In my gradebook, I've written

\[ 52 + C \]
\[ 68 + B \]
\[ 84 + A \]

for this exam.

Done. Making Turing machines do stuff:

We can interpret a TM as giving output - whatever is left on the tape at the end. Instead of just saying "Yes" or "No!"

For programming TMs, it's easiest to deal with numbers in "unary" - the # is the number of 1's,
Write a TM that, given a number \( n \) as input (in unary), outputs \( 2n \). (i.e., if we start with

\[
\begin{array}{c}
\text{V} \\
\circ \quad \text{□□□□□□□□□□□□□□□□□□□□} \\
\text{we end up with}
\end{array}
\]

\[
\begin{array}{c}
\text{V} \\
\circ \quad \text{□□□□□□□□□□□□□□□□□□□□} \\
\end{array}
\]

\[\begin{array}{c}
(\lambda, q_0, \text{□□□□□□□□□□□□□□□□□□□□}) \\
5
\end{array}\]

\[\begin{array}{c}
(\lambda, \text{some state}, \text{□□□□□□□□□□□□□□□□□□□□}) \\
10
\end{array}\]

Start state \( q_0 \):

\[
(q_0, 1) \rightarrow (q_1, x, R)
\]

\[
(q_1, 1) \rightarrow (q_1, 1, R)
\]

\[
(q_1, \text{□}) \rightarrow (q_2, y, L)
\]

\[
(q_1, y) \rightarrow (q_1, y, R)
\]

\[
(q_2, y) \rightarrow (q_2, y, L)
\]

\[
(q_2, 1) \rightarrow (q_2, 1, L)
\]

\( q_1 \) - marked a 1 - need to copy it

\( q_2 \) - copying done - move left to find next 1 to copy
Problem:

\[
\begin{array}{cccccccc}
\text{x} & \text{x} & \text{i} & \text{i} & \text{i} & \text{i} & \text{i} & \text{x} \\
\text{x} & \text{x} & \text{x} & \text{i} & \text{i} & \text{i} & \text{i} & \text{i} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{y} & \text{y} & \text{y} & \text{y} & \text{y} & \text{y}
\end{array}
\]

\[(q_2, x) \rightarrow (q_0, x, R)\]

---

How do we tell we are "done," and how to finish from there?

\[
\begin{array}{cccccccc}
\text{x} & \text{i} & \text{i} & \text{i} & \text{i} & \text{i} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{y} & \text{y} & \text{y} & \text{y} & \text{y} & \text{y}
\end{array}
\]

\[(q_0, y) \rightarrow (q_3, y, R)\]

\[(q_3, y) \rightarrow (q_3, y, R)\]

\[(q_3, \text{□}) \rightarrow (q_4, \text{□}, \text{L})\]

---

When moving left we need to treat these X's differently. Add y's instead.

Clean-up when see a y in state q_0. q_3 - move to end in clean-up phase.
(q_4, y) \rightarrow (q_4, 1, L)
(q_4, x) \rightarrow (q_4, 1, L)
(q_4, \square) \rightarrow (q_f, \square, R)

94- move left, converting everything to 1's, in cleanup.

What about multiplying 2 numbers?

\[
\begin{array}{c|ccccc}
\cdot & 1 & 1 & 1 & 1 & 1 \\
\hline
6 & 6 & 6 & 6 & 6 & 6 \\
\end{array}
\]

Output should be

\[
\begin{array}{c}
1 & 1 & \cdots & 1 \\
\hline
1 & 8 \\
\end{array}
\]

Strategy:
Mark each a 1 in a,
make a copy of b each time (copying involves marking and unmarking).
When all the 1's in a are marked, we are done, erase a (replace with blanks) and unmark all copies.
copies of b.

Given an input n,

if n is even, output $\frac{n}{2}$
if n is odd, output n+1.

There are lots of ways to check even - a finite automaton can check this by just
switching btw 2 states.

(Note - a DFA is equivalent to a TM where you cannot move L)

B/c we'll need to divide by 2,
and squishing a string (after some middle bits are removed) is
annoying, we'll match beginning w/ end.
1. Mark the beginning with an x, move to end (a 0 or a y) and mark with y, and return to beginning (the letter after the x).

2. If we end up at
   
   \[ \times x x y y \]
   
   and try to mark an x as a y, then n is odd.

3. If we end up at
   
   \[ \times x x y y y y \]
   
   and try to mark a y as an x, then n is even.

   Change all the y's to 1's and erase all the x's.