(exit) Leaves CLIPS
; etc Use ";" for comments

FACTS
(assert (duck) (owl)) creates 2 facts. Can't assert a fact twice.
(assert (duck nil)) asserts a fact with an empty field.
(facts) lists the active facts, and their numbers.
(retract 2) removes fact number 2.
(watch facts) shows the fact list at every operation.
   also (unwatch all) asserts "initial fact", removes other facts and rules,
   then asserts all of the deffacts (to prepare for a run).
(reset)

Can create facts to be asserted at (reset):

(deffacts walk
   (status walking)
   (walksign walk))
(list-deffacts) shows the names of deffacts currently loaded.
(ppdeffacts walk) prints a particular deffacts nicely.

Convention: start facts with a description:

(assert (hunter-game Brian duck))
(assert (animal-is horse))

RULES

example with printout:
   (defrule duck "comments can go here"
      (animal-is duck)
   =>
      (printout t "quack quack" crlf))

Can also retract facts:

(defrule older-ducks
   ?x <- (duck-age ?age)
   =>
      (retract ?x)
      (assert (new-duck-age (+ ?age 1))))

Can test the value of a bound variable:

(defrule factorial
   ?old <- (num-is ?x ?y)
   (test (> ?x 1))
   =>
      (assert (num-is (- ?x 1) (* ?y (?x)))
      (retract ?old))
Simple Automobile Diagnostic Example

(defrule p1
    ?p <- (start)
    =>
    (printout t "engine turns? ")
    (assert (turns (read)))
    (retract ?p))

(defrule p2
    (turns no)
    =>
    (printout t "lights work? ")
    (assert (lights (read))))

(defrule rule2
    (turns no)
    (lights no)
    =>
    (printout t "problem is battery" crlf))

(defrule rule3
    (turns no)
    (lights yes)
    =>
    (printout t "problem is starter" crlf))

(defrule p5
    (turns yes)
    =>
    (printout t "tank has gas? ")
    (assert (tgas (read))))

(defrule p6
    (turns yes)
    (tgas yes)
    =>
    (printout t "carb has gas? ")
    (assert (cgas (read))))

(defrule rule4
    (tgas yes)
    (cgas yes)
    =>
    (assert (egas yes)))

(defrule rule5
    (turns yes)
    (egas yes)
    =>
    (printout t "problem is sparkplugs" crlf))

(deffacts startup
    (start))
Blocks-World Example

(deffacts initial-state
  (block A) (block B) (block C) (block D) (block E) (block F)
  (on-top-of nothing A)
  (on-top-of A B)
  (on-top-of B C)
  (on-top-of C floor)
  (on-top-of nothing D)
  (on-top-of D E)
  (on-top-of E F)
  (on-top-of F floor)
  (move-goal C on-top-of E))

(defrule move-directly
  ?goal <- (move-goal ?block1 on-top-of ?block2)
  (block ?block1)
  (block ?block2)
  (on-top-of nothing ?block1)
  ?stack-1 <- (on-top-of ?block1 ?block3)
  ?stack-2 <- (on-top-of nothing ?block2)
  =>
  (retract ?goal ?stack-1 ?stack-2)
  (assert (on-top-of ?block1 ?block2))
  (assert (on-top-of nothing ?block3))
  (printout t ?block1 "moved on top of" ?block2 "." crlf))

(defrule move-to-floor
  ?goal <- (move-goal ?block1 on-top-of floor)
  (block ?block1)
  (on-top-of nothing ?block1)
  ?stack <- (on-top-of ?block1 ?block2)
  =>
  (retract ?goal ?stack)
  (assert (on-top-of ?block1 floor))
  (assert (on-top-of nothing ?block2))
  (printout t ?block1 "moved on top of floor." crlf))

(defrule clear-upper-block
  (move-goal ?block1 on-top-of ?)
  (block ?block1)
  (on-top-of ?block2 ?block1)
  (block ?block2)
  =>
  (assert (move-goal ?block2 on-top-of floor)))

(defrule clear-lower-block
  (move-goal ? on-top-of ?block1)
  (block ?block1)
  (on-top-of ?block2 ?block1)
  (block ?block2)
  =>
  (assert (move-goal ?block2 on-top-of floor)))

CLIPS> (reset)
CLIPS> (run)
A moved on top of floor.
B moved on top of floor.
D moved on top of floor.
C moved on top of E.
7 rules fired