ARITHMETIC GOALS

N > M

N < M

N =< M

N >= M

M and N should be bound to numbers for these tests to succeed or fail.

X <u>is</u> 1+2 is used to assign numeric value of RHS to variable in LHS.

= is not equal to ==
or =:=

X = Y   X \= Y

test whether X and Y can be or cannot be unified.

X == Y   X \== Y

test whether X and Y are currently co-bound, i.e., have been bound to, or share same value.

X =:= Y   X =\= Y

test arithmetic equality and inequality.

MORE EQUALITIES

X =@= Y   X \=@= Y

test whether X and Y are structurally identical.

- =@= is weaker than ==
  but stronger than = .

- Examples:

a	=@= A	false
A	=@= B	true
x(A,A)	=@= x(B,C)	false
x(A,A)	=@= x(B,B)	true
x(A,B)	=@= x(C,D)	true

EQUIVALENCE CLASSES

$$X == Y$$

$$\longrightarrow \quad X =e= Y$$

$$\longrightarrow \quad X = Y$$

but not the other way ($\leftarrow$)

If two terms are currently co-bound, they are structurally identical, and therefore they unify.

PROLOG OPERATORS

:- op (P, T, O).

declare an operator symbol O with precendence P and type T.

e.g.

```
:- op(500, xfx, 'has_color').
a has_color red.
b has_color blue.
```

then:

```
?- b has_color C.
C = ~~red~~ blue
?- What has_color red.
What = a
```

OPERATOR PRECEDENCE/TYPE

Precedence P is an integer; the larger the number, the less the precedence (ability to group).

Type T is one of:

		e.g.
xfx	infix nonassociative	is
xfy	infix right-associative	, ;
yfx	infix left-associative	+ - * /
fx	prefix nonassociative	?-
fy	prefix right-associative	not
xf	postfix nonassociative	
yf	postfix left-associative	

TESTING TYPES

atom(X)	like Oz atom/ Scheme symbol, e.g. 'foo', bar
integer (X)	not complex terms, e.g. 4/2 fails.
~~[illegible]~~/float(X)	match exact type
string (X)	enclosed in "...".

PROLOG INPUT / OUTPUT

seeing(X) succeeds if X is (or can be) bound to current read port. X = user is keyboard input.

see(X) opens port for input file bound to X, and makes it current.

seen closes current port for input, and makes user current.

read(X) reads Prolog type expression from current port, storing value in X.

end_of_file is returned by read at <EOF>.

PROLOG INPUT / OUTPUT

telling(X) succeeds if X is (or can be) bound to current output port. X=user is screen.

tell(X) opens port for output file bound to X, and makes it current

told closes current output port and reverts to screen output (user)

write(E) writes Prolog expression bound to E into current output port.

nl new line (line feed).

tab(N) write N spaces to current output port.

I/O EXAMPLE

```
browse(File) :-

        seeing(Old),          % save for later
        see(File),            % open file
        repeat,
        read(Data),           % read from File
        process(Data),
        seen,                 % close File
        see(Old),             % back to previous
        !.                    % stop now.

process(end_of_file) :- !.
process(Data) :- write(Data), nl, fail.
```

PARSING NATURAL LANGUAGE

- Definite Clause Grammars (DCG) are useful for natural language parsing.

- Prolog can load DCG rules and convert them automatically to Prolog parsing rules

DCG SYNTAX

- `-->` DCG operator.

e.g.

```
sentence --> subject, verb, object.
```

Each goal is assumed to refer to the head of a DCG rule.

- `{prolog_code}` Include Prolog code in generated parser.

e.g.

```
subject --> modifier, noun, {write('subject')}
```

- `[terminal_symbol]` Terminal symbols of the grammar.

e.g.

```
noun --> [cat]
```