1) a) \[ \begin{align*}
&\langle \text{EXPR} \rangle \\
&\langle \text{TERM} \rangle \\
&\langle \text{FACTOR} \rangle \\
&\langle \text{FACTOR} \rangle \\
&d
\end{align*} \]
\[ a + a \]

b) \[ \langle \text{EXPR} \rangle \\
\langle \text{FACTOR} \rangle \\
\langle \text{FACTOR} \rangle \\
\langle \text{FACTOR} \rangle \\
\langle \text{FACTOR} \rangle \\
\langle \text{TERM} \rangle \\
\langle \text{TERM} \rangle \\
\langle \text{FACTOR} \rangle \\
\langle \text{FACTOR} \rangle \\
\langle \text{FACTOR} \rangle \\
( \langle d \rangle ) \]

2) a) \[ S \rightarrow 0S0 | 1S1 | 1 | 0 | \epsilon \]

All palindromes are built by padding any 0 or 1 length string with the same character at the front and end repeatedly. The first two rules enforce the latter idea and the last three are the "base case" strings.

b) \[ S \rightarrow 0S1 | 1A \]
Follows the OPA CFG construction.
\[ A \rightarrow 0A | 1B \quad \text{S = zero \, 1's, A = one \, 1} \]
\[ B \rightarrow 0B | 1C \quad \text{B = two \, 1's, C = three \ or \ more \ 1's} \]
\[ C \rightarrow 0C | 1C | \epsilon \quad \text{are where each VARIABLE/STATE aligns for} \]

c) \[ S \rightarrow 0S0 | 0S1 | 1S0 | 1S1 | 0 \]
The only "base case" is \( \epsilon \). We successively add a single character to the left and right of this middle \( 0 \) to create a string in the language.

d) \[ S \rightarrow A1A \]
\[ A \rightarrow A1A | A1A0A | A0A1A | \epsilon \]
We force at least one \( 1 \) by going from \( S \) to \( A \).
Any time we add a \( 0 \), we also add a \( 1 \), not restricting order.
3) \[ Q_0 \xrightarrow{\varepsilon, 0} Q_1 \xrightarrow{\varepsilon, 1} Q_2 \xrightarrow{\varepsilon, \varepsilon} \]  

\[ Q_0 \xrightarrow{0, 0} Q_1 \xrightarrow{0, 1} Q_2 \xrightarrow{1, 0} Q_2 \xrightarrow{1, \varepsilon} Q_0 \]  

\[ Q = \{ Q_0, Q_1, Q_2, Q_3 \}, \Sigma = \{ 0, 1 \}, \Gamma = \{ 0, 1, \varepsilon \}, \delta \text{ is above} \]  

\[ Q_0 \text{ is the start state}, \quad F = \{ Q_3 \}. \]

\[ Q_0 \text{ marks starting the string processing by requiring $\varepsilon$ at the bottom of the stack.} \]

\[ Q_1 \text{ is the state where all string processing occurs. We keep track of a symbol by either pushing it on the stack or popping its "opposite" off the stack.} \]

\[ Q_2 \text{ forces there to be at least one } 1 \text{ on the stack after processing all characters.} \]  

We also ensure that NO zeros are on the stack by providing no transitions from $Q_2$ that pop a 0.

We can only get to $Q_3$ after getting to $Q_2$ with no zeros on the stack, after all extra 1s have been popped. We can finally accept here. $Q_3$ can't be an accept state because we can get there by having a bunch of 0s at the bottom of the stack with one 1 at the top.
4) Add a new start symbol

\[ S \rightarrow A \]

\[ A \rightarrow BAB \mid B \mid \epsilon \]

\[ B \rightarrow 00 \mid \epsilon \]

- Remove \( \epsilon \) rules (except \( S \rightarrow \epsilon \))

\[ S \rightarrow A \mid \epsilon \]

\[ A \rightarrow BAB \mid BA \mid AB \mid B \mid BB \]

\[ B \rightarrow 00 \]

- Remove unit rules

\[ S \rightarrow BAB \mid BA \mid AB \mid BB \mid \epsilon \]

\[ A \rightarrow BAB \mid BA \mid AB \mid BB \mid 00 \]

\[ B \rightarrow \epsilon \]

- Convert long rules, terminals

\[
\begin{align*}
S & \rightarrow \epsilon \mid BD \mid BA \mid AB \mid BB \mid CC \\
A & \rightarrow BD \mid BA \mid AB \mid BB \mid CC \\
B & \rightarrow CC \\
C & \rightarrow \epsilon \\
D & \rightarrow AB
\end{align*}
\]
S) Let $G_1$ be a context-free grammar $(V_1, \Sigma, R_1, S_1)$, and $G_2$ be a context-free grammar $(V_2, \Sigma, R_2, S_2)$. We will prove that $L(G_1) \cup L(G_2)$, $L(G_1) \cdot L(G_2)$, and $(L(G_1))^*$ are all languages that are context-free.

(a) We will create a CFG $G$ such that $L(G) = L(G_1) \cup L(G_2)$. Let $G = (V, \Sigma, R, S)$ with

\[ V = V_1 \cup V_2 \cup \Sigma \cup S_1 \cup S_2 \]

\[ R = R_1 \cup R_2 \cup S \rightarrow S_1 | S_2 \]

Now, we show $L(G) = L(G_1) \cup L(G_2)$. Consider any string $w \in L(G)$. The first derivation must be either $S \rightarrow S_1$ or $S \rightarrow S_2$. If $w$ is produced on the first "path" then $w \in L(G_1)$, otherwise is $w$ is produced on the second path then $w \in L(G_2)$. By defn of union it follows that $w \in L(G_1) \cup L(G_2)$. Now consider any string $w \in L(G_1) \cup L(G_2)$. Either $w \in L(G_1)$ or $w \in L(G_2)$. In the first case, we prove $w \in L(G)$ by taking the derivation $S \rightarrow S_1$ and replicating the steps of deriving $w$ in $G_1$. In the second case we also show $w \in L(G)$ by taking the derivation $S \rightarrow S_2$ and replicating the steps of deriving $w$ in $G_2$. Thus we've shown $w \in L(G)$. It follows that $L(G) = L(G_1) \cup L(G_2)$, proving that context free languages are closed under the union operation.
(b) We will create a CFG $G$ such that $L(G) = L(G_1) \cdot L(G_2)$.

Let $G = (V, \Sigma, R, S)$ with

$V = V_1 \cup V_2 \cup \Sigma \cup S \cup \Sigma S \cup S \Sigma$

$R = R_1 \cup R_2 \cup S \Rightarrow S_1 S_2$

Now we show $L(G) = L(G_1) \cdot L(G_2)$. Consider any string $w \in L(G)$. Since the first production must be $S \Rightarrow S_1 S_2$, there exist strings $u, v$ such that $u$ is derived by $S$, and $v$ in derived by $S_2$ with $w = uv$. It follows that $w \in L(G_1)$ and $v \in L(G_2)$. Thus, $w \in L(G_1) \cdot L(G_2)$. Now consider any string $w \in L(G_1) \cdot L(G_2)$. There exist $u, v$ with $u \in L(G_1)$ and $v \in L(G_2)$ with $w = uv$. In $G$, do the production $S \Rightarrow S_1 S_2$ and then replicate the productions for $S_1$ to produce $u$ and in $S_2$ to produce $v$. In this manner $G$ produces $uv$. Thus $uv = \in L(G)$, as desired. It follows that $L(G) = L(G_1) L(G_2)$ and context-free languages are closed under concatenation.

(c) We will create a CFG $G$ such that $L(G) = (L(G_1))^*.$

Let $G = (V, \Sigma, R, S)$ with

$V = V_1 \cup \Sigma \cup S \cup S \Sigma$

$R = R_1 \cup R_2 \cup S \Rightarrow \Sigma \cup S \Rightarrow S_1 S_2$

Now we will show $L(G) = (L(G_1))^*.$

Consider any string $w \in L(G)$, $w$ must have been produced by some number of applications of $S \Rightarrow S_1 S$ and one application of $S \Rightarrow \Sigma$. Thus $w$ must be produced by some number of applications of $S_1$. By definition of $*$ and $S_1$, it follows that $w \in (L(G_1))^*$. Now, consider any string $w \in (L(G_1))^*$. There exist $u_1, u_2, \ldots, u_k$ such that $w = u_1 u_2 \ldots u_k$ and $u_i \in L(G_1)$ for $1 \leq i \leq k$. We produce $w$ in $G$ by repeating the production $S \Rightarrow S_1 S$ $k$ times and using $S \Rightarrow \Sigma$. Finally, we use the $S_1$ variable $S_1$ from the left to produce $u_i$, proving $w \in L(G)$. It follows that $L(G) = (L(G_1))^*$ and context-free languages are closed under $*$. 
6) \( G = (V, \Sigma, R, S) \) with

\[
V = \{S, T, U, V, W\} \\
\Sigma = \{a, b, c\} \\
\text{Start symbol} = S
\]

\[
R:
S \rightarrow aT \\
T \rightarrow aT | U \\
U \rightarrow VW \\
V \rightarrow aVb | \varepsilon \\
W \rightarrow cW | \varepsilon
\]

The first production guarantees that there is at least one a unmatched by a b. T allows us to have any number more a’s without a matched b. U forces us to have a string with two separate components, the first component, V, which allows an addition of matched a’s and b’s in the appropriate place in the string while the second component, W, allows us to add any number of c’s. Intuitively, to derive the string \( a^i b^j c^k \), we use the production for S, followed by the production for T \( i-j-1 \) times, followed by using the production for U once, followed by using the production for V \( j \) times, and then the production for W \( k \) times.

This grammar is unambiguous due to its directed nature. Productions such as S and U must occur exactly once, while all productions of T must occur in between the productions of S and U. This forces a structure for the parse tree of the derivation, since there is only one rule for T that doesn’t lead to U. Similarly, due to the fact that both V and W have only one rule that doesn’t “terminate” the derivation, both of these create unambiguous portions of the total parse tree, making the entire parse tree for any given string, unambiguous.

7) Our PDA will have four states: \( q_{\text{start}} \), \( q_a \), \( q_b \), and \( q_c \) with the start state \( q_{\text{start}} \), the alphabet \{a,b,c\}, the stack alphabet of \{\$,a\}, the accept state \( q_c \), and the following transition function:

\[
\delta(q_{\text{start}}, a, \varepsilon) = \{(q_a, $)\}, \text{push a \$ on the stack after reading the first a.} \\
\delta(q_a, a, \varepsilon) = \{(q_a, a)\}, \text{add an a to the stack for each subsequent a.} \\
\delta(q_a, \varepsilon, \varepsilon) = \{(q_b, \varepsilon)\}, \text{non-deterministically move to } q_b \text{ to get ready to read in b’s.} \\
\delta(q_b, b, a) = \{(q_b, \varepsilon)\}, \text{must match each b with an “extra” a on the stack.} \\
\delta(q_b, \varepsilon, \varepsilon) = \{(q_c, \varepsilon)\}, \text{non-deterministically move to } q_c \text{ to get ready to read in c’s.} \\
\delta(q_c, c, \varepsilon) = \{(q_c, \varepsilon)\}, \text{only stay in } q_c \text{ with any additional c’s.}
\]