PROGRAM DEVELOPMENT STAGES

1) Initially, the "program" = only DB & KB & translator, all in the same file. Test Robbie interactively with individual queries at the prompt – and Robbie's answers go to the screen.

2) Separate the 3 files, and use a "loader" file (which I call the controller) to save time consulting all the files.

3) Once Robbie's giving correct answers . . . put the queries into the query file. (Add this file to the CONTROLLER file's consult's). And each query will have a "name", in effect. For example:

```
q(82) :-
        color(X,red),
        shape(X,aCube).
```

4) But this changes how you get multiple answers printed out on a "fill in the blank" type of question. Since there's no human controller doing the "`;'s`", you'll need a looping mechanism. We'll do this with the "natural" (to Prolog) backtrack/fail looping technique. [More on this in class].

5) Notice that you get duplicate answers (e.g., theRock is listed several times as being somewhere left of theBox). So this will need to be taken care of so you don't duplicate answers. "setof" fixes this (or findall – we'll talk about these, and there's a slight difference between the two). But remember to take out the backtrack/fail loop since setof itself does the looping. However, only put in a setof to replace the backtrack/fail loop for those queries where either:

1) you get duplicates in the answer
OR 2) you get no answer at all for the question.

(NOTE: setof and findall behave differently). The rest of the answer-type questions MUST use BACKTRACK/FAIL LOOPS (because I said so and it's good practice for understanding this type of looping, not because Prolog needs it. Do NOT just use setof's for all the questions).

6) By now you're tired of typing in q(1), q(2), ... q(30,X), … So, you embed these "calls" in the program in a temporary runQueries file (and add that to the controller's consultor) so that you can run the set of all queries by just typing doAllQueries.

7) But now you can't rely on Prolog to write out each of the answers – so you have to do it "manually". So, for T/F questions, put `write(true),nl` and `write(false),nl` in the correct places in each query. Similarly, your code for the fillInTheBlank questions has to write out the correct answer(s) (the names of things) – using write statement(s) and nl's.

8) And there's a problem if any rule fails (like a T/F question would return a "no") – so you have to force each query to SUCCEED, even if the answer to the question was false. You get a similar problem with fillInTheBlank queries which return the "no blocks fit this description" case (depending on whether you used setof or findall for this special case).

9) You'll want to capture your answers in a file, answers, rather than having them sent to the screen. So add that code to the controller file. (No, answers file doesn't need consulting). Use `tell` to open the named answers file – so all write's and nl's are routed to that file rather than the screen

10) But now all you get is a file full of just answers. So, each time a question is asked, you want the question itself (the actual English language question) to appear in the answer file before you send the actual answer to the file. My runQuestions file will take care of this. And get rid of your runQueries file which does the same thing (a simplified version). [Change what's in the controller file].

11) And rather than YOU (as human user) starting the running of the queries, put the necessary code in the controller file to start this. And to keep things tidy, add a `told` to the controller to close the answers file (when you're entirely done with everything). And you might as well make the whole thing self-starting and self-stopping by putting the necessary code for that in the controller file.

NOTE: You need not go through each of these stages completely. This is just meant to help you think through incremental development. Once you get the idea of how something works, proceed to the next stage. Or add several stages at once as you proceed.