CMSC 303 Introduction to Theory of Computation, VCU
Assignment: 5
Name: Steven Hernandez

1. (a) Configuration of a Turing Machine consists of
   i. State
   ii. Head position
   iii. Tape contents

   (b) Yes, a turing machine can write anything that is in the tape's alphabet Γ.

   (c) Γ can $= \Sigma$, however this may cause ambiguity. A blank space usually indicates the end of the string, but if $\sqcup \in \Sigma$, there is no knowing for certain the character indicates the end of the string.

   (d) It depends on how a turing machine is defined. By our definition, a step for the turing machine either moves the head left or right. The head cannot remain in place.

   However, a turing machine can be defined to allow a step to move the left, right or stay in place. This new definition of a turing machine is equal in power to the previous turing machine. This is because we can simply take any steps that say to stay in place and separate them into two steps: move left, then move right. This effectively keeps the head in place.

   (e) By our definition the turing machine's set of states $Q = \{q_{accept}, q_{reject}\}$.

   It could be the case that the turing machine will never halt in one of those two states, but still the state is still present in Q.

   (f)

2. $L = \{x | x$ contains twice as many 0s as 1s$\}$.

   $M =$ "On input string w:

   (a) Starting from the left, move towards the end of the input to ensure there are an even number of zeros, if not, reject.

   (b) Return to the left, scan right until the first uncrossed zero. Cross it off.

   (c) Continue right until the next zero. Cross it off as well.

   (d) Move all the way to the right most symbol in the input.

   (e) Scan left until a non-crossed-off one is found. If no one is found, reject. Otherwise, cross off the one and return the head to the left.

   (f) Repeat the second step until no zeros are found.

   (g) Once scanning from left to right finds no zeros, check to see if there are any uncrossed ones. If there are, reject. Otherwise, accept.

3. (a) First, we determine what sort of differences there are between the two models.
   i. If the head of our turing machine is on the first symbol of the tape and then is instructed to move left, the head stays in place. Alternatively, the doubly infinite tape will successfully move left.

To simulate this feature of our standard Turing Machine, we simply add a few conditions to our new TM. We begin our turing machine by marking the first symbol in some unique way so as to identify it as our initial position. Then we add the condition that if we try to move left but we read this unique identifying mark, we instead remain in place (or move left, then automatically move right).

(b) Not caring about any efficiency, we again mark our first place on the tape with a unique identifier. (Note this does not remove our turing machines ability to read the symbol below the mark). Then, whenever we try to move left when we see this unique identifying mark, we shift all the contents of the tape over to the left once. After shifting, we remove the unique identifying mark and place it on the new first space on the tape (which we will assume would be filled with a blank space symbol).

(c) Because the models can both recreate the mechanisms of one another, this means both are able to recognize the same set of languages.

4. (a)

   (b)

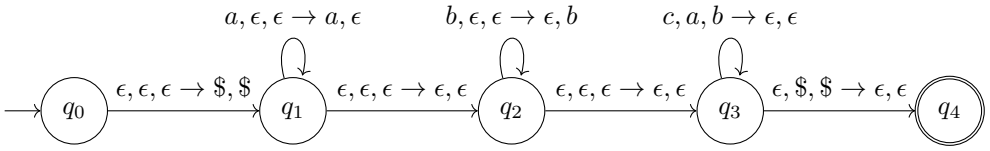
5. Understanding a 1-PDA can recognize context-free languages, and knowing $L = \{a^n b^n c^n | n \geq 0\}$. We can simply show that a 2-PDA is more powerful, because it can recognize L.

   Beforehand though, we can understand that a 2-PDA is at least equivelant to a 1-PDA, because while the 2-PDA has two stacks, we can leave a stack alone during some computation and we it can effectively be used as a 1-PDA.

   To show L is recognized by a 2-PDA.

   We can do this with a state diagram as seen below. Remember, a 1-PDA's transitions are labeled $x, y \to z$ where x is the symbol read from the input, y is the symbol popped off the stack and z is the symbol pushed onto the stack.

   For our 2-PDA, we label transitions: $x, y_1, y_2 \to z_1, z_2$ where x again is the symbol read from the input. y_n is popped from stack n. z_n is pushed onto stack n.



   Thus, a 2-PDA can recognize at least on additional language, thus a 2-PDA is more powerful than a 1-PDA.