REPRESENTACIÓN DEL CONOCIMIENTO - CLIPS

Práctica 2

1. Estructuras condicionales
2. Estructuras iterativas
3. Instrucciones Read y Readline

1. ESTRUCTURAS CONDICIONALES

If...then...else

(**if** <expression>
then
<action>*
[**else**
<action>*])

Pueden anidarse las sentencias if, else es opcional en CLIPS, aunque no siempre debe usarse. Es más recomendable su uso en funciones cuando sea necesario, pero evitar su uso en reglas, ya que internamente se convertirán en reglas independientes (ver estudio del funcionamiento del RETE). Por tanto, es aconsejable que el programador no utilice esta instrucción en los consecuentes de las reglas y explícitamente cree las reglas necesarias en vez de dejar esta conversión al motor de inferencias.

```
(defrule valvulas_cerradas
   (temp Alta)
   (valvula ?v cerrada)
=>
  (if (= ?v 6)  then
         (printout t "ATENCION: La valvula especial " ?v " esta cerrada" crlf)
         (assert (realizar operacion especial))
  else
         (printout t "La valvula " ?v " esta cerrada" crlf)
  ))
```

*Esta regla **DEBE DESCOMPONERSE** en dos reglas:*

```
(defrule valvula6_cerrada
(temp Alta)
(valvula 6 cerrada)
=>
(printout t "ATENCION: La valvula especial esta cerrada"  crlf)
```

```
(assert (realizar operacion especial)))

(defrule valvulas_normales_cerradas
(temp Alta)
(valvula ?v&~6 cerrada)
=>
(printout t "La valvula " ?v " esta cerrada" crlf))
```

Switch

(**switch** <test-expression>
<case-statement>*
[<default-statement>])
<case-statement> ::=
(case <comparison-expression> then <action>*)
<default-statement> ::= (default <action>*)

```
(defglobal ?*x* = 0)   ;; define una variable global
(defglobal ?*y* = 1)
(deffunction foo (?val)
      (switch ?val
          (case ?*x* then ?*y*)
          (case ?*y* then ?*x*)
          (default none)))
```

```
CLIPS> (foo 0)
1
CLIPS> (foo 1)
0
CLIPS> (foo 2)
None
```

2. ESTRUCTURAS ITERATIVAS

While

(**while** <expression> [do]
<action>*)

Ejemplo

```
(defrule valvulas
(valvulas_abiertas ?v)
```

```
=>
(while (> ?v 0)
        (printout t "La válvula " ?v " está abierta" crlf)
        (bind ?v (- ?v 1))))
```

Loop-for-count

(**loop-for-count** <range-spec> [do] <action>*)
<range-spec> ::= <end-index> |
(<loop-variable> <start-index> <end-index>) |
(<loop-variable> <end-index>)
<start-index> ::= <integer-expression>
<end-index> ::= <integer-expression>

Ejemplos

```
(loop-for-count 2 (printout t "Hello world" crlf))


 (loop-for-count (?conta 2 4) do
        (loop-for-count (?contb 1 3) do
            (printout t ?conta " " ?contb crlf)))
```

3. Read

(read [<logical-name>])

Donde <logical-name> es un parámetro opcional para leer desde el lugar asociado a ese nombre, que puede ser un archivo o la entrada estándar cuando se escribe **t** o cuando no se especifica nada (de stdin).
Devuelve un tipo de datos primitivo, por lo que espacios, retornos de carro, tabuladores sólo actúan como delimitadores y no forma parte del valor devuelto a menos que se incluyan entre comillas dobles.

Ejemplo 1

```
(defrule inicial
;; no hay condiciones en el antecedente
=>
        (printout t "Escriba su DNI ")
        (bind ?id (read))

        (printout t "Escriba su PIN  ")
        (bind ?pin (read))

        (printout t "Escriba el importe que desea retirar  ")
        (bind ?d (read))
```

```
        (assert (usuario ?id ?pin ?d))
); inicial
```

Ejemplo 2

```
CLIPS> (open "datos.txt" misdatos "w")
TRUE
CLIPS> (printout misdatos "rojo verde")
CLIPS> (close)
TRUE
CLIPS> (open "datos.txt" misdatos)
TRUE
CLIPS> (read misdatos)
red
CLIPS> (read misdatos)
green
CLIPS> (read misdatos)
EOF
CLIPS> (close)
TRUE
CLIPS>
```

4. Readline

Similar a **read** pero permite introducir una cadena de texto completa en vez de un solo campo. **read** suele detenerse cuando se encuentra un delimitador mientras que **readline** lo hace cuando se encuentra un retorno de carro, un punto y coma, o el símbolo EOF. Los tabuladores y los espacios forman parte de la cadena devuelta por **readline.**
(readline [<logical-name>])

Ejemplo

```
CLIPS> (open "datos.txt" misdatos "w")
TRUE
CLIPS> (printout misdatos "rojo verde")
CLIPS> (close)
TRUE
CLIPS> (open "datos.txt" misdatos)
TRUE
CLIPS> (readline misdatos)
" rojo verde "
CLIPS> (readline misdatos)
EOF
CLIPS> (close)
TRUE
```

1. El cajero automático

Sea un sistema para permitir la obtención de dinero a través de un cajero automático mediante tarjeta de crédito. Se usarán diferentes reglas para validar la tarjeta de un usuario y para concederle el dinero pedido cuando se cumplan una serie de condiciones.
Usaremos tres hechos estructurados:

1. Usuario: que representa la información de la persona que accede al cajero y teclea el pin y la cantidad de dinero que desea obtener:
   Usuario: DNI Titular, Pin, Dinero que desea obtener, por defecto 0.
2. Tarjeta: que contiene la información que tendría la tarjeta:
   Tarjeta: Pin, DNI Titular, Nº intentos, Límite de dinero autorizado a extraer en un mismo día, Año de expiración de la tarjeta (por simplificar no tenemos en cuenta ni mes ni día), y un slot que indica si la tarjeta está Validada o no (valores permitidos Si o No, por defecto a No). Todos los campos menos el pin y el DNI deben tener asociado un valor inicial por defecto.
3. Cuenta representa la información de la cuenta asociada a la tarjeta con el saldo actual:
   Cuenta: DNI Titular, Saldo de la cuenta, y un estado con los posibles valores: enPantalla (mostrado el saldo en pantalla), dineroEntregado, Inicial (valor por defecto), SuperaLimite y SinSaldo.

El proceso se puede descomponer en dos pasos principales:

- **Validación de la Tarjeta:** donde tendremos 3 reglas principales:
  1. Regla Supera_Intentos: Cuando se supera el límite se informa mediante un mensaje. El número de intentos debe verificarse antes que cualquier otro requisito.
  2. Regla Pin_Invalido: Cuando no se valida el Pin se informa mediante mensaje.
  3. Regla Valida_Tarjeta: La validacion consiste en 3 pasos: valida intentos, valida fecha, valida pin.
- **Comprobación de Saldo y Entrega del dinero**:
  1. Regla Muestra_Saldo: cuando la tarjeta se ha validado se muestra el saldo en pantalla, y se actualiza el estado de la cuenta (enPantalla).
  2. Regla Saldo_NoSuficiente: Si no tiene saldo muestra mensaje en pantalla.
  3. Regla Comprueba_Limite1: Si supera el límite establecido por el banco muestra mensaje en pantalla.
  4. Regla Comprueba_Limite2: Si supera el límite establecido por la tarjeta muestra mensaje en pantalla.
  5. Regla Entrega_Dinero: Muestra mensaje con el nuevo saldo y la cuenta pasa al estado DineroEntregado. Si el cajero da el dinero al usuario se almacenará internamente este nuevo saldo.

  ** Para comprobar la fecha usaremos una variable global ?*ANNO=2023
  ** El límite del banco puede ser otra variable global con el valor ?*LIMITE1=900

1. Implementa en CLIPS este sistema, creando al menos dos funciones para realizar los cálculos aritméticos: Decrementar un valor en 1 y Calcular la diferencia entre dos valores (aunque sean muy sencillos y puedan realizarse directamente).
   Los hechos deben ser estructurados, y con las especificaciones del enunciado.

2. Crea una regla inicial para simular la interfaz del cajero, y que permita obtener la información del usuario: DNI, Pin y Cantidad de dinero que desea sacar del cajero y transformarla en un hecho de la siguiente forma:
   (assert (usuario (DNI 1234567) (Pin 1212) (Cantidad 300)))
3. Realiza una traza de ejecución de este sistema contemplando los distintos casos: superar el número de intentos, pin invalido, acceso correcto y entrega del dinero correctas, acceso correcto y no entrega del dinero. Indica en cada caso cómo se va actualizando la base de hechos y la agenda.

Introduce los datos de los siguientes usuarios para probar el sistema:

Usuario1: (DNI 123456) (Pin 1212) (Cantidad 300))
Usuario2: (DNI 456456) (Pin 1211) (Cantidad 200))
Usuario3: (DNI 456456) (Pin 4545) (Cantidad 3000))

Para facilitar esta simulación, los datos de las tarjetas y de las cuentas suponemos que se encuentran cargados en el sistema (usa deffacts para definir los siguientes hechos iniciales y algunos más que necesites).

(tarjeta (DNI 123456) (Pin 1212) (Intentos 3) (Limite 500) (Anno 2026))
(tarjeta (DNI 456456) (Pin 4545) (Intentos 3) (Limite 500) (Anno 2026)
(tarjeta (DNI 000111) (Pin 0011) (Intentos 0) (Limite 500) (Anno 2026)

(cuenta (DNI 123456) (Saldo 5000))
(cuenta (DNI 456456) (Saldo 33))
(cuenta (DNI 000111) (Saldo 30000))

2. Gestión de Válvulas

Cada válvula se identifica por un nombre, pueden tener 2 estados, abierta o cerrada. Además poseen un valor de presión y dos valores de temperatura, el primero hace referencia a la temperatura interna y el otro a la temperatura externa. Por defecto cualquier valor está a 0, y la válvula cerrada.

En la base de hechos se encuentran los siguientes hechos iniciales:

(valvula (nombre Entrada) (T1 101) (T2 35) (presion 1))
(valvula (nombre Salida) (T1 101) (T2 155) (presion 5))
(valvula (nombre Pasillo1) (T1 99) (T2 37) (estado cerrada))

Las tres reglas que componen la base de conocimientos es la siguiente:

- **R1:** Si una válvula está abierta con un valor de presión de 5, entonces la válvula se cierra y se baja la presión a 0.

- **R2:** Si una válvula cerrada tiene un valor de presión menor de 10 y una temperatura T1 mayor de 35 grados entonces esta válvula deberá abrirse y aumentar la presión en función de la temperatura T1.

  ⇒ Para aumentar la presión crea una función que reciba como argumentos la presión y la temperatura 1 de la válvula: mientras T1 sea mayor de 35 grados aumenta la presión en una unidad, y decrementa la temperatura en 5 grados.

- **R3:** Si dos válvulas distintas, v1 y v2, tienen la misma temperatura T2, y la temperatura T1 de la válvula v2, es menor que T2, entonces se decrementa la temperatura T2 de la válvula v2 y se abren ambas válvulas.

  ⇒ Para decrementar la temperatura crea una función que reciba como argumentos las dos temperaturas, T1 y T2, si la temperatura T2 es mayor que la temperatura T1 entonces T2= T2-T1