Japanese for “read”). While implementing them, I finally realized the situation that demands if - else if - else if - else: When you need each and every “if” statement to be true in order to be proceed, use else if. If not, use if. The specific issue that arose here lay in my using simple “if” statements within a do loop. The do loop should have repeated until satisfied by input, but instead it was registering invalid input and then returning the invalid input! It didn’t ask the user to re-input input. Using if, else if fixed this.

Test Case	Input Values	Driver Function	Expected Outcome	Observed Outcome
User asks for a loaded die	Number of sides, “weight” of die	LoadedDie(int sides, double weight)	Loaded die with proper sides and weight created	CORRECT
User inputs decimal value when integer is appropriate	1.5	intYomu()	Error	CORRECT
User inputs integer value when decimal is appropriate	5	doubleYomu()	Error	CORRECT
User chooses to play with two standard dice	Standard die, standard die	menu(), Die::Die(), game()	Standard dice are created and game proceeds without error	CORRECT
User chooses to play with two loaded dice	Standard die, standard die	menu(), LoadedDie::LoadedDie(), game()	Loaded dice are created and game proceeds without error	CORRECT
User chooses to play with one standard die and one loaded die	Standard die, Loaded die	menu(), Die::Die(), LoadedDie::LoadedDie(), game()	Standard die and Loaded Die are created and game proceeds without error	CORRECT