Tema 11:
Acciones y funciones

Ejemplo

Deseamos que nuestro programa pida los datos al usuario de forma interactiva. Usaremos la función *read*.

```
(defrule pedir-x1
  (calcular)
  (not (x1 ?))
 =>
  (printout t crlf "Primer valor: ")
  (assert (x1 (read)))
)
(defrule pedir-x2
  (calcular)
  (not (x2 ?))
 =>
  (printout t crlf "Segundo valor: ")
  (assert (x2 (read)))
)
```

Ejemplo (cont)

Deseamos que nuestro programa pida los datos al usuario de forma interactiva. Usaremos la función *read*.

```
(defrule sumar-valores
  ?fx1 <- (x1 ?x1)
  ?fx2 <- (x2 ?x2)
  ?cal <- (calcular)
 =>
  (assert (resultado (+ ?x1 ?x2)))
  (retract ?fx1 ?fx2 ?cal)
)

(defrule comenzar-calculos
 =>
  (assert (calcular))
  (printout t "* Preparado para calcular" crlf)
)
```

Funciones predicado

```
(integerp <expresión>)

(floatp <expresión>)

(numberp <expresión>)

(symbolp <expresión>)

(stringp <expresión>)

(lexemep <expresión>)
```

Funciones predicado

(evenp <expresión>)

(oddp <expresión>)

(multifieldp <expresión>)

Funciones predicado

(eq <expresión> <expresión>+)

(neq <expresión> <expresión>+)

(= <expresión-num> <expresión-num>+)

(<> <expresión-num> <expresión-num>+)

(< <expresión-num> <expresión-num>+)

(<= <expresión-num> <expresión-num>+)

(> <expresión-num> <expresión-num>+)

(>= <expresión-num> <expresión-num>+)

Funciones predicado

```
(and <expresión>+)
(or <expresión>+)
(not <expresión>)
```

Funciones multicampo

```
(create$ <expresión>*)
(length$ <expr-multicampo>)
(nth$ <expr-entera> <expr-multicampo>)
(member$ <expresión> <expr-multicampo>)
(subsetp <expr-multicampo> <expr-multicampo>)
(subseq$  <expr-multicampo>
          <exp-entera-inicio>
          <exp-entera-fin>)
```

Funciones multicampo

```
(first$ <expr-multicampo>)
(rest$ <expr-multicampo>)
(explode$ <expr-cadena>)
(implode$ <expr-multicampo>)
```

Funciones multicampo

```
(insert$  <expr-multicampo>
          <exp-entera>
          <expresión>+)
(replace$ <expr-multicampo>
          <exp-entera-inicio>
          <exp-entera-fin>
          <expresión>+)
```

Funciones multicampo

```
(delete$  <expr-multicampo>
          <exp-entera-inicio>
          <exp-entera-fin>)
```

Funciones de cadena

```
(str-cat <expresión>*)
(sym-cat <expresión>*)
(str-length <expr-cadena-o-símbolo>)
(sub-string    <expr-entera> <expr-entera>
               <expr-cadena>)
(str-compare   <expr-cadena-o-símbolo>
               <expr-cadena-o-símbolo>)
```

Funciones de E/S

```
(open <nombre-fichero> <nombre-lógico> <modo>)
```

Cadena	***Modo***
"r"	Sólo lectura
"w"	Sólo escritura
"r+"	Escritura y lectura
"a"	Sólo añadir
"wb"	Escritura binaria

Funciones de E/S

```
(close [<nombre-lógico>])

(printout <nombre-lógico> <expresión>*)

(read [nombre-lógico>])
(readline [nombre-lógico>])
```

Funciones matemáticas

```
(+ <expr-num> <expr-num>+)
(- <expr-num> <expr-num>+)
(* <expr-num> <expr-num>+)
(/ <expr-num> <expr-num>+)
(div <expr-num> <expr-num>+)
(mod <expr-num> <expr-num>)
```

Funciones matemáticas

```
(sqrt <expr-num>)
(** <expr-num> <expr-num>)
(round <expr-num>)
(abs <expr-num>)
(max <expr-num>+)
(min <expr-num>+)
```

Funciones procedurales

```
(bind <variable> <expresión>*)


(if <expresión>
  then <acción>*
  [else <acción>*])
```

Funciones procedurales

```
(switch   <expr-prueba>

          <sentencia-caso>

          [<sentencia-defecto>])

<sentencia-caso> ::=
  (case <expr-comparación> then <acción>*)

<sentencia-defecto> ::=

  (default <acción>*)
```

Funciones procedurales

```
(while <expresión> [do] <acción>*)

(loop-for-count <rango> [do] <acción>*)

<rango> ::=

  <índice-final> |
  (<variable> <índice-final>) |

  (<variable> <índice-inicio> <índice-final>)

<índice-inicio> ::= <expr-entera>

<índice-final> ::= <expr-entera>
```

Funciones procedurales

```
(return [<expresión>])

(break)
```

Ejemplo

```
(deftemplate persona
  (multislot nombre)
  (multislot direccion))

(defrule pedir-fichero
  =>
  (printout t "Nombre fichero: ")
  (assert (fichero (readline))))

(defrule pedir-nombre-pila
  =>
  (printout t "Introduzca nombre de pila: ")
  (assert (nombre (read))))
```

Ejemplo

```
(defrule iniciar-lectura
  (fichero ?f)
  (nombre ?n)
  =>
  (open ?f entrada "r")
  (bind ?nombre (explode$ (str-cat (readline
  entrada))))
  (bind ?dir (explode$ (str-cat (readline
  entrada))))
  (assert (persona (nombre $?nombre) (direccion
  $?dir))))
```

Ejemplo

```
(defrule leer
  (persona (direccion ?dir&~EOF $?))
  =>
  (bind ?nombre (explode$ (str-cat (readline
  entrada))))
  (bind ?dir (explode$ (str-cat (readline
  entrada))))
  (assert (persona (nombre $?nombre) (direccion
  $?dir))))
```

Ejemplo

```
(defrule fin-leer
  ?p <- (persona (direccion EOF $?))
  =>
  (retract ?p)
  (assert (fase borrar)))

(defrule cerrar-fichero  ; Se cierra el fichero de
  entrada.
  (fase borrar)
  =>
  (close entrada))
```

Ejemplo

```
(defrule borrar-sobrantes
  (fase borrar)
  (nombre ?n1)
  ?p <- (persona (nombre ?n2&~?n1 $?))
  =>
  (retract ?p))
```