Tema 14
Clips: Acciones procedimentales

La función if

- **Ejemplo**

```
(defrule elige-jugador
  ?fase <- (fase elige-jugador)
  =>
  (retract ?fase)
  (printout t "Elige quien empieza: computadora o humano (c/h) ")
  (bind ?jugador (read))
  (if (or (eq ?jugador c) (eq ?jugador h))
      then (assert (turno ?jugador))
           (assert (fase elige-numero-de-piezas))
      else (printout t ?jugador " es distinto de c y h" crlf)
           (assert (fase elige-jugador))))

(defrule elige-numero-de-piezas
  ?fase <- (fase elige-numero-de-piezas)
  =>
  (retract ?fase)
  (printout t "Escribe el numero de piezas: ")
  (bind ?n (read))
  (if (and (integerp ?n) (> ?n 0))
      then (assert (numero-de-piezas ?n))
      else (printout t ?n
                     " no es un numero entero mayor que 0" crlf)
           (assert (fase elige-numero-de-piezas))))
```

La función if

```
(defrule eleccion-humana
  ?turno <- (turno h)
  ?pila <- (numero-de-piezas ?n&:(> ?n 1))
  =>
  (retract ?turno)
  (printout t "Escribe el numero de piezas que coges: ")
  (bind ?m (read))
  (if (and (integerp ?m) (>= ?m 1) (<= ?m 3) (< ?m ?n))
      then (retract ?pila)
           (bind ?nuevo-numero-de-piezas (- ?n ?m))
           (assert (numero-de-piezas ?nuevo-numero-de-piezas))
           (printout t "Quedan " ?nuevo-numero-de-piezas
                     " pieza(s)" crlf)
           (assert (turno c))
      else (printout t "Tiene que elegir "
                       "un numero entre 1 y 3" crlf)
           (assert (turno h))))
```

- Sintaxis

```
(if <expresion>
    then <accion>*
    [else <accion>*])
```

La función while

- **Ejemplo 1**

```
(defrule elige-jugador
  ?fase <- (fase elige-jugador)
  =>
  (retract ?fase)
  (printout t "Elige quien empieza: "
              "computadora o humano (c/h) ")
  (bind ?jugador (read))
  (while (not (or (eq ?jugador c) (eq ?jugador h))) do
         (printout t ?jugador " es distinto de c y h" crlf)
         (printout t "Elige quien empieza: ")
                     "computadora o humano (c/h) ")
         (bind ?jugador (read)))
  (assert (turno ?jugador))
  (assert (fase elige-numero-de-piezas)))

(defrule elige-numero-de-piezas
  ?fase <- (fase elige-numero-de-piezas)
  =>
  (retract ?fase)
  (printout t "Escribe el numero de piezas: ")
  (bind ?n (read))
  (while (not (and (integerp ?n) (> ?n 0))) do
         (printout t ?n " no es un numero entero "
                     "mayor que 0" crlf)
         (printout t "Escribe el numero de piezas: ")
         (bind ?n (read)))
  (assert (numero-de-piezas ?n)))
```

La función while

```
(defrule eleccion-humana
  ?turno <- (turno h)
  ?pila <- (numero-de-piezas ?n&:(> ?n 1))
  =>
  (retract ?turno)
  (printout t "Escribe el numero de piezas que coges: ")
  (bind ?m (read))
  (while (not (and (integerp ?m)
                   (>= ?m 1)
                   (<= ?m 3)
                   (< ?m ?n))) do
         (printout t "Tiene que elegir un numero entre 1 y 3"
                   crlf)
         (printout t "Escribe el numero de piezas que coges: ")
         (bind ?m (read)))
  (retract ?pila)
  (bind ?nuevo-numero-de-piezas (- ?n ?m))
  (assert (numero-de-piezas ?nuevo-numero-de-piezas))
  (printout t "Quedan " ?nuevo-numero-de-piezas " pieza(s)" crlf)
  (assert (turno c)))
```

- Sintaxis

```
(while <expresion> [do]
       <action>*)
```

La función while

- **Ejemplo 2**

```
(deffunction jugador-elegido ()
  (printout t "Elige quien empieza: "
              "computadora o humano (c/h) ")
  (bind ?jugador (read))
  (while (not (or (eq ?jugador c) (eq ?jugador h))) do
         (printout t ?jugador " es distinto de c y h" crlf)
         (printout t "Elige quien empieza: "
                     "computadora o humano (c/h) ")
         (bind ?jugador (read)))
  ?jugador)

(defrule elige-jugador
  ?fase <- (fase elige-jugador)
  =>
  (retract ?fase)
  (assert (turno =(jugador-elegido)))
  (assert (fase elige-numero-de-piezas)))
```

La función while

```
(deffunction piezas-elegidas ()
  (printout t "Escribe el numero de piezas: ")
  (bind ?n (read))
  (while (not (and (integerp ?n) (> ?n 0))) do
         (printout t ?n " no es un numero entero "
                     "mayor que 0" crlf)
         (printout t "Escribe el numero de piezas: ")
         (bind ?n (read)))
  ?n)

(defrule elige-numero-de-piezas
  ?fase <- (fase elige-numero-de-piezas)
  =>
  (retract ?fase)
  (assert (numero-de-piezas =(piezas-elegidas))))
```

La función while

```
(deffunction piezas-cogidas-de (?n)
  (printout t "Escribe el numero de piezas que coges: ")
  (bind ?m (read))
  (while (not (and (integerp ?m)
                   (>= ?m 1)
                   (<= ?m 3)
                   (< ?m ?n))) do
         (printout t "Tiene que elegir un numero "
                     "entre 1 y 3" crlf)
         (printout t "Escribe el numero de piezas que coges: ")
         (bind ?m (read)))
  ?m)

(defrule eleccion-humana
  ?turno <- (turno h)
  ?pila <- (numero-de-piezas ?n&:(> ?n 1))
  =>
  (retract ?turno ?pila)
  (bind ?m (piezas-cogidas-de ?n))
  (bind ?nuevo-numero-de-piezas (- ?n ?m))
  (assert (numero-de-piezas ?nuevo-numero-de-piezas))
  (printout t "Quedan " ?nuevo-numero-de-piezas " pieza(s)" crlf)
  (assert (turno c)))
```

Acciones definidas con while

```
(deffunction turno-de-jugador-elegido ()
  (printout t "Elige quien empieza: computadora o humano (c/h) ")
  (bind ?jugador (read))
  (while (not (or (eq ?jugador c) (eq ?jugador h))) do
         (printout t ?jugador " es distinto de c y h" crlf)
         (printout t "Elige quien empieza: "
                     "computadora o humano (c/h) ")
         (bind ?jugador (read)))
  (assert (turno ?jugador)))

(defrule elige-jugador
  ?fase <- (fase elige-jugador)
  =>
  (retract ?fase)
  (turno-de-jugador-elegido)
  (assert (fase elige-numero-de-piezas)))
```

Acciones definidas con while

```
(deffunction numero-de-piezas-elegidas ()
  (printout t "Escribe el numero de piezas: ")
  (bind ?n (read))
  (while (not (and (integerp ?n) (> ?n 0))) do
         (printout t ?n " no es un numero entero "
                     "mayor que 0" crlf)
         (printout t "Escribe el numero de piezas: ")
         (bind ?n (read)))
  (assert (numero-de-piezas ?n)))

(defrule elige-numero-de-piezas
  ?fase <- (fase elige-numero-de-piezas)
  =>
  (retract ?fase)
  (numero-de-piezas-elegidas))
```

Acciones definidas con while

```
(deffunction coge-piezas (?n)
  (printout t "Escribe el numero de piezas que coges: ")
  (bind ?m (read))
  (while (not (and (integerp ?m)
                   (>= ?m 1)
                   (<= ?m 3)
                   (< ?m ?n))) do
         (printout t "Tiene que elegir un numero "
                     "entre 1 y 3" crlf)
         (printout t "Escribe el numero de piezas que coges: ")
         (bind ?m (read)))
  (bind ?nuevo-numero-de-piezas (- ?n ?m))
  (assert (numero-de-piezas ?nuevo-numero-de-piezas))
  (printout t "Quedan " ?nuevo-numero-de-piezas " pieza(s)" crlf))

(defrule eleccion-humana
  ?turno <- (turno h)
  ?pila <- (numero-de-piezas ?n&:(> ?n 1))
  =>
  (retract ?turno ?pila)
  (coge-piezas ?n)
  (assert (turno c)))
```

Observaciones

- **Ayuda sobre las funciones procedimentales**

  `(help) // FUNCTION_SUMMARY // PROCEDURAL_FUNCTIONS`

- **Relación de funciones procedimentales**
  - `(bind <variable> <expresion>*)`
  - `(if <expresion> then <accion>* [else <accion>*])`
  - `(while <expresion> [do] <accion>*)`
  - `(loop-for-count <rango> [do] <accion>*)`
  - `(progn <expresion>*)`
  - `(return [<expresion>])`
  - `(break)`

- **Recomendaciones del uso de funciones procedimentales:**
  - **Uso juicioso**
  - **No escribir acciones con complejos anidamiento de `if` y `while`**