Variables

?<variable-name>

?x ?y ?z

Variable on LHS used in RHS

```
CCLIPS> (duck-sound quack)
CLIPS> (defrule make-quack
           (duck-sound ?sound)
       =>
           (assert (sound-is ?sound)))
CLIPS> (facts)
f-0     (duck-sound quack) CF 1.00
For a total of 1 fact.
CLIPS> (run)
CLIPS> (facts)
f-0     (duck-sound quack) CF 1.00
f-1     (sound-is quack) CF 1.00
For a total of 2 facts.
CLIPS>
```

A variable in the LHS is used in the right hand side to assert a new fact. This is the beginings of rules perpetuating futher rules.

Multiple Matchings

```
CLIPS> (defrule whodunit
    (duckshoot ?hunter ?who)
=>
    (printout t ?hunter " shot " ?who crlf))
CLIPS> (reset)
CLIPS> (assert (duckshoot Brian duck))
<Fact-1>
CLIPS> (run)
Brian shot duck
CLIPS> (assert (duckshoot duck Brian))
<Fact-2>
CLIPS> (run)
duck shot Brian
CLIPS> (assert (duckshoot duck))
<Fact-3>
CLIPS> (agenda)
```

A rule with two arguments are matched. Note that when a rule with only one argument, it is not matched.

Address of a Fact

```
(clear)

CLIPS> (assert (bachelor Dopey))

<Fact-0>
CLIPS> (facts)

f-0     (bachelor Dopey) CF 1.00
For a total of 1 fact.
CLIPS>(defrule get-married
        ?duck <- (bachelor Dopey)
       =>
        (printout t "Dopey is now happily married " ?duck crlf)
        (retract ?duck))
CLIPS> (run)
Dopey is now happily married <Fact-0>
CLIPS> (facts)
CLIPS>
```

The fact has now been retracted.
Note that the output of *?duck* is *<Fact-0>*.
The **Address** of the fact, (bachelor Dopey) was needed for the retract function, but not pretty for output.

A more general Rule

```
CLIPS> (clear)
CLIPS> (defrule marriage
?duck <- (bachelor ?name)
=>
(printout t ?name " is now happily married" crlf)
(retract ?duck))
CLIPS> (deffacts good-prospects
(bachelor Dopey)
(bachelor Dorky)
(bachelor Dicky))
CLIPS> (reset)
CLIPS> (run)
Dicky is now happily married
Dorky is now happily married
Dopey is now happily married
CLIPS> (facts)
f-0     (initial-fact) CF 1.00
For a total of 1 fact.
CLIPS>
```

Two types of variables are used here, one to store the address of the fact so it could be retracted and one to store a symbol for a printout statement

Wild Cards

```
CLIPS> (clear)
CLIPS> (defrule dating-ducks
(bachelor Dopey ?)
=>
(printout t "Date Dopey" crlf)
)
CLIPS> (deffacts duck
            (bachelor Dicky)
            (bachelor Dopey)
            (bachelor Dopey Mallard)
            (bachelor Dopey Junior)
            (bachelor Dopey Mallard Junior))
CLIPS> (reset)
CLIPS> (run)
Date Dopey
Date Dopey
CLIPS>
```

Only two matched because the fact had to have exactly two names to specify the bachelor, those with one or three names did not activate.

Additional Rules for multiple names

```
CLIPS> (defrule dating-ducks-one-name
(bachelor Dopey)
=>
(printout t "Date Dopey" crlf))
CLIPS> (defrule dating-ducks-three-names
(bachelor Dopey ? ?)
=>
(printout t "Date Dopey" crlf))
CLIPS> (reset)
CLIPS> (run)
Date Dopey
Date Dopey
Date Dopey
Date Dopey
CLIPS>
```

Now there are three rules, one for each name length.

Wild Card: Rest of list

```
CLIPS> (clear)
CLIPS> (deffacts duck
(bachelor Dicky)
(bachelor Dopey)
(bachelor Dopey Mallard)
(bachelor Dopey Junior)
(bachelor Dopey Mallard Junior)
)
CLIPS> (defrule dating-ducks
           (bachelor Dopey $?)
=>
           (printout t "Date Dopey" crlf))
CLIPS> (reset)
CLIPS> (run)
Date Dopey
Date Dopey
Date Dopey
Date Dopey
CLIPS>
```

Now with one function, all cases involving one, two and three names can be included.

The rest as Variable

```
CLIPS> (clear)
CLIPS> (deffacts duck
(bachelor Dicky)
(bachelor Dopey)
(bachelor Dopey Mallard)
(bachelor Dopey Junior)
(bachelor Dopey Mallard Junior))
CLIPS> (defrule dating-ducks
           (bachelor Dopey $?names)
        =>
           (printout t "Date Dopey "  ?names crlf))
CLIPS> (run)
Date Dopey (Mallard Junior)
Date Dopey (Junior)
Date Dopey (Mallard)
Date Dopey()
CLIPS>
```

Now the wild card with the rest list is saved and printed on the RHS.
Note that even the empty list is significant

Matching Anywhere

```
CLIPS> (deffacts duck
                 (bachelor Dopey Mallard)
                 (bachelor Mallard Dopey)
                 (bachelor Junior Mallard Dopey)
                 (bachelor Dopey)
                 (bachelor Dopey Mallard Junior))
CLIPS> (defrule dating-ducks
           (bachelor $?first Dopey $?last)
        =>
           (printout t ?first " Dopey " ?last crlf))
CLIPS> (run)
() Dopey (Mallard Junior)
() Dopey (Junior)
() Dopey (Mallard)
() Dopey ()
CLIPS>
```

Using the fact that the empty list is significant, we can match Dopey anywhere in the set of arguments.

Wild Wild and Specifically Wild Matches

```
CLIPS> (clear)
CLIPS> (deffacts ducks
(bachelor Mr George Dopey Duck)
(bachelor Dr Harry Dopey Duck Mallard)
(bachelor Harry Dopey Duck)
(bachelor Dopey)
(bachelor Ms Sally Dopey Mallard)
)
CLIPS> (defrule dating-ducks
            (bachelor ? $?first Dopey ?)
        =>
            (printout t ?first " Dopey is dating" crlf))
CLIPS> (reset)
CLIPS> (run)
(Sally) Dopey is dating
() Dopey is dating
(George) Dopey is dating

CLIPS> (defrule dating-ducks
            (bachelor ? $?first Dopey $?)
                =>
            (printout t ?first " Dopey is dating" crlf))
CLIPS> (reset)
CLIPS> (run)
(Sally) Dopey is dating
() Dopey is dating
(Harry) Dopey is dating
(George) Dopey is dating
CLIPS>
```

Note the number of arguments that are matched

Multiple Conditions to Satisfy

```
(defrule ideal-duck-bachelor
(bill big ?name)
(feet wide ?name)
=>
(printout t "The ideal Duck is " ?name crlf)
)
CLIPS> (deffacts duck-assets
           (bill big Dopey)
           (bill big Dinky)
           (bill little Dorky)
           (feet wide Dopey)
           (feet narrow Dinky)
           (feet wide Dorky))
CLIPS> (reset)
CLIPS> (run)
The ideal Duck is Dopey
CLIPS> (reset)
CLIPS> (matches ideal-duck-bachelor)
Matches for Pattern 1
f-1
f-2
Matches for Pattern 2
f-4
f-6
Partial matches for CEs 1 - 2
f-1,f-4
Activations
f-1,f-4
CLIPS>
```

Making a List

```
CLIPS> (defrule ideal-duck-bachelor
           (bill big ?name)
           (feet wide ?name)
          =>
           (printout t "The ideal Duck is " ?name crlf)
           (assert (move-to-front ?name)))
CLIPS> (defrule move-to-front
           ?move-to-front <- (move-to-front ?who)
           ?old-list <- (list $?front ?who $?rear)
         =>
           (retract ?move-to-front ?old-list)
           (assert (list ?who ?front ?rear))
           (assert (change-list yes)))
CLIPS> (defrule print-list
           ?change-list <- (change-list yes)
           (list $?list)
          =>
          (retract ?change-list)
          (printout t "List is: " ?list crlf))
CLIPS> (deffacts duck-assets
          (bill big Dopey)
          (bill big Dinky)
          (bill little Dorky)
          (feet wide Dopey)
          (feet narrow Dinky)
          (feet wide Dorky))
CLIPS> (deffacts duck-bachelor-list (list Dinky Dorky Dopey))
CLIPS> (reset)
CLIPS> (run)
The ideal Duck is Dopey
List is: (Dopey Dinky Dorky)
CLIPS>
```