Here is the `double` program in its entirety:

```prolog
double([], []).
double([I|L], [[I, I]|R]) :-
    double(L, R).
```

The answer I had in mind is

```prolog
?- append(Y, Z, L), Y \= Z.
```

I didn’t take any points off for not remembering how to test for inequality in Prolog; it wasn’t a memory test, except for big things like “and” being expressed with a comma.

The following answer also works:

```prolog
?- append(Y, Z, L), \+ append(Y, Y, L).
```

It took me a minute to realize how and why it works. You got 10 points for this; it’s a bit indirect, but less efficient only by a constant factor, since \(Y\) is known by the time we get to the second `append` goal, so testing whether it can be achieved requires no backtracking.

The language accepted by the mystery DCG:

```prolog
mystery --> [\_].
mystery --> [A, A].
mystery --> [A], mystery, [A].
```

is the set of nonempty palindromes, lists of tokens that are the same when reversed.

Half a point was taken off for omitting the word “nonempty.” To include the empty case, replace the second rule by

```prolog
mystery --> [].
```
(If you just omit the second rule, you get the set of odd-length palindromes.)

A common error was to suppose that the A in the first use of a rule from the grammar was the same in subsequent rules. But every use of a Prolog rule binds fresh variables, just as for any other programming language. Remembering variable values across rule applications is one use of the parameters so conspicuously (but not as conspicuously as I thought) absent in the mystery case, and in every context-free language. Subject-verb agreement is an instance where parameters are crucial in real languages.

Here is how a parameter would make the language more like what several midterm takers hoped:

\[
\begin{align*}
mystery(.) & \rightarrow [\_]. \\
mystery(A) & \rightarrow [A,A]. \\
mystery(A) & \rightarrow [A], \mystery(A), [A].
\end{align*}
\]

Letting center dot ("·") stand for an arbitrary character, and superscript "?" mean what precedes it is optional, the DCG defines the language \(\cdot \cup c^n \cdot ? c^n, \ n \geq 1\), for any token \(c\). (The awkwardness at the beginning has to be tacked on to pick up the case where the string of tokens is of length 1 and \(c\) doesn’t matter.) For this version, the query \(\mystery(C, [a,a,d,a,a,b], .)\) succeeds 3 times, accepting prefixes \([a]\), \([a,a]\), and \([a,a,d,a,a]\), and binding \(C\) to \(a\) each time. (The first character seen must be the one repeated for all cases of length > 1.)

Describing a language close to the last version above earned about 5 or 6 points, depending on how the [\_] was handled.