

# Constructing a prolog program

- We will learn predicates for:
  - input and output.
  - Modifying the knowledge base.
  - Controlling backtracking process.

# Output: write, nl, display

- write: takes any prolog term and displays that term on screen.
- nl: a new line.

```
?- write('Hello'), nl, write('Goodbye').
```

```
Hello
```

```
Goodbye
```

```
true.
```

In lower case, just like in the knowledge base

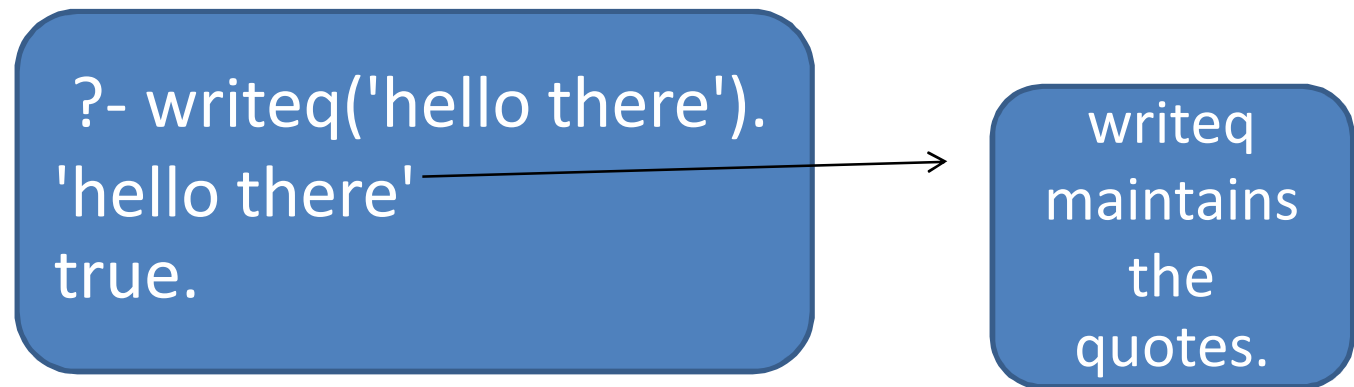
?- mother(X,cathy), write('The mother of Cathy is '), write(X).

The mother of Cathy is melody

X = melody ;

false.

The single quote is lost.



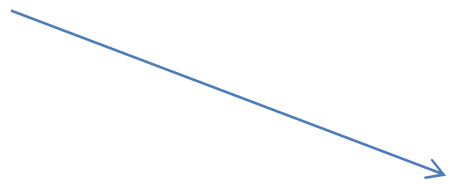
This is useful if we want to write to a file and read back, because an atom will not be splited into two atoms.

For `write(X)`, if `X` is not instantiated, prolog will print a variable with no name: such as `_G232`

?- `write(3.14*2).`

`3.14*2`

`true.`



No math  
calculation

- display

- Put all functions in front of their arguments. For example:

- ?- display(2+2).

- +(2, 2)

- true.

- ?- display('don't panic').

- don't panic

- true.

- `write_canonical`
  - Combines the effect of `writeln` and `display`:

```
?- write_canonical(2+3).
```

```
+(2, 3)
```

```
true.
```

```
?- write_canonical('hello there').
```

```
'hello there'
```

```
true.
```

# List all solutions with fail & write

- Let us have

`capital_of(georgia,atlanta).`

`capital_of(california,sacramento).`

`capital_of(florida,tallahassee).`

`capital_of(maine,augusta).`

- Let's ask:



The writes print what they have. They do not care whether their values make the query succeed in the end.

?- capital\_of(State, City), write(City), write(' is the capital of '), write(State), nl, fail.

atlanta is the capital of georgia

sacramento is the capital of california

tallahassee is the capital of florida

augusta is the capital of maine

false.

- write, writeq, nl, display do not produce alternative solutions.
- Therefore, when fail is executed, prolog has to go back to do its unification at capital\_of(State, City).

# Predicates as subroutines

?- capital\_of(State, City), write(City), write(' is the capital of '), write(State), nl, fail.



We can put this query as one of our rules in the file.

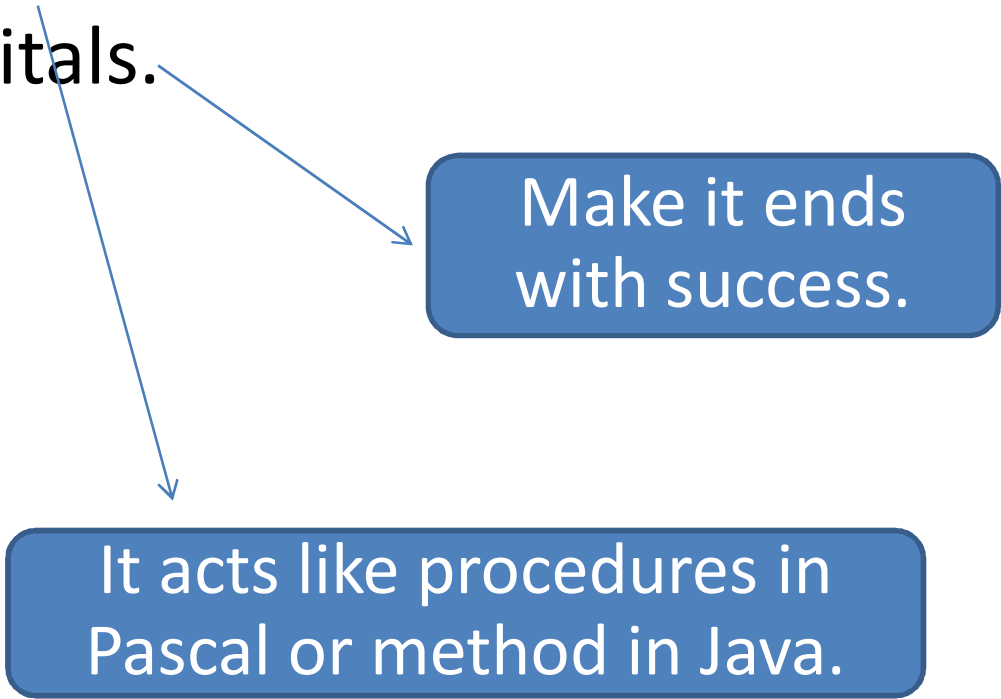
```
print_capitals :- capital_of(State, City),  
write(City), write(' is the capital of '),  
write(State), nl, fail.
```

We can even make the program structure clearer. ->

```
print_a_capital :- capital_of(State, City),  
    write(City), write(' is the capital of '),  
    write(State), nl.
```

```
print_capitals :- print_a_capital, fail.
```

```
print_capitals.
```



Make it ends  
with success.

It acts like procedures in  
Pascal or method in Java.

# read

- Accepts any prolog term from the keyboard.
- The term must be like in a prolog program.
- Must end with a period. (otherwise prolog will wait for it forever)

```
?- read(X).
```

```
| hello.
```

```
X = hello.
```

```
?- read(X).
```

```
| hello there.
```

```
ERROR: Stream user_input:5:61 Syntax error: Operator  
expected
```



2 terms! Prolog does not accept it.

- If the argument of read is already instantiated, then read will try to unify that argument with what the user types.

```
?- read(hello).  
| hello.  
true.
```

```
?- read(hello).  
| goodbye.  
false.
```

```
?- read(X).  
| mother(melody,cathy).  
X = mother(melody, cathy).
```

Any legal prolog term will work.

```
?- read(X).  
| f(Y) :- g(Y).  
X = (f(_G307):-g(_G307)).
```

Uninstantiated  
variable.

- read does not produce alternative solutions upon backtracking.

- Now we can create a program that interacts with users better. See [interac.pl](#)

```
capital_of(georgia,atlanta).
```

```
capital_of(florida,tallahassee).
```

```
go :- write('What state do you want to know about?'),nl,  
      write('Type its name, all lower case, followed by a period. '),nl,  
      read(State),  
      capital_of(State, City),  
      write('Its capital is: '),write(City),nl.
```



?- go.

What state do you want to know about?

Type its name, all lower case, followed by a period.

|: florida.

Its capital is: tallahassee


true.

# Manipulating the knowledge base

- A prolog program can modify itself.
  - asserta : adds clause at the beginning of the clauses of the same name
  - assertz : adds clause at the end of the clauses of the same name
  - retract : removes a clause
- The commands change the file in memory. The disk file is unaffected.
- Before any of these can work, you must indicate in your program that you want a predicate to be modifiable.

- You let your interpreter know by adding dynamic rule in your program. For example:

Number of arguments

 :- dynamic capital\_of/2.  
capital\_of(georgia,atlanta).  
capital\_of(florida,tallahassee).

- Then you can add/remove things

?- asserta(capital\_of(hawaii,honolulu)).



Adds this new fact before the other clauses for capital\_of.

- **Retract**

- Its argument is either

- A complete clause, or
    - A structure that matches the clause but contains some uninstantiated variables.

?- retract(mother(melody,cathy)).

Removes mother(melody,cathy) from the knowledge base.

?- retract(mother(X,Y)).

Finds the first clause that matches mother(X,Y) and removes it. X and Y are instantiated to the arguments found in that clause. If there is no matching clause, retract fails.

?- retract(capital\_of(X,Y)).

X = hawaii,

Y = honolulu ; 

X = georgia,

Y = atlanta .

It finds the first clause of capital\_of and removes it. Then our request for alternative solution makes prolog removes the first clause at that moment.

Overall, 2 clauses are removed. The result of assert and retract cannot be undone upon backtracking.

- Extra parentheses are required when the argument contains comma or an if operator:
- See family.pl that I inserted male(...).

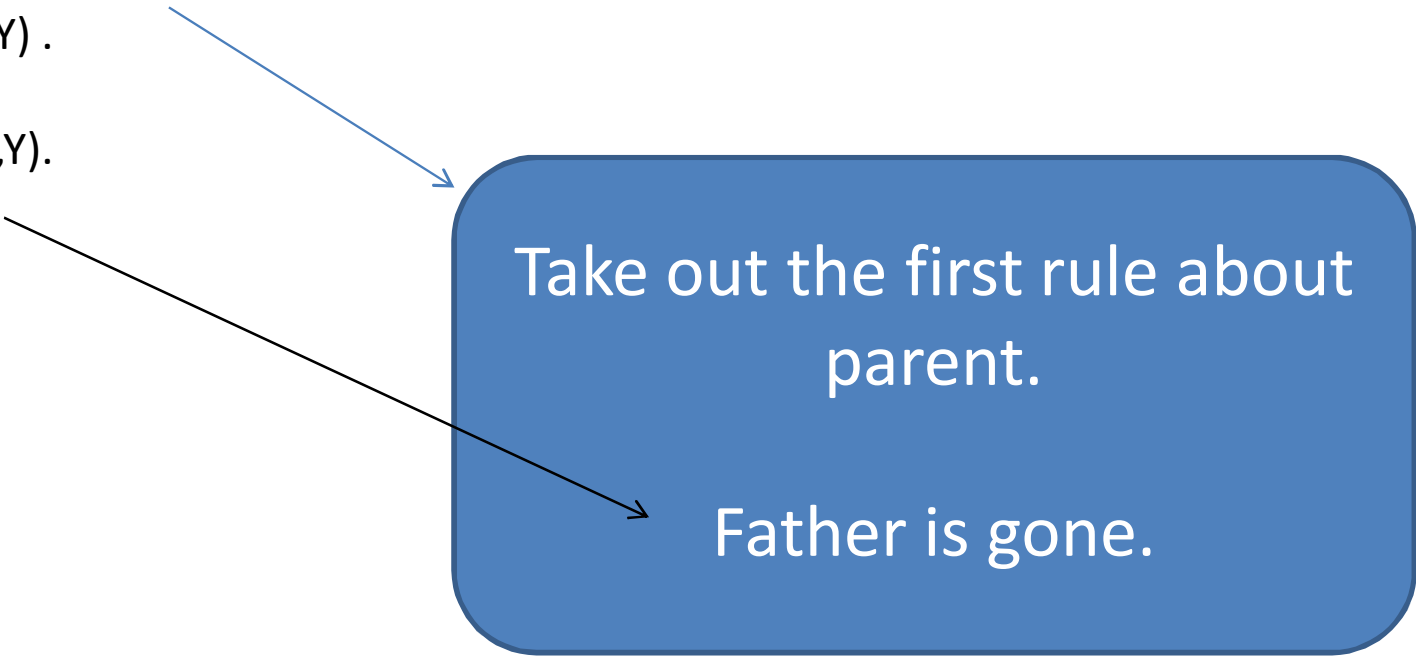
?- asserta((father(X) :- male(X))).  
true.

```
36 ?- father(F).  
F = michael ;  
F = charles_gordon ;  
F = charles ;  
F = jim ;  
F = elmo ;  
F = greg ;  
F = tim.
```

:- dynamic male/1.  
:- dynamic father/1.  
should be in the  
prolog code.

```
retract((parent(X,Y) :- Z)).  
Z = father(X, Y) .
```

```
9 ?- parent(X,Y).  
X = melody,  
Y = cathy ;  
X = melody,  
Y = sharon ;  
X = hazel,  
Y = michael ;  
X = hazel,  
Y = julie ;  
X = eleanor,  
Y = melody ;  
X = eleanor,  
Y = crystal ;  
X = crystal,  
Y = stephanie ;  
X = crystal,  
Y = danielle ;  
X = julie,  
Y = tim.
```



Take out the first rule about  
parent.

Father is gone.



- abolish

- Removes all clauses of the specified name and arity.

```
?- abolish(father/2).
```

```
true.
```

```
3 ?- father(X,cathy).
```

```
ERROR: Undefined procedure: father/2
```

```
ERROR: However, there are definitions for:
```

```
ERROR: father/1
```

```
false.
```

If you want to add things back again, you may have to exit and re-launch prolog.

- listing
  - List everything in the knowledge base at that moment.
  - To see only a particular predicate, see an example below :

?- listing(mother/2).  
mother(melody, cathy).  
mother(melody, sharon).  
mother(hazel, michael).  
mother(hazel, julie).  
mother(eleanor, melody).  
mother(eleanor, crystal).  
mother(crystal, stephanie).  
mother(crystal, danielle).  
mother(julie, tim).

true.

# Static and dynamic predicates

- Static = cannot assert, retract, etc.
- Dynamic = can assert, retract, etc.
- Normally, all are static unless you make them dynamic.
- In some implementations, all are dynamic. In others, they are dynamic if you load them with **consult** or **reconsult**, and static if you load them with **compile**.

- Any predicate that is created with assert is automatically dynamic.
- Querying a non existing predicate normally causes prolog to raise an error condition.
  - But if we tell prolog that the predicate is dynamic:
    - The query will fail without raising an error condition.

```
test :- f, write('not the first time').
```



```
?- test.  
ERROR: test/0: Undefined procedure: f/0
```

```
:- dynamic(f/0).  
test :- f, write('not the first time').
```



```
?- test.  
false.
```

- Abolish clears a predicate and its dynamic declaration.
- If we want the dynamic declaration to remain, we must write a removal rule ourselves:

```
clear_away :- retract(f(_, _)), fail.
```

```
clear_away :- retract(f(_, _) :- _), fail.
```

```
clear_away.
```

Try this with `located_in` from `geo.pl`

# More about consult and reconsult

- Consult = read/1 from a file and put each term that it reads into the knowledge base.
- Reconsult = the same as consult, but it throws any existing definition of a predicate away.

SWI-> consult = reconsult

- Any term consult reads, that starts with :- is regarded as a query.

- Embedded queries can be useful:

```
:- write('Type "go." to start.').
```

Put it at the end of your code. When consult is performed, the program will type this instruction.

```
:- reconsult('capitals.pl').
```

Include capitals.pl with the current file. All facts and rules from capitals.pl are loaded too. (the file must be in the same folder)

```
:- multifile(capital_of/2).
```

Some prolog does not support it.

Allow reconsult to take the definition of capital\_of from more than one file, instead of dumping a set of definition if clauses of the same name are found in another file. This declaration must be in every file that has the clause capital\_of.

# File handling: see, seen, tell, told

?-see('mydata'),

read(X), read(Y), read(Z),

seen.

Tell prolog to take inputs from this file instead of from the keyboard.

Close all input files and switch input back to the keyboard.

As long as the file is open, prolog knows the position of the next term to be read.



- Calling several `see` allows you to switch between several input files that are open at once.

```
?- see('aaa'),  
   read(X1),  
   see('bbb'),  
   read(X2),  
   see(user),  
   read(X3),  
   seen.
```

The keyboard (this way we can read from the keyboard without having to close other files.)

Close all files

- If we read past the end of a file, prolog normally returns the special atom `end_of_file`

Use this file as an output instead of console.

```
?- tell('aaa'),  
   write('First line of AAA'), nl,  
   tell('bbb'),  
   write('First line of BBB'), nl,  
   tell(user),  
   write('This goes on the screen'), nl,  
   tell('aaa'),  
   write('Second line of AAA'), nl,  
   told.
```

Close all output files and switch output back to the console.

`:- dynamic(f/2).`

`test :- f, write('not the first time').`

`test :- \+ f, asserta(f), write('the first time').`

A program that  
writes itself.  
See [test.pl](#)

`?- test.`  
the first time  
true.

`8 ?- test.`  
not the first time  
true ;  
false.

# Program that remembers to the next session

- Adding information – use assert
- Remembering – redirect output to a file and do a listing
- See [LEARNER.PL](#)
  - Given a state, it attempts to name its capital.
  - If unable to name the capital, it asks user to name the capital, and stores the information in its knowledge base.
  - The knowledge base is stored separately on [KB.PL](#)

# Running learner.pl

?- start.

% kb.pl compiled 0.00 sec, 880 bytes

Type names entirely in lower case, followed by period.

Type "stop." to quit.

State? georgia.

The capital of georgia is atlanta

State? hawaii.

I do not know the capital of that state.

Please tell me.

Capital? honolulu.

Thank you.

State? maine.

The capital of maine is augusta

State? hawaii.

The capital of hawaii is honolulu

State? stop.

Saving the knowledge base...

Done.

true



It learns!

# Kb.pl after that run

`:- dynamic(capital_of/2).`

`:- dynamic capital_of/2.`

Extra?

`capital_of(georgia, atlanta).`

`capital_of(california, sacramento).`

`capital_of(florida, tallahassee).`

`capital_of(maine, augusta).`

`capital_of(hawaii, honolulu).`

Did not have it at first.

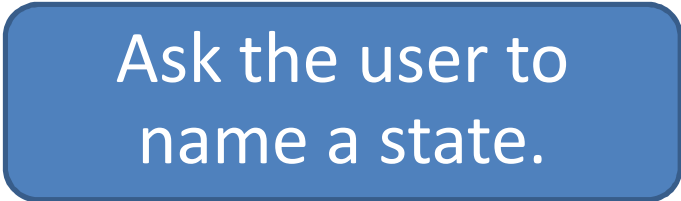
# Inside learner.pl

```
start :- reconsult('kb.pl'),  
        nl,  
        write('Type names entirely in lower case, followed by period.'),  
        nl,  
        write('Type "stop." to quit.'), nl,  
        nl,  
        process_a_query.  
  
process_a_query :- write('State? '),  
                  read(State),  
                  answer(State).
```


Load the knowledge base



Ask the user to  
name a state.



```
answer(stop) :- write('Saving the knowledge base...'),nl,  
                tell('kb.pl'),  
                write(':- dynamic(capital_of/2).'),nl, % omit if not needed  
                listing(capital_of),  
                told,  
                write('Done.').
```



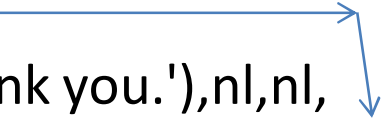
```
% If the state is in the knowledge base, display it, then  
% loop back to process_a_query
```

```
answer(State) :- capital_of(State,City),  
                write('The capital of '),  
                write(State),  
                write(' is '),  
                write(City),nl,  
                nl,  
                process_a_query.
```



```
% If the state is not in the knowledge base, ask the  
% user for information, add it to the knowledge base, and  
% loop back to process_a_query
```

```
answer(State) :- \+ capital_of(State,_),  
    write('I do not know the capital of that state. '),nl,  
    write('Please tell me. '),nl,  
    write('Capital? '),  
    read(City),  
    write('Thank you. '),nl,nl,  
    assertz(capital_of(State,City)),  
    process_a_query.
```



# Character input/output: get, get0, put

- put: output 1 character, its argument is an integer that gives the character's ASCII code.

?- put(42).

\*

true.



Ascii for asterisk.

7 -> beep

8 -> backspace

12 -> start new page on printer.

13 -> return without new line.

- `get`: accepts one character and instantiates the character to its ASCII code.

```
?- get(X).
```

```
| *
```

```
X = 42.
```

Get -> skips blanks, returns, or other nonprinting character.

```
?- get0(X).
```

```
|
```

```
X = 10.
```

Press Return

If you want to read these characters, use `get0`.

Reading past end of file returns -1

# Constructing menu

- See [menudemo.pl](http://menudemo.pl)

?- start.

Which state do you want to know about?

- 1 Georgia
- 2 California
- 3 Florida
- 4 Maine

Type a number, 1 to 4 -- 3

The capital of florida is tallahassee  
true.

```
start :- display_menu,  
         get_from_menu(State),  
         capital_of(State, City),  
         nl,  
         write('The capital of '),  
         write(State),  
         write(' is '),  
         write(City),  
         nl.
```

```
display_menu :- write('Which state do you want  
to know about?'),nl,  
                write(' 1 Georgia'),nl,  
                write(' 2 California'),nl,  
                write(' 3 Florida'),nl,  
                write(' 4 Maine'),nl,  
                write('Type a number, 1 to 4 -- ').
```

```
get_from_menu(State) :- get(Code), % read a character
                        get0(_),  % consume the Return keystroke
                        interpret(Code,State).
```

```
interpret(49,georgia).  /* ASCII 49 = '1' */
interpret(50,california). /* ASCII 50 = '2' */
interpret(51,florida).  /* ASCII 51 = '3' */
interpret(52,maine).    /* ASCII 52 = '4' */
```

# Getting only yes or no

- See [getyesno.pl](#)

```
get_yes_or_no(Result) :- get(Char),          % read a character
                        get0(_),           % consume the Return after it
                        interpret(Char,Result),
                        !.                  % cut -- see text
```

```
get_yes_or_no(Result) :- nl,
                        write('Type Y or N:'),
                        get_yes_or_no(Result).
```

```
interpret(89,yes). % ASCII 89 = 'Y'
interpret(121,yes). % ASCII 121 = 'y'
interpret(78,no). % ASCII 78 = 'N'
interpret(110,no). % ASCII 110 = 'n'
```



# An expert system

- [Car.pl](#) tells user why a car won't start.
- See the conversation with it.

?- start.

This program diagnoses why a car won't start.

Answer all questions with Y for yes or N for no.

When you first started trying to start the car,  
did the starter crank the engine normally?

|: y

Does the starter crank the engine normally now?

|: n

Your attempts to start the car have run down the battery.

Recharging or jump-starting will be necessary.

But there is probably nothing wrong with the battery itself.

Look in the carburetor. Can you see or smell gasoline?

|: n

Check whether there is fuel in the tank.

If so, check for a clogged fuel line or filter

or a defective fuel pump.

true.

?- start.

This program diagnoses why a car won't start.  
Answer all questions with Y for yes or N for no.

When you first started trying to start the car,  
did the starter crank the engine normally?

| n

Check that the gearshift is set to Park or Neutral.  
Try jiggling the gearshift lever.

Check for a defective battery, voltage  
regulator, or alternator; if any of these is  
the problem, charging the battery or jump-  
starting may get the car going temporarily.  
Or the starter itself may be defective.  
true.

If the starter is  
defective,  
there is no  
point  
collecting  
other  
information.

start :-

```
write('This program diagnoses why a car won't start. '),nl,  
write('Answer all questions with Y for yes or N for no. '),nl,  
clear_stored_answers,  
try_all_possibilities.
```

```
try_all_possibilities :- % Backtrack through all possibilities...  
    defect_may_be(D),  
    explain(D),  
    fail.
```

```
try_all_possibilities.
```

```
%  
% Diagnostic knowledge base  
% (conditions under which to give each diagnosis)  
%
```

```
defect_may_be(drained_battery) :-  
    user_says(starter_was_ok,yes),  
    user_says(starter_is_ok,no).
```

```
defect_may_be(wrong_gear) :-  
    user_says(starter_was_ok,no).
```

```
defect_may_be(starting_system) :-  
    user_says(starter_was_ok,no).
```

```
defect_may_be(fuel_system) :-  
    user_says(starter_was_ok,yes),  
    user_says(fuel_is_ok,no).
```

```
defect_may_be(ignition_system) :-  
    user_says(starter_was_ok,yes),  
    user_says(fuel_is_ok,yes).
```

```
%  
% Case knowledge base  
% (information supplied by the user during the consultation)  
%
```

```
:- dynamic(stored_answer/2).
```

```
    % (Clauses get added as user answers questions.)
```

```
%  
% Procedure to get rid of the stored answers  
% without abolishing the dynamic declaration  
%
```

```
clear_stored_answers :- retract(stored_answer(_, _)), fail.  
clear_stored_answers.
```

```
%  
% Procedure to retrieve the user's answer to each  
% question when needed,  
% or ask the question if it has not already been asked  
%
```

```
user_says(Q,A) :- stored_answer(Q,A).
```

```
user_says(Q,A) :- \+ stored_answer(Q,_),  
    nl,nl,  
    ask_question(Q),  
    get_yes_or_no(Response),  
    asserta(stored_answer(Q,Response)),  
    Response = A.
```

```
%
```

```
% Texts of the questions
```

```
%
```

```
ask_question(starter_was_ok) :-
```

```
    write('When you first started trying to start the car, '),nl,
```

```
    write('did the starter crank the engine normally? '),nl.
```

```
ask_question(starter_is_ok) :-
```

```
    write('Does the starter crank the engine normally now? '),nl.
```

```
ask_question(fuel_is_ok) :-
```

```
    write('Look in the carburetor. Can you see or smell  
    gasoline?'),nl.
```



```
%  
% Explanations for the various diagnoses  
%  
  
explain(wrong_gear) :-  
    nl,  
    write('Check that the gearshift is set to Park or Neutral. '),nl,  
    write('Try jiggling the gearshift lever. '),nl.  
  
explain(starting_system) :-  
    nl,  
    write('Check for a defective battery, voltage'),nl,  
    write('regulator, or alternator; if any of these is'),nl,  
    write('the problem, charging the battery or jump-'),nl,  
    write('starting may get the car going temporarily. '),nl,  
    write('Or the starter itself may be defective. '),nl.  
  
explain(drained_battery) :-  
    nl,  
    write('Your attempts to start the car have run down the battery. '),nl,  
    write('Recharging or jump-starting will be necessary. '),nl,  
    write('But there is probably nothing wrong with the battery itself. '),nl.  
  
explain(fuel_system) :-  
    nl,  
    write('Check whether there is fuel in the tank. '),nl,  
    write('If so, check for a clogged fuel line or filter'),nl,  
    write('or a defective fuel pump. '),nl.  
  
explain(ignition_system) :-  
    nl,  
    write('Check the spark plugs, cables, distributor, '),nl,  
    write('coil, and other parts of the ignition system. '),nl,  
    write('If any of these are visibly defective or long'),nl,  
    write('overdue for replacement, replace them; if this'),nl,  
    write('does not solve the problem, consult a mechanic. '),nl.
```