Ejercicios Básicos de CLIPS

Departamento de Informática
Universidad Carlos III de Madrid

1. Dado el siguiente programa en CLIPS:

```
(defrule regla-sumar1
 (declare (salience 10))
 ?a <- (elemento ?x)
=>
 (assert (elemento (+ 1 ?x))))

(defrule regla-parar
 (declare (salience 20))
 (elemento ?x)
 (test (> ?x 99999))
=>
 (halt))

(deffacts hechos-iniciales
  (elemento 1))
```

   1 Sin ejecutar el programa en CLIPS, describe en papel qué hace este programa.

   2 Ahora carga el programa en CLIPS, ejecuta paso a paso e inspecciona el contenido de la agenda y la base de hechos en cada paso. Asegúrate de que entiendes perfectamente como CLIPS ejecuta el programa antes de pasar a la siguiente cuestión. Si no coincide la ejecución con lo descrito en el apartado anterior, vuelve a describir que hace el programa.

   3 ¿Cuántas veces se activa la regla regla-sumar1 con el mismo elemento? ¿Por qué?.

2. Dado el siguiente programa en CLIPS:

```
(defrule regla-sumar-elementos
 (declare (salience 10))
 (elemento ?x)
 (elemento ?y)
=>
 (assert (elemento (+ ?x ?y)))
 (printout t (+ ?x ?y) crlf))

(defrule regla-parar
 (declare (salience 20))
 (elemento ?x)
 (test (> ?x 99999))
=>
 (halt))

(deffacts hechos-iniciales
  (elemento 1))
```

   1 Sin ejecutar el programa en CLIPS, describe en papel qué hace este programa.

2 Ahora carga el programa en CLIPS, ejecuta paso a paso e inspecciona el contenido de la agenda y la base de hechos en cada paso. Asegúrate de que entiendes perfectamente como CLIPS ejecuta el programa antes de pasar a la siguiente cuestión. Si no coincide la ejecución con lo descrito en el apartado anterior, vuelve a describir que hace el programa.

3 Modifica el programa para que produzca la misma salida por pantalla, pero en cada iteración sólo haya una única activación en la agenda.

4 Cambia la estrategia de resolución del conjunto conflicto haciendo (set-strategy random). ¿Qué efecto tiene hacer esto en la ejecución?.

3. Haz un programa en CLIPS que dada una base de hechos con un único hecho inicial del tipo (elemento <valor>), genere una base de hechos con un único hecho, con la misma estructura que el inicial, y cuyo valor sea el factorial del valor inicial.

4. Dado el siguiente programa en CLIPS:

```
(deftemplate elemento
 (slot valor (type INTEGER)))

(defrule regla1
 (declare (salience 10))
 (elemento (valor ?x))
=>
 (assert (valor ?x)))

(defrule regla2
 (declare (salience 5))
 ?a <- (valor ?x)
 (valor ?y)
 (test (< ?x ?y))
=>
 (retract ?a))

(defrule regla3
 (declare (salience 1))
 ?a <- (valor ?x)
=>
 (printout t "Resultado: valor " ?x crlf)
 (retract ?a))

(deffacts hechos-iniciales
 (elemento (valor 1))
 (elemento (valor 8))
 (elemento (valor 5)))
```

1 Explica brevemente qué calcula el programa. Sin ejecutar el programa en CLIPS, describe en papel qué hace este programa.

2 Ahora carga el programa en CLIPS, ejecuta paso a paso e inspecciona el contenido de la agenda y la base de hechos en cada paso. Asegúrate de que entiendes perfectamente como CLIPS ejecuta el programa antes de pasar a la siguiente cuestión. Si no coincide la ejecución con lo descrito en el apartado anterior, vuelve a describir que hace el programa.

3 Ejecuta los siguientes comandos y escribe lo que sale por pantalla tras la ejecución de cada uno

```
CLIPS>(reset)
CLIPS>(watch rules)
CLIPS>(matches regla1)
CLIPS>(run 1)
```

```
CLIPS>(matches regla1)
CLIPS>(matches regla2)
CLIPS>(matches regla3)
CLIPS>(run 1)
CLIPS>(matches regla1)
CLIPS>(matches regla2)
CLIPS>(matches regla3)
```

Explica qué hace el comando matches

5. Dado el siguiente programa en CLIPS:

```
(defrule R1
 (declare (salience 15))
 ?a <- (numero ?x ?u)
 ?b <- (numero ?y ?v)
 (test (> ?u ?v))
=>
 (assert (numero (+ ?x ?y) (+ ?u 1)))
 (retract ?b))

(defrule R2
 (declare (salience 5))
 ?b <-(total ?x)
 (test (> ?x 0))
=>
 (assert (numero 0 1)))

(defrule R3
 (declare (salience 5))
 ?b <-(total ?x)
 (test (> ?x 1))
=>
 (assert (numero 1 2)))

(defrule R4
 (declare (salience 20))
 (total ?a)
 (numero ?x ?a)
=>
 (printout t "OK:" ?x crlf)
 (halt))

(defrule R5
 (declare (salience 1))
=>
 (printout t "ERROR" crlf))
```

1 Explica brevemente para qué sirve este programa.

2 Ahora carga el programa en CLIPS. Introduce en la base de hechos el hecho (total 5). Ejecuta paso a paso e inspecciona el contenido de la agenda y la base de hechos en cada paso. Asegúrate de que entiendes perfectamente como CLIPS ejecuta el programa antes de pasar a la siguiente cuestión. Si no coincide la ejecución con lo descrito en el apartado anterior, vuelve a describir que hace el programa.

3 Escribe qué saldría por pantalla si hiciéramos:

```
CLIPS> (reset)
CLIPS> (assert (total 6))
CLIPS> (run)
```

¿Cuántos ciclos son necesarios para que el programa se tenga?

4 Responder a las mismas preguntas del apartado anterior cambiando el (assert (total 6)), por (assert (total 7)) y por (assert (total 8))

5 Y si en vez de (assert (total 5)) hiciéramos (assert (total -1))

6 Describe 2 casos de uso del comando (matches) con reglas que se pueden disparar y otros 2 casos con reglas que no cumplen todas las precondiciones

6. Escribe un programa CLIPS (reglas y clases necesarias) para calcular la duración media de todas las actividades que una persona realiza durante la visita a una ciudad. Suponiendo que la persona hace todas las actividades de la ciudad en la que está y que los datos vienen representados de la siguiente forma:

```
(definstances personas
  (of persona (nombre Juan) (ciudad Paris))
  (of persona (nombre Ana) (ciudad Edimburgo)))

(definstances actividades
   (of actividad (nombre Torre_Eiffel) (ciudad Paris) (duracion 2))
   (of actividad (nombre Castillo_de_Edimburgo) (ciudad Edimburgo) (duracion 5))
   (of actividad (nombre Louvre) (ciudad Paris) (duracion 6))
   (of actividad (nombre Montmartre) (ciudad Paris) (duracion 1))
   (of actividad (nombre Royal_Mile) (ciudad Edimburgo) (duracion 3)))
```

Después de la ejecución tendríamos que tener la siguiente salida:
La duración media de las actividades de Ana fue 4.0
La duración media de las actividades de Juan fue 3.0

7. Dado el siguiente programa en CLIPS:

```
(deftemplate elemento
 (slot valor (type INTEGER)))

(defrule R1
 (declare (salience 40))
 (elemento (valor ?x))
 (not (inicial ?))
 (test (>= ?x 0))
=>
 (assert (inicial ?x)))

(defrule R2
 (declare (salience 20))
 (elemento (valor ?x))
 (not (elemento (valor 2)))
 (not (borrando))
 (test (> ?x 2))
=>
 (assert (elemento (valor (- ?x 1)))))

(defrule R3
 (declare (salience 15))
 ?a <- (elemento (valor ?x))
 ?b <- (elemento (valor ?y))
 (test (> ?x ?y))
=>
```

```
 (assert (borrando))
 (assert (elemento (valor (* ?x ?y))))
 (retract ?a ?b))

(defrule R4
 (declare (salience 30))
 ?a <- (elemento (valor 0))
=>
 (retract ?a)
 (assert (elemento (valor 1))))

(defrule R5
 (declare (salience 10))
 (inicial ?x)
 (elemento (valor ?y))
 (test (>= ?y 0))
=>
 (printout t "OK:" "(" ?x "," ?y ")" crlf)
 (halt))

(defrule R6
( declare (salience 1))
=>
 (printout t "ERROR" crlf))
```

1 Explica brevemente para qué sirve este programa.

2 Ahora carga el programa en CLIPS. Introduce en la base de hechos el hecho (elemento (valor 3)). Ejecuta paso a paso e inspecciona el contenido de la agenda y la base de hechos en cada paso. Asegúrate de que entiendes perfectamente como CLIPS ejecuta el programa antes de pasar a la siguiente cuestión. Si no coincide la ejecución con lo descrito en el apartado anterior, vuelve a describir que hace el programa.

3 Escribe qué saldría por pantalla si ejecutamos cada uno de los comandos:

```
CLIPS>(reset)
CLIPS>(assert (elemento (valor 3))
CLIPS>(matches R1)
CLIPS>(matches R3)
CLIPS>(matches R6)
```

Explica el significado de cada uno de las ejecuciones anteriores del comando matches