Find a grammar that generates the language

$$L = \{\ a^n b^n c^n d^n \mid n \in \mathbb{N}\ \}$$

Also explain how your grammar works.

Answer: This grammar works by first generating the correct number of non-terminals corresponding to the desired symbols, then putting the non-terminals in the correct order, and finally sweeping through the string, replacing non-terminals with the corresponding terminal.

$S \longrightarrow SABCD$	generate enough of each symbol, keeping the numbers equal
$S \longrightarrow W$	set up the sweeper
$BA \longrightarrow AB$	put the non-terminals in order
$CA \longrightarrow AC$	
$CB \longrightarrow BC$	
$DA \longrightarrow AD$	
$DB \longrightarrow BD$	
$DC \longrightarrow CD$	
$WA \longrightarrow aW$	sweep across, replacing non-terminals with terminals
$W \longrightarrow X$	switch to the next stage of the sweeper
$XB \longrightarrow bX$	
$X \longrightarrow Y$	
$YC \longrightarrow cY$	
$Y \longrightarrow Z$	
$ZD \longrightarrow dZ$	
$Z \longrightarrow \epsilon$	eliminate the sweeper when done

Discussion: A good starting point can be a grammar for a similar language — understand how it works, then adapt it for the particular desired language. Recall the grammar for $a^n b^n c^n$ that was discussed in class:

$$
\begin{aligned}
S &\longrightarrow SABC \\
S &\longrightarrow X \\
BA &\longrightarrow AB \\
CA &\longrightarrow AC \\
CB &\longrightarrow BC \\
XA &\longrightarrow aX \\
X &\longrightarrow Y \\
YB &\longrightarrow bY \\
Y &\longrightarrow Z \\
ZC &\longrightarrow cZ \\
Z &\longrightarrow \varepsilon
\end{aligned}
$$

See the posted derivations examples for a more complete discussion of how this grammar works, but the idea was that there were three stages — first the $S \longrightarrow SABC$ rule is used to generate enough As, Bs, and Cs (and because each application of the rule generates one of each, there will be the same number of each in the end). Then, the rules of the form $BA \longrightarrow AB$ are used to get the non-terminals in the right order. Finally, the rules of the form $XA \longrightarrow aX$ in conjunction with rules of the form $S \longrightarrow X$ sweep through the string, replacing non-terminals with terminals.

So, for $a^n b^n c^n d^n$:

$S \longrightarrow SABCD$	generate enough of each symbol, keeping the numbers equal
$S \longrightarrow W$	set up the sweeper
$BA \longrightarrow AB$	put the non-terminals in order
$CA \longrightarrow AC$	
$CB \longrightarrow BC$	
$DA \longrightarrow AD$	
$DB \longrightarrow BD$	
$DC \longrightarrow CD$	
$WA \longrightarrow aW$	sweep across, replacing non-terminals with terminals
$W \longrightarrow X$	switch to the next stage of the sweeper
$XB \longrightarrow bX$	
$X \longrightarrow Y$	
$YC \longrightarrow cY$	
$Y \longrightarrow Z$	
$ZD \longrightarrow dZ$	
$Z \longrightarrow \epsilon$	eliminate the sweeper when done

Find a grammar that generates the language

$$L = \{\ a^{2^n}\ \mid n \in \mathbb{N}\ \}$$

Also explain how your grammar works.

Answer: The idea is to repeatedly double the number of symbols. We'll use B to keep track of how many times to do the doubling, and A to keep track of the number of as to generate.

$S \longrightarrow DTAE$	initial setup — D is the sweeper for the last round and E marks the end
$T \longrightarrow TB$	generate n Bs
$T \longrightarrow \epsilon$	clean up the T once we have enough Bs
$BA \longrightarrow AAB$	sweep the Bs through, doubling each A
$DA \longrightarrow aD$	sweep the D through, converting the As to as
$BE \longrightarrow E$	clean up once the B has done its job
$DE \longrightarrow \epsilon$	clean up once the D has done its job

Discussion: This is similar to a^{n^2}, which was discussed in class. Recall that grammar:

$$\begin{aligned}
S &\longrightarrow DTE \\
T &\longrightarrow BTA \\
T &\longrightarrow \varepsilon \\
BA &\longrightarrow AaB \\
Ba &\longrightarrow aB \\
BE &\longrightarrow E \\
DA &\longrightarrow D \\
Da &\longrightarrow aD \\
DE &\longrightarrow \varepsilon
\end{aligned}$$

See the posted derivations examples for a more complete discussion of how this grammar works, but first recognize that $a^{n^2} = a^{nn}$ — i.e. the grammar is computing the product of $n \times n$. The idea of the grammar is to use the $T \longrightarrow BTA$ rule to produce n Bs and n As, then sweep the group of n Bs to the right, generating an a each time BA arises — the rule $BA \longrightarrow AaB$ moves the B past the A and generates the a. Each time the group of n Bs goes past one A, n as are generated, so when the n Bs have passed all n As, n^2 as have been generated. E absorbs Bs that have made it to the end ($BE \longrightarrow E$), and D eliminates As once all of the Bs have passed ($DA \longrightarrow D$).

So what about a^{2^n}? 2^n isn't the product of two numbers, it's $\overbrace{2 \times 2 \times \ldots \times 2 \times 2}^{n \text{ times}}$. But this can be written $(((1 \times 2) \times 2) \times 2) \times \ldots$ — the idea is to repeatedly double the previous value. So, if we start with one A and double it each time a B passes...

Start the grammar with rules to create one A and n Bs:

$$\begin{aligned} S &\longrightarrow DTAE \\ T &\longrightarrow TB \\ T &\longrightarrow \epsilon \end{aligned}$$

Let's try generating $a^8 = a^{2^3}$ as an example and a test of the grammar. Use the $T \longrightarrow TB$ rule to generate n Bs:

$$\begin{aligned} S &\Longrightarrow DTAE && S \longrightarrow DTAE \\ &\Longrightarrow DTBAE && T \longrightarrow TB \\ &\Longrightarrow DTBBAE && T \longrightarrow TB \\ &\Longrightarrow DTBBBAE && T \longrightarrow TB \\ &\Longrightarrow DBBBAE && T \longrightarrow \epsilon \end{aligned}$$

Now we want to move the Bs to the right, doubling the As each time. Add a rule:

$$BA \longrightarrow AAB$$

Then continue the derivation:

$$\begin{aligned} &\Longrightarrow DBBAABE && BA \longrightarrow AAB \\ &\Longrightarrow DBAABABE && BA \longrightarrow AAB \\ &\Longrightarrow DBAAAABBE && BA \longrightarrow AAB \\ &\Longrightarrow DAABAAABBE && BA \longrightarrow AAB \\ &\Longrightarrow DAAAABAABBE && BA \longrightarrow AAB \\ &\Longrightarrow DAAAAAABABBE && BA \longrightarrow AAB \\ &\Longrightarrow DAAAAAAAABBBE && BA \longrightarrow AAB \end{aligned}$$

Observe that we now have the right number of As. We're done with the Bs, so clean them up. Add a rule:

$$BE \longrightarrow E$$

Then continue the derivation:

$$\begin{aligned} &\Longrightarrow DAAAAAAAABBE && BE \longrightarrow E \\ &\Longrightarrow DAAAAAAAABE && BE \longrightarrow E \\ &\Longrightarrow DAAAAAAAAE && BE \longrightarrow E \end{aligned}$$

Next, sweep the D through to convert the As to as. (Why sweep instead of just a rule $A \longrightarrow a$? Sweeping ensures that A can't be converted to a until all of the Bs have passed (and duplicated) that A.) Add a rule:

$$DA \longrightarrow aD$$

Then continue the derivation:

$$\begin{array}{ll}
\Longrightarrow aDAAAAAAAE & DA \longrightarrow aD \\
\Longrightarrow aaDAAAAAAE & DA \longrightarrow aD \\
\Longrightarrow aaaDAAAAAE & DA \longrightarrow aD \\
\Longrightarrow aaaaDAAAAE & DA \longrightarrow aD \\
\Longrightarrow aaaaaDAAAE & DA \longrightarrow aD \\
\Longrightarrow aaaaaaDAAE & DA \longrightarrow aD \\
\Longrightarrow aaaaaaaDAE & DA \longrightarrow aD \\
\Longrightarrow aaaaaaaaDE & DA \longrightarrow aD
\end{array}$$

Finally, clean up the DE. Add a rule:

$$DE \longrightarrow \epsilon$$

Then continue the derivation:

$$\Longrightarrow aaaaaaaa \quad DE \longrightarrow \epsilon$$

Find a grammar that generates the language

$$L = \{ \ ww \mid w \in \{a, b\}^* \ \}$$

Also explain how your grammar works.

Answer: The strategy is to generate ww^R, then reverse w^R using the idea of a stack, and finally clean up the remaining non-terminals.

$S \longrightarrow RT$	T is a marker for the top of the stack used to reverse w^R
$R \longrightarrow aRa$	generate ww^R
$R \longrightarrow bRb$	
$R \longrightarrow D$	add a divider between w and w^R
$D \longrightarrow DP$	create a pusher P to move symbols to the top of the stack
$Paa \longrightarrow aPa$	push the symbol following the P to the top of the stack (the other side of the T)
$Pab \longrightarrow bPa$	
$Pba \longrightarrow aPb$	
$Pbb \longrightarrow bPb$	
$PaT \longrightarrow Ta$	
$PbT \longrightarrow Tb$	
$DT \longrightarrow \epsilon$	clean up

Discussion: There wasn't a direct example of this type of grammar discussed in class, but we can still draw inspiration from the general idea of the $a^nb^nc^n$ grammar — first generate the right symbols, then get them in the right order. We know how to generate matched sets from context-free grammars — the following generates ww^R:

$$S \longrightarrow aSa$$
$$S \longrightarrow bSb$$
$$S \longrightarrow \epsilon$$

So then the idea is to reverse the second half of the generated string. Recall from pushdown automata that a stack reverses things — constructing a pushdown automaton to accept wcw^R was discussed in class, and the idea was to push w onto the stack one symbol at a time, then pop a matching symbol for each symbol of w^R. How is this relevant to our grammar? Let's set up an intermediate step of the derivation as follows:

$$wDPw^RT$$

where D acts as a divider between w and w^R, T sits just to the left of the top of the stack, and P is a pusher that will push one symbol of w^R to the top of the stack. To do this, start with the following rules:

$$\begin{aligned} S &\longrightarrow RT \\ R &\longrightarrow aRa \\ R &\longrightarrow bRb \\ R &\longrightarrow D \end{aligned}$$

And a sample derivation to test the grammar:

$$\begin{aligned} S &\Longrightarrow RT && S \longrightarrow RT \\ &\Longrightarrow aRaT && R \longrightarrow aRa \\ &\Longrightarrow aaRaaT && R \longrightarrow aRa \\ &\Longrightarrow aabRbaaT && R \longrightarrow bRb \\ &\Longrightarrow aabbRbbaaT && R \longrightarrow bRb \\ &\Longrightarrow aabbDbbaaT && R \longrightarrow D \end{aligned}$$

Now create a pusher and have it move the first symbol of w^R to the top of the stack i.e. just past the T:

$$\begin{aligned} D &\longrightarrow DP \\ Paa &\longrightarrow aPa \\ Pab &\longrightarrow bPa \\ Pba &\longrightarrow aPb \\ PaT &\longrightarrow Ta \\ PbT &\longrightarrow Tb \end{aligned}$$

Continuing the derivation:

$$\begin{aligned} &\Longrightarrow aabbDPbbaaT && D \longrightarrow DP \\ &\Longrightarrow aabbDbPbaaT && Pbb \longrightarrow bPb \\ &\Longrightarrow aabbDbaPbaT && Pba \longrightarrow aPb \\ &\Longrightarrow aabbDbaaPbT && Pba \longrightarrow aPb \\ &\Longrightarrow aabbDbaaTb && PbT \longrightarrow Tb \end{aligned}$$

Generate a new pusher and repeat:

$$\begin{aligned} &\Longrightarrow aabbDPbaaTb && D \longrightarrow DP \\ &\Longrightarrow aabbDaPbaTb && Pba \longrightarrow aPb \\ &\Longrightarrow aabbDaaPbTb && Pba \longrightarrow aPb \\ &\Longrightarrow aabbDaaTbb && PbT \longrightarrow Tb \end{aligned}$$

And again:

$$\begin{aligned} &\Longrightarrow aabbDPaaTbb && D \longrightarrow DP \\ &\Longrightarrow aabbDaPaTbb && Pba \longrightarrow aPb \\ &\Longrightarrow aabbDaTabb && PaT \longrightarrow Ta \end{aligned}$$

Once more:

$$\begin{aligned} &\Longrightarrow aabbDPaTabb && D \longrightarrow DP \\ &\Longrightarrow aabbDTaabb && PaT \longrightarrow Ta \end{aligned}$$

Now we have $wDTw$, so all that remains is to clean up the DT. Add a rule:

$$DT \longrightarrow \epsilon$$

And finish the derivation:

$$\Longrightarrow aabbaabb \quad DT \longrightarrow \epsilon$$