SWI environment

```
Starting SWI Prolog.
--------------------

To start the prolog interpreter type 'pl'

    % pl

after the Unix prompt '%'.

Using SWI Prolog.
-----------------

As soon as you start up SWI Prolog, you will see the message:

Welcome to SWI-Prolog (Multi-threaded, Version 5.2.11)
Copyright (c) 1990-2003 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?-

Goals in Prolog are entered by the user following the '?- ' prompt.

You can access the on-line help information by typing

?- help(help).

as the introductory message above says.

To load (or "consult") a file called file.pl:
    consult('file.pl').
    consult(file).
    ['file.pl'].
    [file].

Note the dot (".") at the end.

For this to work, make sure that you have started prolog in the
```

directory where you have saved your file.

To define some predicates without having to first save them to file, consult "user":

```
    [user].
```

Then, Prolog will present you with the following prompt:

```
|:
```

And you can type in your predicates as you would in a text editor (but you cannot edit a line after you have pressed enter.) To exit this mode, press Ctrl+D. Then Prolog will parse your predicates and make them accessible.

For example, you can enter the second_cousin predicate as follows:

```
?- [user].
|: sibling(X,Y) :- parent(Z, X), parent(Z,Y).
|: 5) second_cousin(X,Y):- grandchild(X,Z), grandchild(Y,V), sibling(Z,V).
|: % user://1 compiled 0.02 sec, 672 bytes

Yes
?-
```

It's a good idea, however, to get in the habit of using files most of the time, because what you type here will be lost as soon as SWI exits, which is likely to happen if you make mistakes typing this file.

To exit SWI:

```
    halt.
or
    Ctrl-D
```

Using predicates.

Consider the builtin predicate member/2,(i.e., the predicate "member" with 2 arguments) which succeeds if the first argument is a member of a the second arument, a list. A sample session with it might look like this (note that text between /* and */ is not part of the output:

```
?- member(1, [2,5,6,7,1]).  /* User types this */

Yes                         /* Prolog answers that the predicate
                               evaluates to true */
?- member(4, [2,5,6,7,1]).  /* User types this */

No                          /* Prolog answers that the predicate
                               evaluates to false */
?- member(X, [2,5,6,7,1]).   /* User types this */

X = 2 ;                      /* User presses ';' */

X = 5 ;                      /* Another solution */

X = 6 ;

X = 7 ;

X = 1 ;

No                           /* No more solutions */
?-
```

```
Debugging in SWI Prolog.
------------------------

For full details on debugging in SWI Prolog, read sections 2.9 and
4.39 of the documentation at www.swi-prolog.org.

Here are a few specifics:

Traces: those let you see every call one by one.
    Turning trace on:
        trace.
    Turning trace off:
        notrace.

(For more on trace mode, see the example near the end of this file)

Spy point: those let you enter trace mode when a particular predicate
is called.

    Adding a spy point for predicate pred with arity n
```

```
  (i.e., pred takes n arguments):
    spy(pred/n).
 Adding a spy point for predicate pred, tracing every
 instance of the predicate, regardless of arity:
    spy(pred).
 Removing a spy point:
    nospy(pred/n).
    nospy(pred).
```

Useful commands while in trace mode:

```
    <enter>: move one step forward
    l: leap causes the program to resume (i.e., to leave trace
       mode) until the next spy point, or until the program is done.
    s: skip the current subprogram:  the current subprogram is
       executed with trace off, and the trace resumes when that
       subprogram is done. Useful to quickly go through long
       subprograms that you've already
       debugged and you don't need to go through the details.
    a: abort: exit the program and stop tracing.
```

When you use these predicates (trace, spy), prolog enters debug mode. In this mode, prolog stops at spy points and trace points and disables some optimization so that debugging information is available. When in debug mode, the prompt will change to:

```
[debug]  ?-
```

You can also enter debug mode by typing:

```
?- debug.
```

To exit debug mode, type

```
nodebug.
```

Note that the behaviour of these commands may not be intuitively obvious when you work with them, so you should first get accustomed to them with simple programs.

These are probably the commands you'll use most often in the debugger. For more commands, read the manual.

```
Dealing with misbehaving code:
    If your code gets into infinite recursion, you can hit Ctrl-C
    to stop it.  The interpreter will ask you for an action. You can
    return to the toplevel by typing "a".  "c" will continue the
    program, while "t" will enter trace mode.  "?" will give you a
    list of available options. Of note is "b" which will get you in
    a break mode, which  starts another prolog toplevel. You can leave
    the new toplevel by typing Ctrl-D.


Example
-------

To practice with the above, type in the example presented in class:

male(tom).
male(peter).
male(doug).
male(david).
female(susan).
parent(doug, susan).
parent(tom, william).
parent(doug, david).
parent(doug, tom).
grandfather(GP, GC) :- male(GP), parent(GP, X), parent(X, GC).

And save it in a file called "parents.pl". Now start up prolog and try
some things:

?- [parents].
% parents compiled 0.00 sec, 1,856 bytes

Yes
?- male(X).

X = tom ;

X = peter ;

X = doug ;

X = david ;
```

```
No
?- parent(doug, X).

X = susan ;

X = david ;

X = tom ;

No
?- grandfather(tom, X).

No
?- grandfather(doug, X).

X = william ;

No
?- grandfather(X, william).

X = doug ;

No
?- trace, grandfather(doug, X).
   Call: (8) grandfather(doug, _G158) ? creep
   Call: (9) male(doug) ? creep
   Exit: (9) male(doug) ? creep
   Call: (9) parent(doug, _L196) ? creep
   Exit: (9) parent(doug, susan) ? creep
   Call: (9) parent(susan, _G158) ? creep
   Fail: (9) parent(susan, _G158) ? creep
   Redo: (9) parent(doug, _L196) ? creep
   Exit: (9) parent(doug, david) ? creep
   Call: (9) parent(david, _G158) ? creep
   Fail: (9) parent(david, _G158) ? creep
   Redo: (9) parent(doug, _L196) ? creep
   Exit: (9) parent(doug, tom) ? creep
   Call: (9) parent(tom, _G158) ? creep
   Exit: (9) parent(tom, william) ? creep
   Exit: (8) grandfather(doug, william) ? creep

X = william ;
```

No

The output of trace consists of the following information:

```
   Call: (8) grandfather(doug, _G158) ? creep
   ^      ^  ^                          ^
   |      |  |                          Appears after you press enter.
   |      |  |                          It means that prolog will continue
   |      |  |                          execution
   |      |  The name of the current goal
   |      Recursion depth.
   Port name
```

The port tells you the state of the current goal. The port name can be one of four: Call, Redo, Exit, Fail. Call means that prolog is trying to evaluate the goal. Exit means that the goal has succeeded. Redo means that prolog is trying to find another solution. Fail means that there is no solution (or no other solution.)

Note that in the above trace, we have pressed enter at each goal to get the default "creep" action. You can press "?" instead to get a list of available options.

More

You can find another tutorial for SWI Prolog online at
 http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/1.html

You can find the documentation for SWI Prolog at
 http://www.swi-prolog.org/

And click the link on the left column that reads "Documentation" or go directly to

http://www.swi.psy.uva.nl/projects/SWI-Prolog/Manual/Contents.html