- Practice extra make big number print english spelling
    - Keep dividing by 10 and map it back on to the words
- To find how many bits it takes to store a number take log base 2 of it
- Can we take integer value and direct the order of a deck from it
    - It is 52! Amount of ways a deck can be shuffled
    - There is no good way of doing it
    - Can we swap out Rank and Suit with something more compact
    - SizeOf(); will tell you how many bits it is
        - Cards are 8 bytes the way we did it
        - Can we compact it
        - Having private data members we can redesign the class to do it without having their implementation to change
        - We can represent suit as 2 bits
            - 00,01,10,11
    - Unsigned char data;
        - //is 8 bits
        - //stores the rank and suit as follows
        - * can make them multiples of 10s so spades would be 0's and hearts would be 10's and so on
            - Then comparing cards would be easier to compare the higher num
                - If suits are ranked as well
        - Bit manipulation to unsigned char
            - //00ssrrrr
            - //where ss are the two bits needed to store the sui
            - //and rrrr is the four bits needed to store the rank
                - Left shift multiply it by 2 need it to shift 2 times so multiply it by 4
    - Card(Rank r, Suit s)
        - data(static_cast<unsigned>(s) << 4|static_cast<unsigned>(r))
            - a<<left shift
            - Second r does not be shifted it is 0-12
            - | bitwise or
                - 1 or 1 is 1
                - 1 or 0 is 1
                - But with bits so it is 1s and 0s
                - Van combine bits and bit masking
            - Bitwise operation is a lot faster than integer multiplication and division
                - Division is slow
    - Rank get_rank() const {return static_cast<Rank>(data &0xf);
        - & mask of and
            - 1 and 1 is 1
            - 1 and 0 is 0

        - Now we are selecting the low 4 bits and getting the rank and converting it to that
    - Suit get_suit() const {return static_cast<Suit>(data >> 4);}
        - Any bits at end that fall off dont get cycled around just become 0s
    - Struct Deck : std::array<Card,52>
        - Lot less bits
        - Using std::array<Card.52>
            - Not a thing so
            1. Deck(initilization_list<Card>list)
            2. {
            3. std::copy(list.begin(), list.end(), begin());
            4. }
    - Sometimes you need a default constructor
        - We now only need 52 bits of data for our program which was at least 52X8 larger
            - So making them private we can change it without changing how it works
        - It is no unmuttable though
            - Can't take off top or put on bottom new
- Classes
    - Can represent some value abstructly
    - Card does not have mutable numbers after initialization
    - Helps with abstract representation
        - Rational number class shouldn't have 0 on bottom so it needs to catch that do you reduce the rational num or learn it as original fraction
    - Data structures
        - All mutable is someway
            - Can be modified in some way
    - Also has mutable data
        - Datas that can be changed over time
        - Like player health
        - Concrete entities
    - Resources
        - Anything will limited access of them
        - Memory
    - Function
        - Machine devices
    - These are the parts of a class and what a class can hold
    - To hand off function to other sub processes
        - You can add a third argument to sort as a function
            - f()
                - Has back one.author < book2.author
            - sort(x.begin(), x.end(), f());

- Practice 3
    - Due feb 14
    - Could make it for every card game
        - With generalized functions that are shared between games