Manipulación de hechos en CLIPS.

Víctor Tomás Tomás Mariano[1], Felipe de Jesús Núñez Cárdenas[1], Efraín Andrade Hernández

[1]Universidad Autónoma del Estado de Hidalgo: Escuela Superior de Huejutla.
Huejutla de Reyes, Hidalgo, México, C.P. 43000
{victor_tomasm, felipe.huejutla}@hotmail.com
[2]Universidad Autónoma del Estado de Hidalgo: Centro de Investigación en Tecnologías de la Información y Sistemas (CITIS).
Pachuca de Soto, Hidalgo, México, C.P. 42180

Resumen. Se hace una descripción de la implementación de hechos en lenguaje de programación CLIPS, su representación interna, las operaciones básicas que se pueden hacer sobre los hechos: agregar, modificar, eliminar y hacer consultas definidas por el usuario. La entrada de datos desde el teclado.

Palabras Clave: CLIPS, hechos, representación de hechos.

1 Introducción.

La representación de hechos en clips debe obedecer a la sintaxis del lenguaje, cuando los hechos se cargan en la memoria de trabajo se les asigna una enumeración por el intérprete, estas direcciones las podemos manejar como si fueran apuntadores que guardan la dirección de una variable si hablamos de lenguajes como C o C++ solo por mencionar un ejemplo. En clips es el intérprete se encarga de asignar la dirección de cada hecho, entre las restricciones que impone el lenguaje es que, el contenido de un hecho es único así como también su dirección, por lo que si queremos manipular estas direcciones es necesario abstraer la información del hecho para lograr buenos resultados.

Aparentemente las direcciones de los hechos se asignan de forma continua y enumerada como si fueran registros dentro de un arreglo unidimensional, si analizamos la forma de acceder a un arreglo, basta con saber el "índice" o posición dentro del arreglo para acceder al elemento. Sin embargo, cuando se manipulan los hechos en clips: eliminar, duplicar o modificar, las direcciones también sufren alteraciones en la asignación del número de hecho.

2 Tipos de hechos en clips.

Un hecho, en inglés *fact*, es un trozo de información que se almacena en una lista de hechos (*fact-list*). A cada hecho en la memoria de trabajo MT se le asigna un número o identificador en el formato: *Fact-XXX*, donde *XXX* es el número consecutivo conforme el orden en que se insertan los hechos, por ejemplo, *Fact-5*, significa el hecho que tiene el índice número 5.

Los hechos que se manejan en clips son dos: hechos ordenados y hechos no ordenados, en ambos casos se manejan direcciones de hechos para su representación en la MT.

Los hechos ordenados. Son aquellos que se cargan en la memoria sin seguir un "patrón" o plantilla para su captura. Están formados por un *símbolo* seguido de cero o más campos separados por espacios y todo delimitado por paréntesis. Un modo fácil de entender este tipo de hechos es interpretando el primer campo, que es un *símbolo*, como una relación y el resto de los campos como los *términos* que se relacionan vía esa relación [2].

En una relación el orden de los términos es importante. No es lo mismo decir *"Juan es hijo de José"* que decir *"José es hijo de Juan"*. Una manera de insertar y representar estos hechos sería: `(assert (hijo Juan Jose))` `(assert (hijo Jose Juan))`, resultado en figura 1.

Figura 1.- Ejemplo de hechos ordenados en clips

Es importante resaltar que los hechos ordenados codifican la información conforme la posición que guardan en el hecho almacenado, por lo que el usuario, cuando accede a esa información, debe hacer una correcta interpretación de los datos [2]. En la figura 1, se observa que se asignó f-0 y f-1 para los hechos no ordenados capturados.

Hechos no ordenados o *deftemplate*. A diferencia de los hechos ordenados, este tipo de hechos se le *asigna un nombre a cada campo*, y a cada campo un valor que debe almacenar. Por ejemplo si deseamos representar los atributos de un objeto alumno: nombre, edad, estatura y calificación, se puede hacer:

```
(deftemplate alumno “datos de un alumno”
 (slot nombre)
 (slot edad)
 (slot estatura)
 (slot calificacion)
)
```

Figura 2.Plantilla alumno

A partir de la plantilla “alumno” se puede agregar hechos a la memoria de trabajo asignando a cada campo el valor respectivo:

```
(assert (alumno (nombre Juan) (edad 18) (estatura 1.8) (calificacion 10) ) )
(assert (alumno (nombre Pedro) (edad 12) (estatura 1.5) (calificacion 9) ) )
```

Salida se muestra en la figura 3.

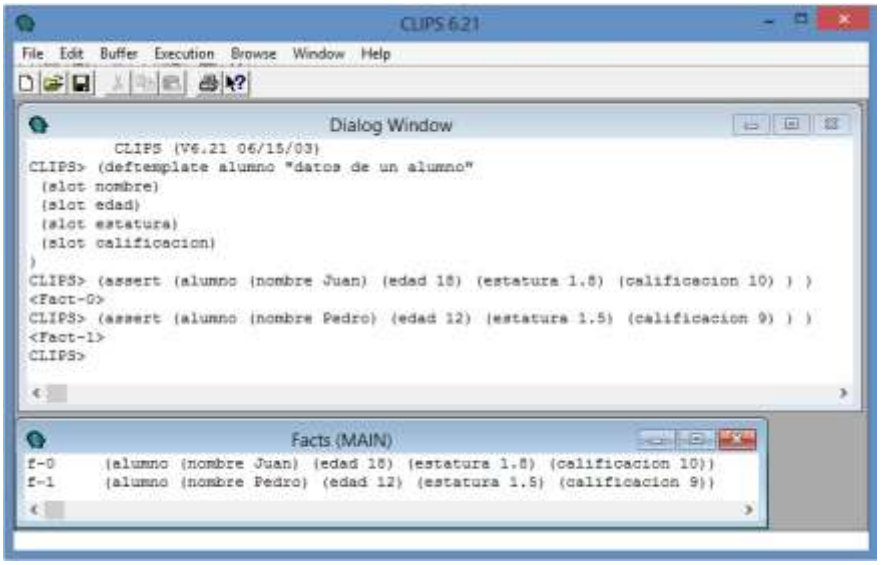

Figura 3.- Ejemplo de hechos no ordenados con deftemplate

El campo “calificacion” de la plantilla alumno se ha dejado sin acento, debido a que la sintaxis de clips no acepta. En la figura 3, los hechos se le asignaron f-0 y f-1.

El orden en que son introducidos el campo y valor, no importa, clips se encarga de representarlo de forma adecuada en la memoria de trabajo, es decir, se puede introducir un hecho como:

`(assert (alumno (edad 20) (nombre Jose) (calificacion 8) (estatura 1.75) ))`, si agregamos este hecho, se puede observar que clips mantiene el orden de los campos y sus valores con concordancia con la definición de la plantilla “alumno”.

Cada uno de los campos de un hecho recibirá el nombre de casilla, que puede ser simple o múltiple. Es ***simple*** cuando el campo recibe un valor y es ***múltiple*** cuando puede tener más de un valor posible. Aquí se usan hechos simples.

La captura de los hechos desde la ventana de comandos es recomendable si se desea probar pocos hechos, pero en ocasiones, se tiene información de hechos a priori, en estos casos, se recomienda utilizar el comando **deffacts,** por ejemplo, con la plantilla "alumno" se puede definir una lista de hechos de alumnos, tal como sigue:

```
;definición de la plantilla alumno
(deftemplate alumno
   (slot nombre)
   (slot edad)
   (slot estatura)
   (slot calificacion)
)
; hechos iniciales
(deffacts hechos_iniciales
(alumno (nombre Victor)(edad 22)(estatura 1.8)(calificación 0))
(alumno (nombre Tomas)(edad 22)(estatura 1.8)(calificación 0))
(alumno (nombre Mariano)(edad 22)(estatura 1.8)(calificación 0))
(alumno (nombre Felipe)(edad 22)(estatura 1.8)(calificación 0))
(alumno (nombre Jesus)(edad 22)(estatura 1.8)(calificación 0))
)
```

Figura 4: Plantilla y hechos iniciales con ***deftemplate*** y ***deffacts***

El programa de la figura 4 se captura desde el editor de código fuente en clips con el nombre *alumnos.clc*. Para cargar los hechos en la memoria de trabajo es necesario usar el comando **"*load*"** desde el menú ***"file"*** de la interfaz de clips, posteriormente visualizar los hechos en la MT tecleando el comando (***reset)*** en la ventana de comandos.

Restricción de Tipos de Datos.

El código de la plantilla alumno, figura 1, deja muy abierto la posibilidad de introducir cualquier valor en cada campo, se puede introducir un hecho como:

```
(assert (alumno (edad nose) (nombre 1020) (calificacion a) (estatura dos_metros) ))
```

Sin embargo, hay una manera de restringir los tipos de datos en cada campo, por ejemplo, se puede declarar la plantilla alumno de la siguiente manera:

```
(deftemplate alumno “datos de un alumno”
 (slot nombre (type STRING))
 (slot edad (type INTEGER))
 (slot estatura (type FLOAT))
 (slot calificacion (type FLOAT))
)
```

a)

```
(deftemplate alumno “datos de un alumno”
 (slot nombre (type SYMBOL))
 (slot edad (type INTEGER) (range 1 99))
 (slot estatura (type FLOAT))
 (slot calificacion (type FLOAT))
)
```

b)

Figura 5. Plantilla alumno: a) restricción en tipos de datos. b) restricción en tipo de datos y rango de valores.

En la figura 5.a) se ha restringido el tipo de dato en los campos, esto permite introducir valores más reales al nombre del campo, por ejemplo, se espera que la edad sea un valor entero y positivo. Declarar el campo nombre como tipo STRING permite introducir valores como cadena de caracteres: “victor”, “jose”, etc., notar que están entre comillas dobles. En la figura 5.b) se ha declarado el campo nombre como SYMBOL, esto permite asignar valores como victor, jose, victor_tomas, notar que no se hace uso de las comillas y que en el campo edad, se usa el comando ***range*** para limitar los valores permitidos en el rango del 1 al 99. Se recomienda consultar el [2][3] para ver más detalles.

Hechos y reglas en clips.

Los hechos en clips se pueden alterar en su contenido: *eliminar, duplicar, modificar, insertar y consultar*. Para ejemplificar estas operaciones sobre los hechos en clips, se hace uso de un menú de opciones, llamada a reglas y a patrones de hechos.

Las reglas en clips tienen la siguiente sintaxis:

SI condición_1, condición_2, condición_n se cumplen, **ENTONCES**
hacer acción_1, acción_2, acción_n, un ejemplo sencillo es la regla EdadMayor:

```
(defrule EdadMayor "imprime el nombre del alumno mayor o igual a 18 años"
  (alumno  (nombre ?nom) (edad ?edad))(test (>= ?edad 18))
```

```
 =>
  (printout t "Es mayor de edad "?nom crlf)
)
```

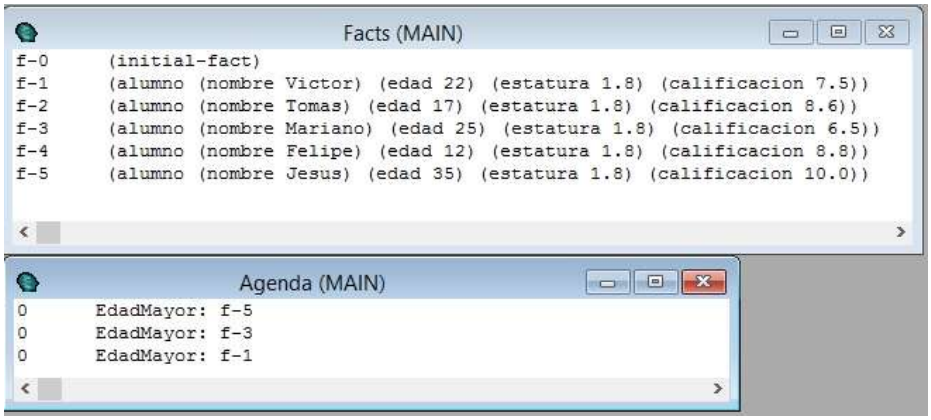

Figura 6. Hechos y activación de la regla EdadMayor

En la figura 6, se observa que la regla EdadMayor se activa con los hechos f-5, f-3 y f-1, estos hechos cumplen con la condición introducida en el comando test, para ejecutar le regla es necesario introducir el comando ***run*** desde la ventana de comandos.

En la regla EdadMayor se hace uso de variables, en este caso son dos, ***?nom*** y ***?edad***, las variables como su nombre lo indica, toman distintos valores, en el ejemplo expuesto, se buscan ***patrones*** de hechos que cumplan con la condición del comando test, se extrae el valor del campo edad para validar si es mayor o igual a 18, el valor del campo nombre se imprime en pantalla. Se pueden usar tantas variables como se desee. El comando test aumenta su potencialidad si combinamos su funcionamiento con los operadores lógicos: and, or, not. Hacer una regla que imprima los alumnos que tengan bajas calificaciones o los alumnos más sobresalientes.

Las reglas lo que hacen es buscar patrones de hechos, también se conoce como emparejamiento de hechos, por ejemplo, podemos imprimir los alumnos que cursen la carrera de sistemas, o aquellos que se apellidan Pérez, etc. dependiendo de la finalidad que se busque con los hechos, en ocasiones también es necesario hacer operaciones aritméticas, solo por manejar un ejemplo, la regla AumentaPunto hace una operación aritmética con los valores del campo calificación, hay que tener cuidado, se puede ciclar una regla y ejecutarse de manera indefinida.

```
(defrule AumentaPunto "Operacion aritmetica en el hecho calificacion"
  ?fn <- (alumno(calificacion ?cal)) (test (and (>= ?cal 7.0)(< ?cal 8.0)))
   =>
```

```
  (bind ?cal (+ ?cal 1.0))
  (modify ?fn (calificacion ?cal))
)
```

La regla AumentaPunto extrae el contenido del campo calificación, valida si el valor es mayor o igual a 7 y menor que 8, los hechos que cumplan esta condición se le aumenta en una unidad el valor. En este caso, se accede al número de hecho, para ello se utiliza una variable ***?fn***, es necesario hacer esto porque en realidad se modifica el contenido de un hecho, si observamos después de ejecutar la regla, clips elimina el hecho anterior y agrega un nuevo hecho conservando los valores de los demás campos excepto el del campo calificación pero con diferente enumeración de hecho. Al cargar los hechos en la MT, se activan las reglas: EdadMayor y AumentaPunto, si manejamos varias reglas, hay que tener cuidado con las reglas que se activan y como afectan los hechos si se llegan a ejecutar, una sugerencia es utilizar un esquema general de la activación de las reglas.

Esquema para operaciones con hechos.

Un ejemplo más completo de las operaciones básicas que se pueden hacer con los hechos en clips se expone a continuación, se tiene una base de hechos de alumnos y se desea hacer:

1. Insertar nuevos hechos de un alumno.
2. Eliminar hechos de un alumno.
3. Buscar hechos de un alumno.
4. Modificar hechos de un alumno.

Un esquema general del modelo implementado se ve en la figura 7.

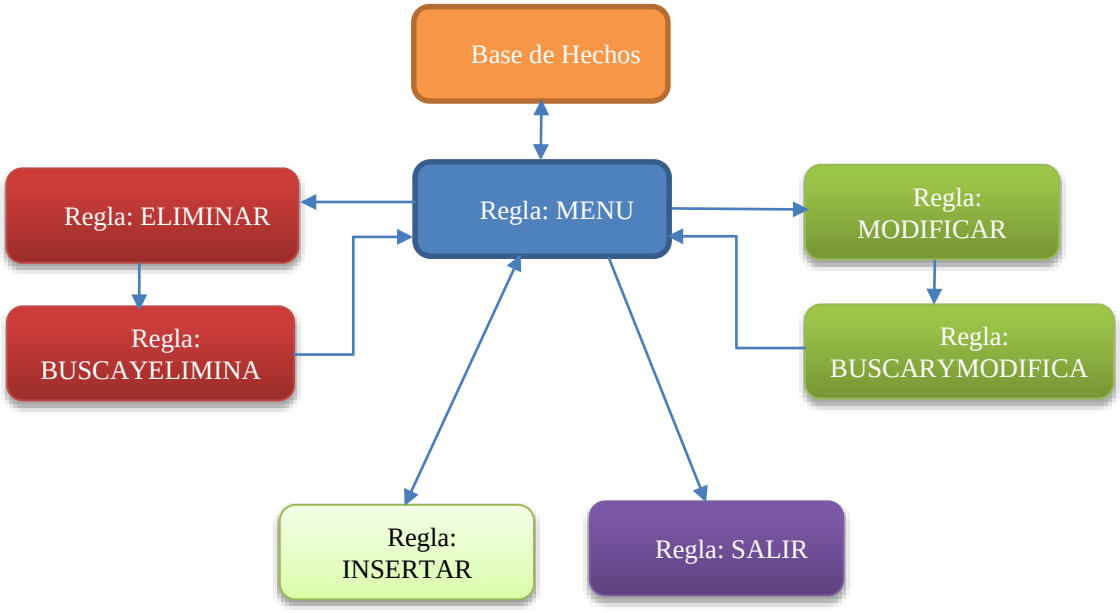

Figura 7: Esquema de activación y ejecución de reglas propuesto.

La dirección de las flechas indican el orden en que se activan las reglas, la regla menú es la que activa las otras reglas, en función de la opción elegida por el usuario. Se hace mención que en esta propuesta se viola el principio de programación imperativa, debido a la forma en que se activan las reglas. Se usa el enfoque de un paradigma de programación estructurada, se hace con fines académicos, para hacer una demostración útil en la activación y ejecución de las reglas en clips. El usuario debe conocer los métodos de inferencia para tener nociones de la forma en que se ejecutan las reglas [1].

A continuación se describe a grandes rasgos los módulos del esquema.

- La regla insertar, lee los datos del teclado, las almacenas en variables respectivas para que con el comando ***assert*** se agregue un nuevo hecho a la memoria de trabajo. Se hace uso del comando ***bind*** para la lectura. Al finalizar regresa el control a la regla menú con el comando ***refresh***.
- Para eliminar un hecho, es necesario "saber" si el hecho esta en la memoria de trabajo, primero se hace una búsqueda, y si es el caso, se activa una segunda regla para eliminar el hecho encontrado. Se hace uso del comando ***retract***.
- Para modificar un hecho, es necesario conocer el hecho a modificar, se busca y si es localizado en la memoria de trabajo, se modifican los valores. Se usa el comando ***modify***.
- Para salir del programa se activa la regla salir y se termina la ejecución con el comando ***halt.***

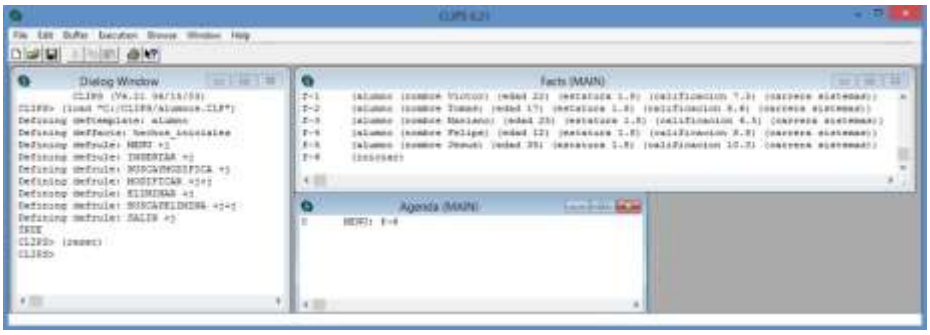

Figura 8. Memoria de trabajo y regla menú activado.

Para iniciar la ejecución del programa es necesario ejecutar el comando (***run***), seguir las instrucciones en pantalla, ver figura 8.

En la figura 9 se muestran los resultados obtenidos para insertar un nuevo hecho y eliminar un hecho de la memoria de trabajo.

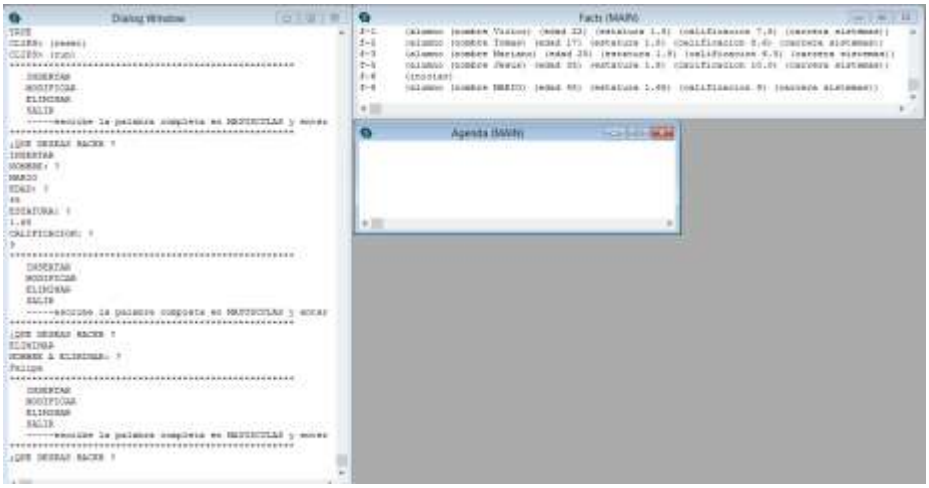

Figura 9. Memoria de trabajo y regla menú activado.

Para probar el programa, se anexa el código fuente. El lector debe poner especial atención en tiempo de ejecución con la activación y desactivación de la reglas, ver figura 9.

3 Conclusiones.

Los hechos en clips de almacenan en la memoria de trabajo, para manipular los hechos es necesario usar reglas que permitan activar y ejecutar en función de la opción elegida por el usuario. Hay que considerar que las reglas son activadas por los hechos, sin embargo, se propone un esquema de activación en función de la elección del usuario.

Referencias.

1. Tomas Mariano V. T., (2014), “Apuntes de Inteligencia Artificial”, curso julio-diciembre 2014.
2. Hernández Molinero L. D.,(2006), “Tutorial de CLIPS”, Universidad de Murcia, España.
3. Mandow Andaluz, L., Perez de la Cruz Molina, J. L. (2003), “Introducción a la programación basada en reglas”, Universidad de Málaga, España.

Anexo.

```
;*****************************************
; MANIPULACION DE HECHOS EN CLIPS
;*****************************************
;SE HACE CON FINES ACADEMICOS.
deftemplate alumno "datos de un alumno"
 (slot nombre (type SYMBOL))
 (slot edad (type INTEGER) (range 1 99))
 (slot estatura (type FLOAT))
 (slot calificacion (type FLOAT))
 (slot carrera (type SYMBOL)(default sistemas))
)

; hechos iniciales
(deffacts hechos_iniciales
(alumno (nombre Victor)(edad 22)(estatura 1.8)(calificacion 7.5))
(alumno (nombre Tomas)(edad 17)(estatura 1.8)(calificacion 8.6))
(alumno (nombre Mariano)(edad 25)(estatura 1.8)(calificacion 6.5))
(alumno (nombre Felipe)(edad 12)(estatura 1.8)(calificacion 8.8))
(alumno (nombre Jesus)(edad 35)(estatura 1.8)(calificacion 10.0))
(iniciar)
)


(defrule MENU
     (iniciar)
      =>
     (printout t "********************************************** " crlf)
     (printout t "   INSERTAR  " crlf)
     (printout t "   MODIFICAR " crlf)
     (printout t "   ELIMINAR  " crlf)
```

```
     (printout t "   SALIR     " crlf)
     (printout t "   -----escribe la palabra completa en MAYUSCULAS y enter " crlf)
     (printout t "************************************************* " crlf)
     (printout t "¿QUE DESEAS HACER ? " crlf)
     (assert (respuesta (read)))
)

(defrule INSERTAR
      ?f<-(respuesta INSERTAR)
      =>
     (printout t "NOMBRE: ?" crlf)
     (bind ?nombre (read))
     (printout t "EDAD: ?" crlf)
     (bind ?edad (read))
     (printout t "ESTATURA: ?" crlf)
     (bind ?est (read))
     (printout t "CALIFICACION: ?" crlf)
     (bind ?cal (read))
     (retract ?f)
     (assert (alumno (nombre ?nombre)(edad ?edad)(estatura ?est)(calificacion ?cal)))
     (refresh MENU)
)

(defrule BUSCAYMODIFICA
       ?f<-(respuesta MODIFICAR)
        =>
       (printout t "NOMBRE A MODIFICAR: ?" crlf)
       (bind ?nombre (read))
       (retract ?f)
       (assert (Modificarhechos ?nombre))
)

(defrule MODIFICAR
       ?f1<-(Modificarhechos ?nombre)
       ?f<-(alumno (nombre ?nombre))
        =>
       (printout t "El" ?nombre " está en el hecho "?f crlf)
       (printout t "NUEVO NOMBRE: ?" crlf)
       (bind ?nombre (read))
       (printout t "NUEVA EDAD: ?" crlf)
       (bind ?edad (read))
       (printout t "NUEVA ESTATURA: ?" crlf)
       (bind ?est (read))
       (printout t "NUEVA CALIFICACION: ?" crlf)
       (bind ?cal (read))
```

```
       (modify ?f (nombre ?nombre)(edad ?edad)(estatura ?est) (calificacion ?cal))
       (retract ?f1)
       (refresh MENU) ;PARA ACTIVAR Y EJECUTAR LA REGLA MENU
)

(defrule ELIMINAR
       ?f<-(respuesta ELIMINAR)
        =>
       (printout t "NOMBRE A ELIMINAR: ?" crlf)
       (bind ?nombre (read))
       (retract ?f)
       (assert (ELIMINAR ?nombre))
)

(defrule BUSCAYELIMINA
       ?f1<-(ELIMINAR ?nombre)
       ?f <-(alumno (nombre ?nombre))
        =>
       (retract ?f1)
       (retract ?f)
       (refresh MENU);PARA ACTIVAR Y EJECUTAR LA REGLA MENU
)

(defrule SALIR
       ?f<-(respuesta SALIR)
        =>
       (retract ?f)
       (halt)
)
```