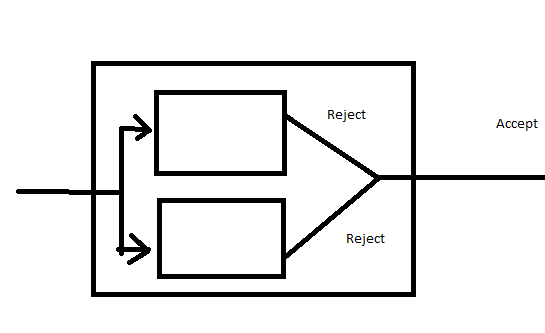
1. Design single-tape Turing machines that accept the following languages using JFLAP
   1. **L2 = { w : na(w) = nb(w) : w{a, b}+ }.**
      1. First state will need to transition to an A or B scenario that will be iterated through
      2. Scenario A
         1. We will want to mark off an a as X, and then make our way to the nearest b.
         2. Once we reach that b, we will replace it with a Y,
         3. then we will iterate back to the beginning, skipping over our X’s and Y’s then pickup the next a or b
      3. Scenario B
         1. We will want to mark off a b as Y, and then make our way to the nearest.
         2. Once we reach that a, we will replace it with an X,
         3. then we will iterate back to the beginning, skipping over our X’s and Y’s then pickup the next a or b
      4. After we’ve gone through all the a’s and b’s and transformed them into X and Y, we will want to check that our string doesn’t have any remaining by skipping over all the X’s and Y’s
      5. If we reach the blank end of the tape, this can be accepted
   2. **L3 = {ww : w{a, b}+ }**
      1. This one was a bit tough to implement, but I'll do my best to explain what I did
      2. We need to allow a or b to be our initial variable, and use it to help determine the start of the second w.
      3. We will iterate through the whole string, grabbing any a or b respectively and using it as a test for the middle
      4. When we reach an a or b, we will mark it off as a possible start of the second string, then iterate back to the beginning
      5. We will now compare our assumed initial string against our assumed second string, this will be separated into two scenarios
      6. Scenario 1
         1. If we get an a, we will blank it out, then make our way to our marked off assumed start of second string
         2. We will now iterate through all the ‘mirrored’ numbers between the strings that we have matched
         3. Finally, when we reach an a that has not been marked off, we will mark it off
      7. Scenario 1
         1. If we get a b, we will blank it out, then make our way to our marked off assumed start of second string
         2. We will now iterate through all the ‘mirrored’ numbers between the strings that we have matched
         3. Finally, when we reach an a that has not been marked off, we will mark it off
      8. We now want to iterate back to the beginning of our string
      9. When we reach the blank, we will go to the beginning of that prior scenario process
      10. If we only have our marked off variables remaining, we know that the strings match, and we have a situation where ww exists
2. Design Turing Machines using JFLAP to compute the following functions for x and y positive integers represented in unary. The value f(x) represented in unary should be on the tape surrounded by blanks after the calculation.
   1. **𝑓(𝑥) = { 𝑥 − 𝑦, 𝑥 > 𝑦 0, otherwise**
      1. We will take in the first 1, then iterate to the last 1 after the - sign
      2. We will pop that 1 off of the problem, and then we will iterate back to the beginning of the problem, and pop the very first 1 that appears
      3. We will iterate through this problem until one of two scenarios occurs
      4. Scenario A
         1. We run out of variables before the - sign
         2. We will want to pop everything that remains in the problem off the tape, and then replace the last item with a 0
      5. Scenario B
         1. We run out of variables after the - sign
         2. We will want to pop off the - sign from the problem, and set our pointer to the beginning of the problem
   2. **𝑓(𝑥) = 𝑥 𝑚𝑜𝑑 5**
      1. This one got a little messy, but I think I have a working solution
      2. We will intake up to 5 1’s at a time, we can make sure it is five by marking them off
         1. if this part receives any number less than 5, we will want to un-mark them, and then send the remainder of the string to the user, so we will iterate back through until out pointer is at the beginning, then send the result to the user
      3. If we have 5 or more values, we will proceed to pop the first 4 of them off the stack, and make the last one a 0
         1. If there exists no other values after the 0, we can send a 0 to a final state, telling the user that there are mod 5 values by returning
      4. If there is a value after the 0, we will pop the zero off of the stack, and start the process over again
3. Prove that recursive languages are closed under the nor operation.
   1. Let m1 be a turing machine that will accept l1
   2. Let m2 be a turing machine that will accept l2
   3. w1 contains l1
   4. W2 contains l2
   5. For w1, m1 halts in accepting state
   6. For w2, m2 halts in accepting state
   7. For some (w does not contain l1) and (w does not contain l2) the machine will halt without entering an accept state
   8. Then there is a turing machine M that shows
      1. If on w, m1 and m2 will both halt without entering to the accepting state, then M will accept
      2. Else, M halts without entering into the accepting state
   9. Since M1, and M2 are both going to halt, M will halt, therefore this will be a turing machine