Chapter 6 - Built-in Predicates

* Entering New Clauses
  + You want to tell the system what clauses to use, as well as ask questions about them
  + Built-in predicate *consult* is meant for those situations when you want the clauses in some file to replace all existing clauses for the same predicates
* Success and Failure
  + When executing a program, a goal succeeds when it can be satisfied, and if it fails, there is no way to satisfy it
  + True- goal always succeeds
  + Fail- goal always fails
* Classifying Terms
  + var(X) succeeds if X is currently an *uninstantiated* variable
  + nonvar(X) succeeds if X is not currently an uninstantiated variable
  + atom(X) succeeds if X currently stands for a Prolog atom
  + number(X) succeeds if X currently stands for a number
  + atomic(X) succeeds if X currently stands for either a number or an atom
  + listing(A)- A is instantiated to an atom, causes all the clauses with the atom as predicate to be written out, as Prolog terms, on the current output file
  + clause(X,Y) matches X and Y with the head and body of an existing clause in the database
  + asserta(X), assertz(X) allow one to add new clauses to the database
  + retract(X) enables a program to remove clauses from the database
* Constructing and Accessing Components of Structures
  + functor(T,F,N) means “T is a structure with functor F and arity (number of arguments) N”
  + arg(N,T,A) must always be used with its first two arguments instantiated
  + Used to access a particular argument of a structure
  + X=.. L means “L is the list consisting of the functor of X followed by the arguments of X”
  + atom\_chars(A,L) deals with arbitrary atoms
  + number\_chars(A,L) deals with arbitrary numbers
* Affecting Backtracking
  + Repeat- extra way to generate multiple solutions through backtracking
* Constructing Compound Goals
  + “,” specifies a conjunction of goals
  + “;” specifies a disjunction of goals
  + call(X) assumes that X is instantiated to term that can be interpreted as a goal
  + “\+” is declared as a prefix operator
    - It is assumed that X is instantiated to a term that can be interpreted as a goal
* Equality
  + X = Y
    - Prolog attempts to make X and Y equal by matching them together
  + “==” represents a much stricter equality test than “=”. If X == Y succeeds then X = Y does as well
* Input and Output
  + get\_char(X) succeeds if X can be matched with the next character encountered on the current input stream
  + read(X) reads the next term from the current input stream and matches it with X
  + put\_char(X) writes the character X on the current output stream
  + nl writes a control sequence to the current output stream that causes a “new line”
  + write(X) writes the term X to the current output stream
  + write\_canonical(X) works like write, except that it ignores any operator declarations
  + op(X, Y, Z) declares an operator having precedence class X, position and associativity Y, and name Z
* Handling Files
  + open(X, Y, Z) opens a file whose name is X. If Y is read then the file is opened. Z is instantiated to a special term naming the stream that must be referred to when the file is accessed later
  + close(X)
  + set\_input(X)
  + set\_output(X)
  + current\_input(X)
  + current\_output(X)
* Evaluating Arithmetic Expressions
  + X is Y
  + +
  + –
  + \*
  + / - floating point division
  + // - integer division
  + Mod
* Comparing Terms
  + X = Y
  + X =:= Y
  + X =\= Y
  + X < Y
  + X > Y
  + X >= Y
  + X =< Y
  + X @< Y – succeeds when the LH term argument is less than the RH term according to a specified ordering
  + X @> Y
  + X @>= Y
  + X @=< Y
* Watching Prolog at Work
  + trace turns on exhaustive tracing
  + notrace stops exhaustive tracing
  + spy P is used when you want to pay special attention to goals involving some specific predicates
  + debugging
  + nodebug
  + nospy can specify exactly which spy points you wish to have removed