

## If-then-else

... Other Useful Prolog Built-Ins ...

*If P then Q, else R* can be written as follows:

```
S :- P -> Q ; R.
```

Here's an example:

```
max(X,Y,Z) :-  
  ( X =< Y  
    -> Z=Y  
    ; Z=X  
  ).
```

Interestingly, one common use of the cut predicate is to mimic the “if-then-else” construct found in imperative languages. Here's how we can define it:

```
S :- P, !, Q.  
S :- R.
```

## If-then-else (cont)

Another example:

Write a predicate to add an element to a list with the restriction that no duplicates are added to the list. Define the predicate `add(X,L1,L2)` to mean "the result of adding X to L1 is L2."

Here's how to do it with cut:

```
add(X,L1,L2) :- member(X,L1), !, L2 = L1.  
add(X,L1,L2) :- L2 = [X|L1].
```

Here's how to do it using if-then-else:

```
add(X,L1,L2) :- member(X,L1) -> L2 = L1  
                ; L2 = [X|L1].
```

## univ

The standard built-in predicate called 'univ' (`=..`) translates a predicate and its arguments into a list whose first element is the predicate name and whose remaining elements are the arguments. It works in reverse as well.

For example,

```
?- pred(arg1,arg2) =.. X.  
X = [pred, arg1, arg2]
```

```
?- pred =.. X.  
X = [pred]
```

```
?- X =.. [pred,arg1,arg1].  
X = pred(arg1, arg2)
```

```
?- X =.. [pred].  
X = pred
```

## Example using univ

Define polygons figures as follows:

```
square(Side)
triangle(Side1,Side2,Side3)
circle(R)
...
```

We'd like to define a predicate that enlarges each of these figures

```
enlarge(Fig,Factor,Fig1).
```

Here's one way:

```
enlarge(square(A),F,square(A1)) :-
    A1 is F*A.
enlarge(circle(R),F,circle(R1)) :-
    R1 is F*R1.
...
```

Using **univ**, we can do it much more elegantly:

```
enlarge(Fig,F,Fig1) :-
    Fig=..[Type|Parameters],
    multiplylist(Parameters,F,Parameters1),
    Fig1=..[Type|Parameters1].

multiplylist([],_,[]).

multiplylist([X|L],F,[X1|L1]) :-
    X1 is F*X, multiplylist(L,F,L1).
```

## call, functor, arg

**call** allows you to call a predicate. E.g.,

```
Goal=..[Functor | Arglist].
call(Goal).
```

Alternatively, you can do this with **functor** and **arg**.

```
functor(Term,F,N)
```

**functor** is true if F is the principal functor of Term and N is the arity of F.

```
arg(N,Term,A)
```

**arg** is true if A is the Nth argument in Term, assuming that arguments are numbered from left to right starting with 1.

E.g.,

```
?- functor(t(f(X),X,t),Fun,Arity).
```

```
Fun=t
Arity=3
```

```
?- arg(2,f(X,t(a),t(b)),Y).
Y=t(a)
```

```
?- functor(D,examdate,3),
    arg(1,D,22),
    arg(2,D,april),
    arg(3,D,2004).
```

```
D=examdate(22,april,2004)
```

## assert/retract

Here is an example illustrating how clauses may be added and deleted from the Prolog data base. The example shows how to simulate an assignment statement by using **assert** and **retract** to modify the association between a variable and a value.

```
:- dynamic x/1 .
```

```
x(0).           % provide an initial value
```

```
assign(X,V) :- Old =..[X,_], retract(Old),  
               New =..[X,V], assert(New).
```

Here is an example using the assign predicate.

```
?- x(N).
```

```
N = 0
```

```
Yes
```

```
?- assign(x,5).
```

```
Yes
```

```
?- x(N).
```

```
N = 5
```

## Other Useful Syntax

Semi-colon for disjunction:

```
happy(X) :- fed(X),wellslept(X),drydiaper(X)  
           ;outside(X).
```