REPRESENTACIÓN DEL CONOCIMIENTO - CLIPS

Práctica 1

Objetivos:
1. Familiarización con el entorno de CLIPS. Manejo de Hechos y Reglas. Acciones de E/S
2. Diseño estructurado de hechos.

CLIPS

(C Language Integrated Production System) : Lenguaje de especificación de sistemas basados en el conocimiento. El motor de inferencia de CLIPS es una implementación de un algoritmo de encadenamiento hacia delante. El sistema dispone de un intérprete que permite escribir comandos de manera interactiva, opciones de ejecución y depuración de programas, y puede mostrar información sobre el estado actual de la memoria de trabajo y las reglas activadas en cada momento.
En CLIPS se puede representar el conocimiento de tres formas distintas:

- Con Reglas que se prueban con objetos y hechos.
- Con Funciones y funciones genéricas, para conocimiento procedural.
- Utilizando Programación orientada a objetos, permitiendo la herencia y el polimorfismo.

Algunas características generales de CLIPS:

- Los comandos utilizan la notación tipo Lisp, por lo tanto hay que escribir las instrucciones **entre paréntesis**.
- Notación prefija (/ ((* 4 3) 2).
- Distinción entre mayúsculas y minúsculas.
- Funciones aritméticas: + - / * mod div sqrt round integer

1. HECHOS

En un SBR los hechos representan la información del mundo real que se conoce y la que se va obteniendo mediante la aplicación de reglas.

Un hecho puede constar de uno o varios campos. El primer campo suele representar una relación entre los restantes:

```
(<símbolo>      <datos>*)

(nombre         Alicia Mata Gonzalez)
(color          Especie1 Rosa)
(patas          Especie1 Largas)
```

También existen hechos estructurados que permiten agrupar en un mismo hecho distintas características o propiedades de una misma entidad.
Las principales operaciones que se pueden realizar con los hechos se resumen en la siguiente tabla con su orden correspondiente:

Tabla 1. Operaciones con hechos:

Definir plantillas para trabajar con hechos estructurados	**deftemplate**
Definir los hechos iniciales	**deffacts**
Insertar nuevos hechos	**assert**
Borrar un hecho	**retract (reset, clear)**
Modificar un hecho	**modify**
Duplicar un hecho	**duplicate**

A cada hecho se le asigna automáticamente un identificador único (fact-index): f-0, f-1, etc.
** No está permitido el anidamiento de hechos, lo siguiente no es correcto:

```
((persona (gaditana Maria))
```

a) Visualización estática:
- En una ventana aparte: en el menú principal Window - 1 Facts (MAIN)
- En la línea de comandos con **(facts)**.

b) Visualización dinámica: (**watch facts)** / **(unwatch facts)**
La doble flecha hacia la derecha ==> indica que un hecho se añade a la lista de hechos mientras que si está dirigida a la izquierda <== el hecho se está eliminando de la lista.

1.1 Definición de plantillas para disponer de hechos estructurados: DEFTEMPLATE

A través de las plantillas (*templates*) podremos definir hechos de forma estructurada. Las plantillas de definición son análogas a las estructuras de C (*struct*). El formato:

```
(deftemplate <nombre-entidad> [<comentario-opcional>]
             <definición-atributos>* )
```

La definición de los atributos se realiza con **slot** o **multislot,** dependiendo de si el atributo va a tener uno o más argumentos.
Un slot puede tener tres características:
- **type** tipo**:** tipo de datos del slot { SYMBOL, STRING, INTEGER, }
- **allowed-values** lista_valores**:** limita los valores que se pueden introducir a los de la lista
- **default:** valor inicial o por defecto

Abre el editor de clips y guarda esta práctica como un archivo bat, que podrás ir probando de forma incremental cargando el archivo en el entorno con **load o batch**

EJERCICIO 1

```
(clear)
(reset)
(deftemplate persona
      (multislot nombre) ;para almacenar nombre y apellidos
      (slot dni (type INTEGER))
      (slot profesion (default estudiante))
      (slot nacionalidad (allowed-values Es Fr Po Al In It))
)
```

```
(deftemplate intelectual_europeo
     (multislot nombre)
     (slot dni)
      (slot idioma)
     (slot nacionalidad)
)

(list-deftemplates)  ;; para mostrar las plantillas definidas
```

1.2 Definición de los hechos iniciales: DEFFACTS

Los hechos iniciales se definen con **deffacts:**

```
(deffacts <nombre>
     <hecho>)
```

Continuando con el ejercicio 1 …

Hechos:
Juana Bodega Gallego con dni 3334444
Mario Canterio Cansino, con dni 122333 de profesión escritor y nacionalidad española.
Gertrudis es gaditana y Filoberto de Sevilla

```
(deffacts iniciales
    ;; con hechos estructurados
     (persona  (nombre Mario Cantero Cansino)
               (dni 122333)
               (profesion escritor)
               (nacionalidad Es))
      (persona (nombre Juana Bodega Gallego) (dni 3334444))
     ;; con hechos no estructurados
     (ciudad Gertrudis Cadiz)
     (ciudad Filoberto Sevilla)
);;deffacts

(facts)   ;; visualizar los hechos actuales
```

;; no verás nada porque para añadir los hechos iniciales hay que usar la orden **(reset)** que restaura las condiciones iniciales:

```
(reset)
(facts)
```

1.3 Inserción de hechos: ASSERT

La forma más simple y habitual de insertar hechos en la base de hechos se realiza mediante la orden **assert.**

```
(assert <hecho>*)
```

Continuando con el ejercicio 1 …

```
(assert (ciudad Juana Cadiz))
(assert (persona (nombre Alicia Mata Rueda) (dni 1234)))
(assert  (intelectual_europeo (nombre Victor Hugo) (idioma francés)))

(assert (persona (nombre Mateo Duran Barbera)
                 (dni 333221)))
```

1.4 Eliminación de hechos: RETRACT, RESET, …

La eliminación de un hecho concreto se realiza con **retract**: (* es un carácter comodín en retract)

```
(retract índice-hecho)
```

Continuando con el ejercicio 1 …

```
(facts)
(retract 1)
(facts) ;; comprueba que se ha eliminado el hecho 1
(retract 3 5)
(facts) ;; comprueba que se han eliminado los hechos 3 y 5
(retract *)
(facts)  ;; comprueba que se ha eliminado todo

(reset)

¿qué ha pasado después de ejecutar reset? Comprueba la lista de hechos de
la memoria de trabajo
```

Además es posible desde el entorno:

a) La eliminación de todos los hechos con **(reset),** que elimina también las activaciones de la agenda y restaura las condiciones iniciales definidas con deffacts.

b) La eliminación de todos los hechos y construcciones de la memoria de trabajo **(clear)**

1.5 Modificación de hechos: MODIFY

- **modify :** permite modificar los valores de los slots de una plantilla. Para un hecho modificado se genera un nuevo índice de hecho, porque en realidad lo que hace es eliminar el hecho antiguo y agregar uno nuevo con los datos actualizados. Si no es un hecho estructurado (definido con plantilla) no funciona esta instrucción.

```
(modify  <indice-hecho>  <modificador-ranura>+)
```

Continuando con el ejercicio 1 …

```
(reset)
(modify  1 (nombre Maria Cantero Cansino)))
(facts)

¿cuál es el nuevo índice de este hecho?
```

** Al modificar el hecho 1, este desaparece y se crea uno nuevo con un nuevo índice.

1.6 Copiado de hechos: DUPLICATE

duplicate: duplica un hecho y lo actualiza con los valores de los slots que se especifiquen.

```
(duplicate  <indice-hecho>  <modificador-ranura>+)
```

Continuando con el ejercicio 1 …

```
(duplicate 2 (nombre Maria Bodega Gallego) (dni 11))

(facts)
```

2 REGLAS

Mediante las reglas se expresa el conocimiento para deducir nueva información (nuevos hechos).

```
(<Condición>*) => (<Acción>*)
```

Defrule, es la orden para crear una regla, indicando primero las precondiciones (todas las premisas deben cumplirse para activar una regla), a continuación los símbolos => y por último se especifican el conjunto de acciones que se derivan de las condiciones.

```
(defrule <nom_regla>
     <condición>*
=>
     <acción>*)
```

Cuando las precondiciones de una regla se satisfacen, entonces se produce la activación de la regla, y se añade a la **agenda.** La agenda contiene el conjunto de activaciones producidas al realizar la equiparación de los hechos con las condiciones de una regla.

La orden **(agenda)** muestra las reglas que están preparadas para ser ejecutadas. (porque existen hechos que se equiparan con los antecedentes de las reglas, con la orden (run) se ejecutarían)

USO DE VARIABLES

Pueden usarse variables, entendiéndose que están cuantificadas universalmente. Las variables siempre comienzan con la interrogación final **?**:

```
?x    ?temperatura     ?altura   ?persona
```

Continuando con el ejercicio 1 …

Reglas del sistema:

```
;;Los gaditanos y los sevillanos son andaluces

(defrule R1_gaditanos
  (ciudad ?x Cadiz)
  =>(assert(comunidad ?x Andalucia))
  );;R1
(agenda) ;; lista de reglas del sistema

(defrule R2_sevillanos
  (ciudad ?x Sevilla)
  =>(assert (comunidad ?x Andalucia))
 );;R2

(reset)
(agenda) ;; lista de reglas del sistema

(run 1) ;; ejecuta el sistema para disparar la primera regla de la agenda
(facts) ;; comprueba si se han añadido nuevos hechos al sistema
(run 1) ;; siguiente regla de la agenda
(facts)
```

*** La ejecución de un programa finaliza cuando no hay más activaciones en la agenda.

*** El sistema se reinicia con (reset) y con (run)

***** Añade una nueva regla para inferir que los escritores españoles son intelectuales europeos**

```
(defrule R3_intelectuales
   (persona (nombre ?x ?y ?z) (profesion escritor) (nacionalidad Es))
   => (assert (intelectual_europeo (nombre ?x ?y ?z) (nacionalidad
española)))
  );;R3
```

CLIPS está diseñado bajo el principio de **Refracción o Refractariedad:** cuando un conjunto de hechos satisfacen una regla, ésta sólo se dispara una vez. En caso contrario se tendría un bucle infinito de activación de las reglas hasta que los hechos desaparecieran de la lista de hechos.

*** Con la orden **refresh** se pueden recargar las reglas disparadas:

```
(agenda)
(refresh r1_gaditanos)
(agenda)
```

COMODINES

Dentro de los hechos o patrones, un campo se puede sustituir por un comodín. Existen dos tipos de comodines:
De un solo campo: **?**
De varios campos: **$?** Representa a 0 o más casos de un campo.

Si una variable va precedida del símbolo de $ está haciendo referencia a varios valores de un hecho. Por ejemplo para imprimir los nombres de los hermanos de Juan:

EJEMPLO

```
(assert (hermanos Juan Pedro Paula Patricia))

(defrule mostrarhermanos
      (hermanos Juan $?resto)
      =>
      (printout t "Los hermanos de Juan son " $?resto  crlf))

*** La regla del ejercicio 1 que hace referencia a los intelectuales
europeos podría haberse escrito usando comodines en las variables del
nombre y apellidos.

(defrule R3_intelectuales
   (persona (nombre $?n) (profesion escritor) (nacionalidad Es))
   => (assert (intelectual_europeo (nombre $?n) (nacionalidad española)))
  );;R3
```

Tabla2. Otras órdenes del entorno útiles:

; punto y coma para comentarios que acaban con un retorno de carro

(dribble-on nombre-fichero) ; para grabar en un fichero la traza de ejecución
(dribble-off)

```
(list-defrules)
(list-deftemplates)
(list deffacts)

(undefrule <nombre_regla>)
(undeftemplate <nombre_plantilla>)
(undeffacts <nombre-hecho_inicial>)

(watch statistics) / (unwatch statistics)
(watch rules)/ (unwatch rules)
(watch activations)
(unwatch all)

(ppdefrule <nombre_regla>)
(ppdeftemplate <nombre-plantilla>)
(ppdeffacts <nombre-hecho inicial>)
```

3 REFERENCIA A LOS HECHOS A TRAVÉS DE UNA VARIABLE

En algunas situaciones puede resultar interesante guardar la dirección de un hecho, por ejemplo para borrarlo después. Con **<-** es posible asignar el identificador de un hecho a una variable.

EJERCICIO 2

```
(deffacts inicio
     (soltero Jose)
    (soltera Maria)
);;deffacts

 (defrule casamiento
     (comprometidos ?x ?y)
     ?dir1 <- (soltero ?x)
     ?dir2<- (soltera ?y)
     => (assert (casados ?x ?y))
     (retract ?dir1)
     (retract ?dir2)
);;casamiento
```

Desde el entorno:

```
(reset)
(assert (comprometidos Jose Maria))
```

```
(facts)
(run)
```

*** Comprueba que ya no aparecen los hechos que afirman que Jose y Maria están solteros (pero aparecería que siguen comprometidos y casados) ya que comprometidos no se ha borrado

1. El Concesionario

Una tienda de venta de automóviles tiene un portal que aconseja a sus clientes qué coche comprar en función de sus preferencias. La información sobre los modelos de coches que se pueden comprar se muestra en la siguiente tabla.

Modelo	Precio	Tamaño del Maletero	Número de Caballos	ABS	Consumo en Litros
Modelo1	12000	Pequeño	65	No	4,7
Modelo2	12500	Pequeño	80	Sí	4,9
Modelo3	13000	Mediano	100	Sí	7,8
Modelo4	14000	Grande	125	Sí	6,0
Modelo5	15000	Pequeño	147	Sí	8,5

El portal proporciona a los clientes un formulario con las siguientes preguntas:

1. Cantidad de dinero que desea gastarse
2. Maletero pequeño, mediano o grande
3. Mínimo número de caballos del motor
4. Sistema ABS
5. Consumo máximo de combustible a los 100 km

Si el usuario deja algún campo en blanco se asume lo siguiente:

- El precio del coche no debe superar los 13000 euros
- Maletero grande
- Mínimo 80 caballos
- Sistema ABS
- Consumo máximo de 8 litros

1. Diseña un SBR para gestionar esta tienda virtual en CLIPS, utilizando hechos estructurados de acuerdo a la tabla dada anteriormente.