INSTITUTO UNIVERSITARIO
SIANI
INGENIERIA COMPUTACIONAL

CLIPS

Introducción

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

Índice

- Introducción
- Instalación
- Elementos básicos
  - Órdenes
  - Tipos de datos
  - Constructores
- Hechos
  - Ordenados y no ordenados
  - Plantillas
- Reglas
- Primeros ejemplos
  - Hechos iniciales
  - Comodines
  - Variables
  - Condiciones
  - Entrada y salida
  - Estructuras de control
  - Operadores lógicos y matemáticos
  - Funciones
  - Restricciones
  - Comodines y variables
  - Plantillas y condiciones
  - *El coche no arranca*
  - *Vamos al teatro*
  - Logs
  - Consistencia
  - Resolución de conflictos
- Referencias/Documentación

INSTITUTO UNIVERSITARIO SIANI INGENIERIA COMPUTACIONAL
Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es
UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Introducción

- CLIPS: *C Language Integrated Production System*
- Variación muy especializada de Lisp
- Herramienta para desarrollo de **sistemas expertos**, creada en el Johnson Space Center (NASA) en 1986
  - Problemas usualmente resueltos por “expertos humanos” gracias a su importante base de conocimiento sobre el dominio.
  - Los expertos necesitan uno o varios mecanismos de razonamiento para aplicar su conocimiento a los problemas propuestos.



Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Introducción

- Soporta programación lógica, y tanto programación imperativa como orientada a objeto (COOL)
- Facilita diseñar programas dirigidos por datos:
  - En ellos los datos, o hechos, estimulan la ejecución del programa a través del motor de inferencia
- Permite realizar el procesamiento de forma interactiva, mediante la ventana de comandos, o por lotes. Admite la depuración
- Diseñado para facilitar integración con otros lenguajes:
  - Puede llamarse desde otros lenguajes: CLIPS ejecuta una función, retornando resultado y control.
  - CLIPS puede llamar a funciones externas, que devuelven la salida y control a CLIPS.
- Jess *(Java Expert System Shell)*, CLIPS reprogramado en Java con ligeras variaciones
- Extensiones, p.e. para Lógica Borrosa *(FuzzyCLIPS)*

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

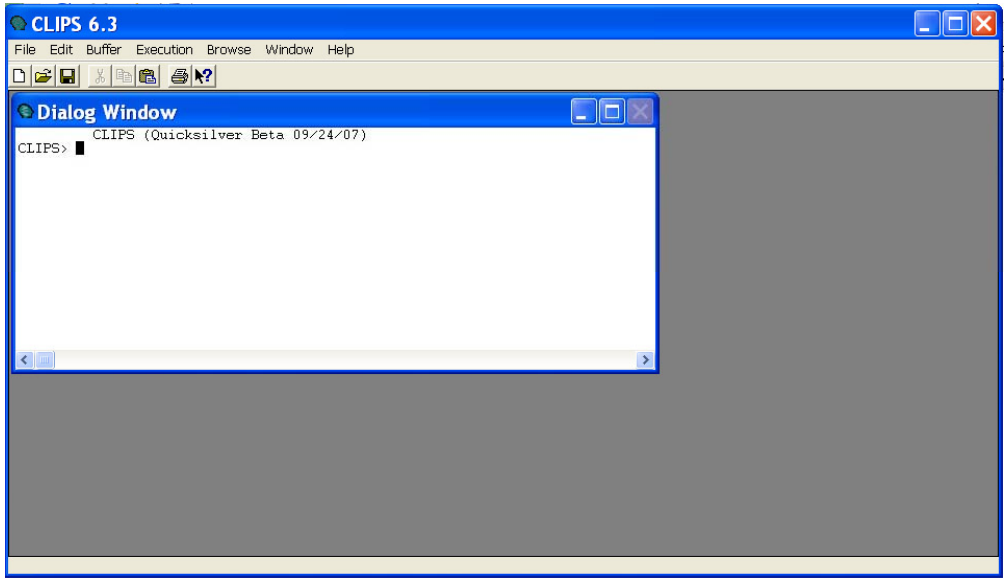
Introducción

- Sistema de producción que incluye:
  - Mantenimiento de la verdad con encadenamiento hacia adelante
  - Adición dinámica de reglas y hechos
  - Diferentes estrategias de resolución de conflictos
- Componentes básicos:
  - Base de hechos: Datos introducidos e inferidos
  - Base de conocimiento: Reglas, funciones, ...
  - Mecanismo de inferencia: Controla la ejecución
- CLIPS proporciona tres elementos básicos de programación:
  - Tipos primitivos de datos
  - Funciones para la manipulación de los datos
  - Constructores

Instalación

Para descargar la aplicación, acude a la página oficial CLIPS.

Disponible para varias plataformas. La versión para Windows dispone tanto de ejecutable en línea, como con ventanas. Haciendo clic sobre el icono de esta última (CLIPSWin) lanzarás la ventana que acompaña a este texto. Para salir teclea *(exit)* o acude a la barra de menú.



Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Elementos básicos: Órdenes

Una vez abierta la interfaz, veamos algunas órdenes o comandos básicos de operación.

Todo comando se escribe siempre entre paréntesis. Los comandos, se ejecutan habitualmente en *top-level* y no devuelven valor, al contrario que las funciones que sí retornan un valor:

(exit) Cierra la interfaz CLIPS.

(run) Lanza la ejecución del programa CLIPS actualmente cargado. Se le puede indicar el número máximo de reglas a lanzar.

(clear) Elimina todos los hechos y reglas almacenados en memoria, equivalente a cerrar y rearrancar CLIPS.

(reset) Elimina sólo los hechos, no las reglas, anulando la agenda y añadiendo los elementos definidos por defecto o iniciales.

(watch <elemento>) Permite realizar depuración del programa.

La barra de menú ofrece acceso a algunos comandos, así como la aparición de ventanas útiles para depuración (p.e. ventana de hechos *Window->Facts Window*).

Elementos básicos: Tipos de datos

- Reales (*float*): 1.5, -0.7, 3.5e-10
- Enteros (*integer*): 1, -1, +3, 65
- Símbolos (*symbols*): Cualquier secuencia de caracteres que no siga el formato de un número. Distingue entre mayúsculas y minúsculas. Ej.: Febrero, febrero, fuego, 35B, fiebre
- Cadenas (*strings*): Deben estar entre comillas
- Direcciones externas (*external-address)*: Estructura de datos externa devuelta por una función escrita en C o Ada
- Direcciones de hechos (*fact-address*): Hechos referenciados por su posición o por un nombre
- Nombres de instancias (*instance-name*)
- Direcciones de instancias (*instance-address*)

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Elementos básicos: Constructores

Permiten al programador añadir elementos tanto a la base de hechos como a la de conocimiento. Modifican el entorno CLIPS:

- **deffunction**: Para definir funciones
- **defglobal**: Para definir variables globales
- **deftemplate**: Para definir plantillas
- **deffacts**: Para definir hechos
- **defrule**: Para definir reglas
- **defmodule**: Para definir módulos
- **defclass**: Para definir clases
- **definstances**
- **defmessage-handler**
- **defgeneric**
- **defmethod**

Hechos

CLIPS mantiene una lista de *hechos* y *reglas*, permitiendo éstas operar con los hechos almacenados en la *lista de hechos*, dado que los hechos son necesarios para disparar o activar las reglas.

Un **hecho** es una forma básica de representación de información, es una pieza de información o **patrón.** Puede tener un campo o varios de tipo numérico, simbólico, cadena, etc., p.e. **(color azul)** or **(padre_de Juan Sara) (nombre "Juan Manuel")**

Los espacios separan distintos símbolos. Observa sin embargo que en una cadena no es lo mismo tener dos espacios en blanco consecutivos que sólo uno.

De similar manera, se debe tener en cuenta el uso de minúsculas y mayúsculas.

Hechos

Un valor o campo puede ser uni o multicampo:

- Unicampo: Cualquier valor de los tipos de datos mencionados anteriormente
- Multicampo: Secuencia de cero o más valores unicampo entre paréntesis. Ejemplos:
  - ( )
  - (x)
  - (hola)
  - (relaciona "rojo" 23 1e10)
- Señalar que no es lo mismo el valor unicampo *hola* que el valor multicampo *(hola)*

Hechos

Los hechos pueden añadirse y eliminarse de **forma dinámica**. Para añadir uno o varios hechos empleamos la orden *assert*. Cada hecho se identificará a continuación mediante un índice único:

```
CLIPS> (assert (tiempo nublado))
<Fact-1>
```

Para eliminar un hecho (o varios), conociendo su identificador o índice utilizamos *(retract <nid>)*, por ejemplo *(retract 1)* elimina el hecho con identificador 1. Podemos eliminarlos todos introduciendo *(retract *)*.

La orden *(clear)* elimina todos los hechos pero también todas las reglas, éstas las trataremos más adelante. El comando **(***reset)* elimina todos los hechos de la lista y posteriormente añade los iniciales, a priori sólo *(initial-fact)*.

Hechos

Tecleando *(facts)* nos permite ver la lista completa de hechos actual, apareciendo cada hecho junto a su índice o identificador único. Para nuestro caso

```
CLIPS> (facts)
f-0     (initial-fact)
f-1     (tiempo nublado)
For a total of 2 facts
```

Para listar los hechos a partir de un identificador usamos *(facts <nid>).*

Para mostrar un rango concreto de hechos podemos teclear los índices límite *(facts <idmin> <idmax>).*

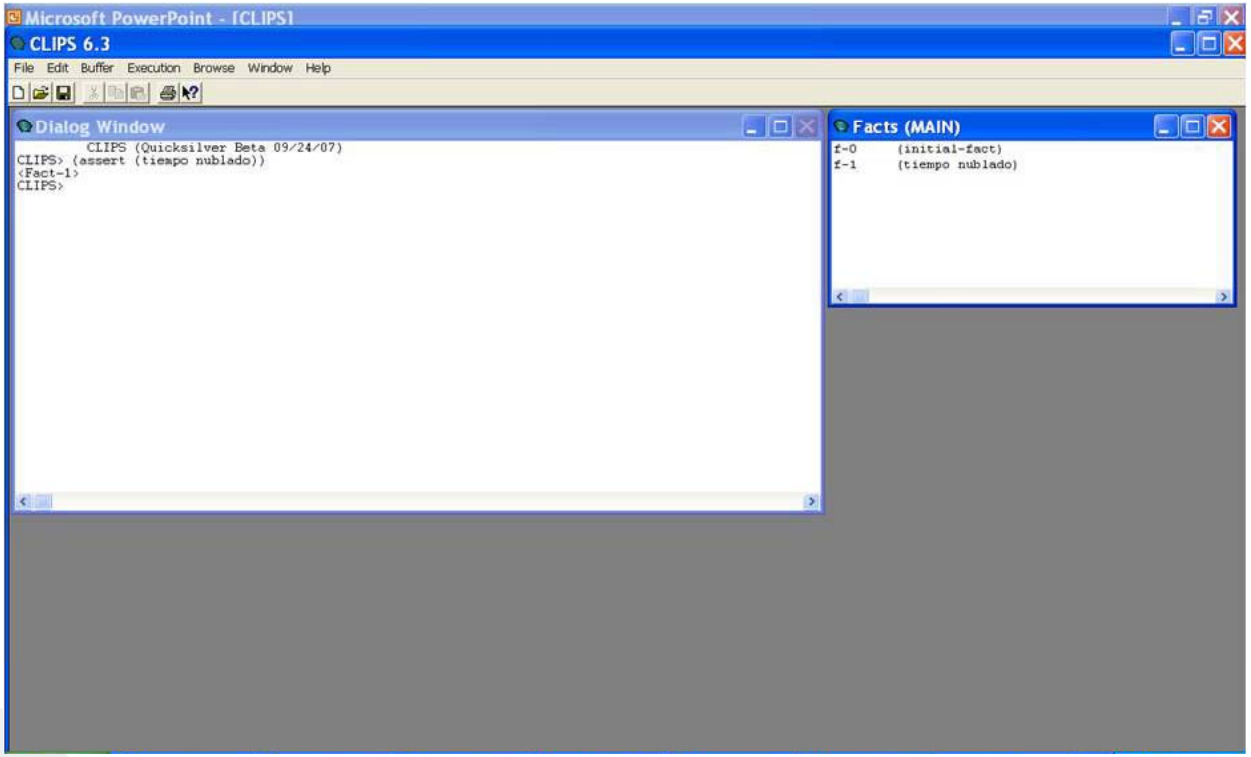
Hechos

Comentábamos que un hecho puede ser eliminado de la base de hechos utilizando el comando *(retract <nid>)*, indicando el índice (* para eliminarlos todos) del hecho que se desea eliminar. Observa en el ejemplo que los índices no se reutilizan tras una eliminación:

```
CLIPS> (reset)
CLIPS> (assert (color azul) (color rojo))
<Fact-2>
CLIPS> (facts)
f-0     (initial-fact)
f-1     (color azul)
f-2     (color rojo)
For a total of 3 facts.
CLIPS> (retract 1)
CLIPS> (facts)
f-0     (initial-fact)
f-2     (color rojo)
For a total of 2 facts.
```

Hechos

La ventana de hechos que se activa por medio del menú *Window->Facts Window* es otra posibilidad muy útil que permite observar los movimientos en la base de conocimiento:



Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Hechos

Ejercicios:

- Inserta el hecho *(color verde)* utilizando *assert*
- ¿Qué ocurre si ahora se ejecuta la orden *reset*?
- Inserta cuatro hechos más.
- Muestra los hechos con índice >=3
- Muestra los hechos del 2 al 4
- Abre la ventana de hechos
- Añade un nuevo hecho
- Elimina el hecho 1
- Elimina todos los hechos

Hechos

Por legibilidad es habitual introducir un hecho (o como veremos luego reglas, etc.) en varias líneas. Debes recordar acabar siempre tecleando ENTER

```
CLIPS> (clear)
CLIPS> (assert (lista-compra
                papas
                cebolla
                ajos
                pescado))
<Fact-0>
CLIPS> (facts)
f-0 (lista-compra papas cebolla ajos pescado)
For a total of 1 fact.
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Hechos

Como herramienta para depuración, la orden *(watch facts)* muestra los hechos cada vez que se insertan o eliminan en la base de conocimiento.

```
CLIPS> (watch facts)
CLIPS> (assert (dia miercoles))
==> f-2     (dia miercoles)
<Fact-2>
```

La orden *(unwatch facts)* desactiva la depuración de la base de hechos. Otros elementos con posibilidad de ser observados:

(watch rules): Muestra las reglas disparadas
(watch activations): Muestra las reglas activadas, los hechos que las han activado, y las reglas que quedan desactivadas.
(watch methods)
(watch deffunctions)
(watch compilations)
(watch slots)
(watch globals)
(watch all)

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Hechos: Ordenados y no ordenados

Los hechos se especifican siempre delimitados por paréntesis, sirviendo éstos de separadores, pudiendo contener uno o varios símbolos.

Recuerda evitar el uso de las tildes, prueba tecleando:

```
(assert (nombre José))
```

El primer símbolo de un hecho se emplea a menudo para indicar una relación entre los restantes símbolos del hecho, p.e.:

```
(padre_de Juan Sara)
```

Siendo en ese caso importante el orden de los mismos. Son **hechos ordenados**.

Hechos: Ordenados y no ordenados

En otras ocasiones el orden no es importante, hablaremos de **hechos no ordenados**, siendo simplemente una relación de campos, donde cada campo tiene su nombre y valor.

(datos-persona (nombre Juan) (apellido Perez))

Para estos últimos CLIPS requiere la definición previa de su plantilla, por medio del constructor *deftemplate,* para especificar el modelo del hecho.

Para los primeros, los ordenados, la plantilla correspondiente se define de forma automática.

Hechos: Plantillas

Para definir una plantilla con el constructor *deftemplate* se especifica el nombre de cada campo*:*

```
(deftemplate datos-persona
    (slot nombre)
    (multislot apellidos)
    (slot edad)
    (slot peso)
    (slot altura)
    (multislot presion-arterial ) )
```

Utilizando *(list-deftemplates)* listamos los nombres de las plantillas definidas, mostrando su contenido con *(ppdeftemplate <nombre>)*. Finalmente *(undeftemplate <nombre>)* permite eliminar la definición, siempre que no existan hechos en la base de hechos que sigan esa plantilla.

Hechos: Plantillas

Los hechos ordenados permiten enlazar trozos de información. Un ejemplo podría ser el estado de forma de una persona:

```
(edad Samuel 20)
(peso Samuel 80)
(altura Samuel 188)
(presion-arterial Samuel 130 80)

(edad Eva 23)
(peso Eva 50)
(altura Eva 155)
(presion-arterial Eva 120 60)
```

Son varios datos unidos por el nombre, es más cómodo utilizar la plantilla:

```
(assert (datos-persona (nombre Samuel) (edad 20)
(peso 80) (altura 188) (presion-arterial 130 80)) )
```

INSTITUTO UNIVERSITARIO SIANI INGENIERIA COMPUTACIONAL
Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es
UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Hechos: Plantillas

Una plantilla permite definir además del nombre, el tipo, los valores por defecto y el rango de sus *slots* o campos. Los tipos posibles son: SYMBOL, STRING, NUMBER, INTEGER y FLOAT.

```
(deftemplate datos-persona
      (slot nombre (type STRING) (default ?DERIVE))
      (slot edad (type FLOAT) (default (* 2.0 3.4)) (range 0.0 100.0)) )
```

Si ahora añadimos un hecho, asignará valores por defecto

```
CLIPS> (assert (datos-persona))
<Fact-1>
CLIPS> (facts)
f-0     (initial-fact)
f-1     (datos-persona (nombre "") (edad 6.8))
For a total of 2 facts.
```

Hechos: Plantillas

Con **?DERIVE** el valor por defecto se obtiene de las restricciones del campo. Por el contrario si se utiliza **?NONE** el valor debe indicarse de forma obligatoria al realizar un assert.

```
(deftemplate dato
      (slot w (default?NONE))
      (slot x (default?DERIVE))
      (slot y (default(gensym*) ))
      (slot z (default-dynamic(gensym*) )) )
```

Al añadir un hecho sin asignar valores a esos campos ocurre:

```
CLIPS> (clear)
CLIPS> (assert (dato))
[TMPLTRHS1] Slot w requires a value because of its (default ?NONE)
   attribute.
```

Hechos: Plantillas

Para la plantilla del ejemplo anterior, si añadimos un hecho indicando un valor para el campo o *slot w*

```
CLIPS> (assert (dato(w 3)))
<Fact-0>
CLIPS> (assert (dato(w 4)))
<Fact-1>
CLIPS> (facts)
f-0 (dato (w 3) (x nil) (y gen1) (z gen2))
f-1 (dato (w 4) (x nil) (y gen1) (z gen3))
For a total of 2 facts.
CLIPS>
```

Se genera de forma automática un símbolo para los campos *y* y *z*. *E*sto se consigue con la función *gensym* que permite de forma automática asignar un símbolo

Hechos: Plantillas

La función *gensym* retorna un símbolo único. Múltiples llamadas proporcionan identificadores de la forma *genX* siendo **X** un entero positivo.

La primera llamada retorna *gen1*, *setgen* permite configurar el primer valor, incrementándose el valor entero en las siguientes

```
(assert (new-id (gensym) flag1 7))
```

Genera en la primera llamada el hecho

```
(new-id gen1 flag1 7)
```

gensym* es similar pero produce un símbolo único actualmente no en uso dentro del entorno CLIPS

Hechos: Plantillas

También es posible especificar los valores permitidos para un *slot* de una plantilla:

```
(deftemplate ficha-personal
    (slot nombre (type STRING))
    (slot genero (type SYMBOL)(allowed-symbol hombre mujer)   )
```

Hay otras opciones al especificar las restricciones:

- allowed-symbols rico pobre
- allowed-strings "Ricardo" "Juan" "Pedro"
- allowed-numbers 1 2 3 4.5 -2.01 1.3e-4
- allowed-integers -100 53
- allowed-floats -2.3 1.0 300.00056
- allowed-values "Ricardo" rico 99 1.e9

Hechos: Plantillas

El constructor *deftemplate* no crea un hecho sino la forma que los hechos pueden tener. Reutilizando la plantilla *datos-persona*:

```
(assert (datos-persona (nombre Samuel) (edad 20) (peso 80) (altura 188) (presion-arterial 130 80)) )
```

Un campo o *slot* no definido toma el valor *NIL* a menos que se especificara alguna regla para su valor por defecto, y que no sea requerido (*?NONE*). El orden empleado al introducir la información puede alterarse al ser hechos no ordenados:

```
(assert (datos-persona (peso 150) (edad 23) (nombre Eva)))
```

Hechos: Plantillas

La función *modify* permite *modificar* un *slot*, sin tener que emplear *retract* y *assert* para ello, conociendo el identificador:

(modify <*id-hecho*> **(altura 200))**

Mostrando posteriormente los hechos tecleando *facts* observaremos el resultado.

Para duplicar un hecho utilizamos *duplicate* que se diferencia del *modify* en no realizar el *retract*, por lo que al final tendremos dos hechos en la base

(duplicate <*id-hecho*> **(altura 100))**

Hechos: Plantillas

Ejercicios:

- Crea una plantilla y define para ella al menos cuatro campos, que no sean todos unicampo, definiendo su tipo y valores por defecto
- Lista las plantillas existentes
- Inserta dos hechos no ordenados
- Muestra los hechos
- Inserta otros dos hechos no ordenados sin proporcionar todos los campos
- Activa el modo *watch* para los hechos
- Inserta dos nuevos hechos y elimina posteriormente uno.
- Muestra los hechos
- Modifica uno de los hechos, duplica otro
- Muestra los hechos
- Elimina la plantilla

Reglas

Las reglas permiten operar con los hechos y consecuentemente realizar programas con alguna utilidad.

Una regla se define con el constructor *defrule* acompañado del nombre de la regla, siguiendo la sintaxis:

SI un patrón o combinación de varios se cumple
ENTONCES realiza una o varias acciones.

```
(defrule nombre_de_regla
    (hecho1)
    =>
    (assert (hecho2)) )
```

No puede haber dos reglas con el mismo nombre.

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Reglas

Una descripción más genérica:

```
(defrule <nombre-regla>
        [<descripción opcional>]
        [(declare (salience <num>))]
        (patrón 1)
        (patrón 2)
        ...
        (patrón N)
        =>
        (acción 1)
        (acción 2)
        ...
        (acción N)
        )
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Reglas

Una vez introducida una o varias reglas, tecleando *(rules)* aparecen los nombres de las presentes en la base de conocimiento. El contenido de una regla se muestra con *(ppdefrule <idregla>)* y *(undefrule <idregla>)* la elimina (***** las elimina todas):

```
CLIPS> (defrule r1 (tiempo nublado) => (assert (llevar paraguas)))
CLIPS> (rules)
r1
For a total of 1 defrule.
CLIPS> (ppdefrule r1)
(defrule MAIN::r1
   (tiempo nublado)
   =>
   (assert (llevar paraguas)))
CLIPS> (undefrule r1)
CLIPS> (rules)
```

Reglas

El motor de inferencia intenta emparejar o encajar los hechos de la lista de hechos con los patrones de las reglas.

Si el encaje ocurre para todos los patrones de una regla, ésta se **activa** o **dispara**. Esta comprobación para una regla sólo se realiza al añadirse nuevos patrones o hechos, o tras eliminar un elemento y volverlo a añadir.

La **agenda** almacena la lista de reglas activadas siguiendo un orden de prioridad.

Existen distintas estrategias de resolución de conflictos que permiten establecer criterios a la hora de insertar una activación de regla en la agenda.

Reglas

Retomemos tras teclear *(clear)* nuestro el ejemplo de los colores *(assert (color rojo))*, añadamos la siguiente regla:

```
(defrule cielo
        (color azul)
        =>
        (assert (cielo-es despejado)))
```

Al lanzar la evaluación de las reglas en base a la lista de hechos, con el comando *(run)*, si existiera el hecho *(color azul)*, se añadiría un nuevo hecho a la lista *(cielo-es despejado)*, circunstancia que podemos comprobar con *(facts)* que no sucede

```
CLIPS> (run)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (color rojo)
For a total of 2 facts.
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Reglas

Añadamos ahora el hecho *(color azul)*, necesario para activar la regla, y ejecutamos

```
CLIPS> (assert (color azul))
<Fact-2>
CLIPS> (run)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (color rojo)
f-2     (color azul)
f-3     (cielo-es despejado)
For a total of 4 facts.
```

Acabamos de ejecutar nuestro primer programa en CLIPS que añade un nuevo hecho.

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Reglas

Activemos ahora la depuración con los mismos hechos y reglas:

```
CLIPS> (reset)
CLIPS> (assert (color azul) (color rojo))
CLIPS> (defrule cielo  (color azul)  =>  (assert (cielo-es despejado)))
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (run)
FIRE    1 cielo: f-1
==> f-3     (cielo-es despejado)
```

Se nos indica la regla activada y el hecho añadido, siendo útil para seguir el proceso seguido por el motor de inferencia

Reglas

Para depurar programas CLIPS son útiles los comandos relacionados con *puntos de ruptura*:

- **(set-break <nombre-regla>)**: Establece un punto de ruptura antes de la ejecución de la regla
- **(remove-break [<nombre-regla>])**: Elimina un punto de ruptura establecido previamente
- **(show-breaks)**: Visualiza los puntos de ruptura establecidos

Las reglas poseen *refracción*: una regla sólo se dispara una vez para un conjunto dado de hechos, con lo que se evitan bucles infinitos.

Reglas

Es posible no definir ningún elemento condicional en el antecedente de una regla, situación en la que se usa automáticamente *(initial-fact)*, el hecho que se añade de forma automática tras ejecutar el comando *(reset)*, como elemento condicional.

También puede no haber ninguna acción en el consecuente, con lo que la ejecución de la regla no tiene en ese caso ninguna consecuencia.

En CLIPS pueden realizarse comentarios colocándolos detrás de un punto y coma (;). Todos los caracteres desde el punto y coma hasta el siguiente salto de línea serán considerados comentarios.

Reglas

La condición o antecedente de una regla puede ser múltiple:

```
(defrule cielo
        (color azul)
        (estamos-en exterior)
        =>
        (assert (cielo-es despejado)))
```

Al igual que la acción resultante, p.e. para mostrar un mensaje por la salida estándar con *printout*:

```
(defrule cielo
        (color azul)
        (estamos-en exterior)
        =>
        (assert (cielo-es despejado))
        (printout t “cielo despejado" crlf) )
```

Recuerda utilizar *reset* para limpiar la base de hechos antes de volver a ejecutar o de lo contrario las reglas no se activarán

Reglas

Como probablemente habrás pensado, no es buena política teclear cada vez todas tus reglas, puedes tenerlas grabadas y cargarlas utilizando **File->Load** para cargar un fichero clp

Desde la consola se carga un fichero clp por medio del comando *load*, por ejemplo:

(load “nombre-fichero.clp”)

Se puede almacenar la información actual de los constructores con *save*, o hacer uso de la interfaz para ello, por ejemplo:

(save “nombre-fichero.clp”)

Reglas

Ejercicios:

- Define una regla
- Lista las reglas existentes
- Define una nueva regla con el mismo nombre. ¿Qué ocurre?
- Define otra regla
- Lista las reglas existentes
- Muestra el contenido de una regla
- Elimina una regla

Primeros ejemplos: Hechos iniciales

Es frecuente tener que utilizar el mismo conjunto de hechos en cada ejecución del programa, equivalente a variables iniciales en un esquema de programación imperativo.

El constructor *deffacts* permite añadir conocimiento en forma de hechos iniciales, tanto ordenados como no ordenados. Los hechos iniciales definidos, se incluyen en la base de hechos tras un *reset* de similar forma al hecho por defecto *initial-fact*.

Por ejemplo:

```
(deffacts arranque (color azul) (color rojo))
```

```
(deffacts gente (datos-persona (nombre Samuel) (edad 20) (peso 80)
    (altura 188) (presion-arterial 130 80)) (datos-persona (nombre
    Mar) (edad 63) (peso 70) (altura 167) (presion-arterial 180 90)))
```

Primeros ejemplos: Hechos iniciales

Los hechos así añadidos se tratan como cualquier otro.

Pueden definirse varios grupos de hechos iniciales, recibiendo cada uno de ellos un identificador al declararse.

El identificador agrupa conceptualmente diferentes hechos sobre el mismo objeto:

```
(deffacts coche
      "información del coche"
      (marca Seat)
      (modelo Ibiza)
      (puertas 5) )
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Hechos iniciales

Como con otros constructores, *(list-deffacts)* muestra los grupos de hechos iniciales definidos con *deffacts*, *(ppdeffacts <id>)* nos permitirá mostrar el contenido de uno de ellos, y *(undeffacts <id>)* suprime los hechos insertados por una orden *(deffacts).*

```
CLIPS> (list-deffacts)
initial-fact
arranque
For a total of 2 deffacts.

CLIPS> (ppdeffacts arranque)
(deffacts MAIN:: arranque
   (color azul)
   (color rojo))
```

Reglas: Hechos iniciales

Ejercicios:

- Define unos hechos iniciales
- Define alguna regla que haga uso no sólo de estos hechos iniciales
- Añade los hechos necesarios para activar la regla
- Prueba a ejecutar varias veces
- Lista las definiciones de hechos iniciales
- Muestra el contenido de una de ellas

Primeros ejemplos: Comodines

Los hechos utilizados hasta ahora en las reglas eran muy simples, al estar referidos a hechos específicos. Los *comodines* permiten especificar reglas que encajen para distintos hechos. Definamos los hechos iniciales y regla siguientes:

```
CLIPS> (deffacts arranque (animal perro) (animal gato) (animal pato))
CLIPS> (defrule animal
    (animal  ?)
    =>
    (printout t "animal encontrado" crlf) )

CLIPS> (run)
animal encontrado
animal encontrado
animal encontrado
```

? es un comodín que encaja con cualquier símbolo de un hecho, en este caso el segundo. Nunca puede colocarse en el primer lugar, es por ello válido **(padre_de ? ?)** pero no **(? Juan ?)**

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Variables

Las variables permiten ofrecer aún mayor flexibilidad, ya que ofrecen la posibilidad de ser reutilizadas en la propia regla. Para ello utilizamos *?nombre-var*. Un ejemplo es la siguiente regla:

```
(defrule lista-animales
    (animal  ?nombre)
     =>
    (printout  t  ?nombre “  encontrado" crlf))

CLIPS> (reset)
CLIPS> (run)
animal encontrado
pato encontrado
animal encontrado
gato encontrado
animal encontrado
perro encontrado
```

Observa que también se activa la regla anterior, *animal*, porque no la hemos eliminado

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Variables

Pueden combinarse el uso de una variable en varias de las condiciones o patrones de una regla. Para ver un ejemplo redefinimos los hechos por defecto con *deffacts* y tras eliminar todas las reglas, con *(clear)*, introducimos una nueva:

```
CLIPS> (deffacts arranque (animal perro) (animal gato) (animal pato) (animal
tortuga) (sangre-caliente perro) (sangre-caliente gato) (sangre-caliente pato)
(pone-huevos pato) (pone-huevos tortuga))

CLIPS> (defrule mamifero
        (animal  ?nombre) (sangre-caliente  ?nombre) (not (pone-huevos  ?nombre))
        =>
        (assert (mamifero  ?nombre))
        (printout  t  ?nombre " es un mamífero" crlf))

CLIPS> (run)
gato es un mamífero
perro es un mamífero
```

Primeros ejemplos: Variables

Pueden aparecer varias variables en la condición:

```
(deffacts arranque (animal perro) (animal gato) (animal pato) (animal
    tortuga) (sangre-caliente perro) (sangre-caliente gato) (sangre-
    caliente pato) (pone-huevos pato) (pone-huevos tortuga) (hijo-de
    perro perrito) (hijo-de gato gatito) (hijo-de pato patito))

(defrule mamifero2
      (mamifero  ?nombre) (hijo-de  ?nombre  ?joven)
      =>
      (assert (mamifero  ?joven))
      (printout  t  ?joven " es un mamífero " crlf))

CLIPS> (run)
gatito es un mamífero
perrito es un mamífero
```

Primeros ejemplos: Variables

Sabemos que podemos eliminar un hecho por su índice con *retract*. Es posible hacerlo en una acción tras conectarlo en una regla con una variable

```
(defrule elimina
      ?hecho <- (mamifero  ?nombre)
      =>
      (printout t "eliminando “ ?hecho crlf) (retract  ?hecho))
```

El operador **<-** almacena la referencia o índice al hecho en la variable **?hecho** y la acción recoge esa variable para realizar el *retract*

```
CLIPS> (run)
gato es un mamífero
eliminando <Fact-12>
perro es un mamífero
eliminando <Fact-13>
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Variables

Las variables también pueden emplearse en el consecuente de una regla por medio de la función *bind* que asigna un valor a una variable, sirviendo ésta como variable temporal:

```
(defrule suma
      (num  ?x) (num  ?y)
      =>
      (bind  ?total (+  ?x  ?y))
      (printout t  ?x " + "  ?y " = “  ?total crlf)
      (assert (total  ?total)))
```

En este caso es una variable temporal que sólo tiene validez dentro de esa regla.

Primeros ejemplos: Variables

Si se necesitan variables globales, es decir para usar en más de una regla, empleamos *defglobal*, estas variables deben nombrarse delimitadas por ***:

```
(defglobal
      ?*var1* = 17  ?*naranjas* = “siete" )
```

Tras realizar un *reset* retoman su valor original. *(get-defglobal-list)* permite obtener la lista, *(show-defglobals)* también su contenido, *(undefglobal var1)* elimina la variable *var1*.

Como el resto de variables, pueden modificar su valor mediante la función *bind*

```
(defrule prueba
      (dato ?)
      => (bind ?*var1* 3))
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Condiciones

test permite establecer condiciones que van más allá de los hechos al permitir hacer uso de variables y valores numéricos

```
(defrule nonecesario
        (test (> 6 5))
        =>
        (printout t “Seis es mayor que cinco" crlf) )
```

```
(defrule positivo
        (valor ?val)
        (test (> ?val 0))
        =>
        (printout t ?val “es positivo" crlf) )
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Variables y Condiciones

Ejercicios:

- Escriba una regla que se active ante hechos del tipo *(valor a b)*
- Escriba una regla que se active con hechos del tipo *(valor a b)*, siempre que *b* sea mayor que *a*, circunstancia en la que se escribirán los valores de *a* y *b* en pantalla. Comprueba el funcionamiento con la siguiente base de hechos:

(valor 6 12)
(valor 6 7)
(valor 15 30)
(valor 14 7)

Primeros ejemplos: Entrada y salida

La función *read* permite a CLIPS leer información proporcionada por el usuario. El programa se detiene cuando, a la espera de que el usuario teclee el dato.

Puedes probar con *(assert (milectura (read)))* que tras leer el dato añade un hecho. Puedes comprobarlo con *(facts)*.

La siguiente regla pregunta por un dato si no está ya disponible en la base de hechos:

```
(defrule estan-luces-funcionando
    (not (luces-funcionando ?))
    =>
    (printout t "¿Están las luces del coche funcionando (S o N)?")
    (assert (luces-funcionando (read))))
```

Si no existe ya un hecho con el símbolo *luces-funcionando*, el sistema preguntará al usuario por el dato.

Primeros ejemplos: Entrada y salida

Podemos controlar que la respuesta no siga el formato esperado haciendo

```
(defrule r3
        ?hecho <- (luces-funcionando ?)
        (not (luces-funcionando S))
        (not (luces-funcionando N))
=>
      (printout t “Formato de respuesta incorrecto")
     (retract ?hecho))
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Entrada y salida

La función *read* lee sólo hasta el primer espacio en blanco. Si queremos introducir una serie de elementos, separados por espacios, debemos utilizar *readline*

```
(defrule obtener-nombre
    =>
    (printout t "Su nombre: ")
    (assert (nombre (readline))))
```

La función *readline* retorna una cadena. Evitamos que sea una cadena y se considere un conjunto de símbolos con *explode$*

```
(defrule obtener-nombre2
    =>
    (printout t "Su nombre: ")
    (assert (nombre (explode$(readline)))))
```

Primeros ejemplos: Entrada y salida

Ejercicios:

- Escribir una base de conocimiento **CLIPS** que lea la respuesta (positiva o negativa a una pregunta) y posteriormente escriba si la respuesta ha sido positiva o negativa.

- Escriba una base de conocimiento **CLIPS** que tras leer una secuencia de tres números enteros, escriba la secuencia si el tercer número es la suma de los dos primeros.

  Nota: Se recomienda usar la función *(readline)* para leer una cadena de entrada, y seguidamente convertirla en un multivalor mediante la función *(explode$)*

Primeros ejemplos: Entrada y salida

El comando *printout* permite realizar operaciones tanto de escritura por pantalla como a fichero

(printout <nombre-lógico> <expresión>)

stdout Nombre lógico de la salida por pantalla, suele sustituirse por *t*

CLIPS> (printout t “¡Hola!!" crlf)
¡Hola!

crlf al final establece un fin de línea

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Entrada y salida

La escritura a fichero con el comando *printout*, modifica el nombre lógico.

Previamente debe abrirse el fichero (*modo escritura w, lectura r, lectura y escritura r+, y apéndice a*). Recuerda indicar la ruta correctamente

```
(open "mifichero.txt" F1 "w“)  ;abre en modo escritura indicando w

(printout F1 “dato 23")        ; escribe datos en fichero

(close F1)                     ; cierra el fichero
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Entrada y salida

El comando *read* lee del fichero indicado un solo campo. Para leer una línea empleamos *readline*

```
(open "mifichero.txt" F1)      ; indicando r o nada abre en modo lectura

(read F1)                      ; lee hasta primer espacio

(read F1)                      ; lee hasta siguiente espacio

(close F1)                     ; cierra fichero
```

stdin Nombre lógico de la entrada por defecto empleada por *read* y *readline*.

Existen también comandos como rename y remove que permiten respectivamente renombrar y eliminar un fichero

Primeros ejemplos: Entrada y salida

Un ejemplo de lectura y escritura

```
CLIPS> (open "data.txt" misdatos "w")
TRUE
CLIPS> (printout misdatos “lunes martes")
CLIPS> (close)
TRUE
CLIPS> (open "data.txt" misdatos)
TRUE
CLIPS> (readline misdatos)
"lunes martes"
CLIPS> (readline misdatos)
EOF
CLIPS> (close)
TRUE
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Estructuras de control

Sentencia condicional

(if (<condición>)
then (<acciones>)
[**else** (<acciones>)] **)**

Sentencia repetitiva o bucle

(**loop-for-count** (<var> <inicio> <final>) [do] <acción>)

(while (<condición>) [do] (<acción>) **)**

Los elementos entre corchetes son opcionales

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Estructuras de control

```
(defrule continua-bucle
      ?bucle <- (bucle)
      =>
      (printout t "¿Continuar?" crlf)
      (bind ?resp (read))
      (if (or (eq ?resp si)(eq ?resp s))
      then
      (retract ?bucle)     ; Ante una respuesta positiva quitamos y
      (assert (bucle))     ; añadimos el hecho
      else
      (retract ?bucle)     ; No continuamos, simplemente eliminamos
      (halt)
      )  )
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Estructuras de control

Prueba los bucles con:

```
(loop-for-count 2 (printout t "Hola mundo" crlf))

(loop-for-count (?i 0 2) do
    (loop-for-count (?j 1 3) do
        (printout t ?i " " ?j crlf)))

(defrule rwhile
=>
(bind ? 4)
(while (> ?v 0)                          ; antes ?v debe tener un valor
    (printout t “v es " ?v “ " crlf)
    (bind ?v (- ?v 1))) )
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Estructuras de control

Ejercicios

- Realizar un programa CLIPS que tras leer un número indique si es positivo o negativo empleando la estructura condicional
- Realizar un programa en CLIPS que tras leer un número entero escriba todos los valores enteros entre 0 y ese número
- Tras almacenar un número en una variable global, se pide realizar un programa en CLIPS que permita al usuario introducir números enteros entre 1 y 100 hasta que adivine el número que el sistema tiene almacenado. La entrada de números se hará mediante la sentencia *read*. El programa indicará por pantalla si el número buscado es mayor o menor que el introducido. La condición de parada se produce cuando el usuario acierte el número en cuestión.

Nota: Para generar números aleatorios usa:

(random [<start-integer-expression> <end-integer-expression>])

(seed (round (time))) ;modifica la semilla

Primeros ejemplos: Operadores lógicos y matemáticos

Dos patrones en una regla se conectan de forma automática con un *and*, ya que requiere que ambos se cumplan para que la regla se dispare. Este ejemplo, visto anteriormente, incluye el uso del *not*

```
(defrule mamifero
        (animal ?nombre) (sangre-caliente ?name) (not (pone-huevos ?nombre))
        =>
        (assert (mamifero ?nombre))
        (printout t ?nombre " es un mamífero" crlf))
```

Nos faltaría ver un ejemplo de uso de *or*:

```
(defrule coge-paraguas
    (or (tiempo lloviendo) (tiempo nevando)) => (assert (paraguas necesario)))
```

Que traducimos por “Si está lloviendo o nevando, coge el paraguas”, haciendo uso de sólo una regla ya que también podrían utilizarse dos independientes con el mismo efecto.

Primeros ejemplos: Operadores lógicos y matemáticos

Observa que el operador *or* está colocado antes de los argumentos y no entre ellos, es la denominada notación prefijo (*prefix notation)* y la utilizan todos los operadores en CLIPS.

Para por ejemplo realizar una suma, la mayoría de los lenguajes, usarían 5 + 7 (*infix notation*), mientras que CLIPS requiere (+ 5 7).

Algunos ejemplos:

```
CLIPS>(+ 5 7)
12
CLIPS>(- 5 7)
-2
CLIPS>(* 5 7)
35
CLIPS>(/ 5 7)
0.7142857142857143
```

Expresiones más complicadas como 10+4*19-35/12:

```
CLIPS> (+ 10 (- (* 4 19) (/ 35 12)))
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Funciones

Una función es un algoritmo identificado con un nombre que puede o no devolver valores (uni o multicampo). Se definen con el constructor *deffunction*, existiendo dos tipos:

- Funciones internas: Definidas en CLIPS
- Funciones externas: Escritas en un lenguaje distinto a CLIPS

```
(deffunction signo (?num)
   (if (> ?num 0) then (return 1))
   (if (< ?num 0) then (return -1))
   0)
```

Las funciones se llaman usando la notación prefija entre paréntesis como ya hemos visto con los operadores aritméticos y lógicos:

```
CLIPS> (signo 3)
1
```

El comando *list-deffunctions* permite listar las funciones definidas

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Funciones

Cuando una función se llama a sí misma hablaremos de recursividad:

$$\sum_{i=1}^{n} i = n + \sum_{i=1}^{n-1} i$$

```
(deffunction sumatorio (?num)
    ( if (> ?num 0) then (return (+ ?num (sumatorio (- ?num 1))))
      else 0) )

CLIPS> (sumatorio 5)
15
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Elementos básicos: Funciones

Ejercicios

- Escribe una función que calcule el factorial de un número

- Escribe función en **CLIPS** para calcular y mostrar por pantalla el máximo común divisor (*mcd*) de dos números obtenidos a través del teclado, asumiendo que son positivos. El *mcd* de estos números se calcula teniendo en cuenta:

  - *Si* $a = b$ *mcd*(a, a) = a
  - Si $a > b$ *mcd*(a, b) = *mcd*($a-b$, b)
  - *Si* $a < b$ *mcd*(a, b) = *mcd*(a, $b-a$)

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Elementos básicos: Funciones

Ejercicios

- El *mínimo común múltiplo* (*mcm*) de dos números enteros *a* y *b* se puede calcular a partir del máximo común divisor (*mcd*) de dichos números tal que mcm(a,b)=a*b /(mcd(a,b)). Escribe una función en **CLIPS** para calcular el *mcm* de dos números enteros que siga los siguientes pasos:

  - Inicialmente se leen dos números *a* y *b* del teclado
  - Seguidamente se calcula el *mcd* de estos números
  - Finalmente calcula el *mcm* y se muestra el resultado en pantalla

Primeros ejemplos: Restricciones

Se pueden especificar restricciones al comparar un patrón (el orden de prioridad es el presentado a continuación):

- negación (~) (color ~rojo)
- conjunción (&) (color rojo&amarillo)
- disyunción (|) (color rojo|amarillo)

Algunos ejemplos considerando un cruce de peatones con semáforo:

```
(defrule no-cruzar
        (luz ~verde)
        =>
        (printout t “No cruce” crlf)

(defrule precaucion
        (luz amarilla|intermitente)
        =>
        (printout t “Cruce con precaución” crlf)

(defrule puedo-cruzar
        (luz ~rojo&~amarillo)
        =>
        (printout t "Puede cruzar“ crlf))
```

Primeros ejemplos: Restricciones

Dado el símbolo de un hecho, se pueden establecer restricciones sobre el valor retornado en la llamada a una función u operación. Se denota por **=**. El valor devuelto por la función debe ser uno de los tipos de datos primitivos

```
(defrule Regla1
      (dato ?x)
      (persona (edad =(* 2 ?x) ) )    ; el dato de la edad debe coincidir
      =>
      (printout t "Encontrada persona con edad igual al doble de "
      ?x crlf))

(defrule Regla2
      (dato ?x)
      (persona (edad ?y&=(* 2 ?x)|=(+ 2 ?x) ) )  ; guardamos en variable
      =>
      (printout t "Encontrada persona con edad " ?y " que es el
      doble de " ?x " o dos años mas que " ?x
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Restricciones

También se puede restringir un campo basándose en la veracidad de una expresión booleana. Se denota por el símbolo ":". *?x&:<restricción-sobre-?x>* se lee: determina el valor *?x* tal que *?x* verifica la restricción *<restricción-sobre-?x>*

```
(defrule Regla1
      (valor ?x&:(> ?x 0))
      =>
      ) ;?x tal que es mayor que 0

(defrule Regla2
      (Hecho (valor ?x&:(integerp ?x)|:(floatp ?x) ) )
      =>
      )   ; si el valor del campo de un hecho es un número
```

Primeros ejemplos: Comodines y variables

Hemos visto el uso del comodín **?** para sustituir a un símbolo en la parte izquierda (la condición) de una regla. Supongamos que tenemos los hechos:

(miembro-de beatles john_lennon paul_mccartney george_harrison ringo_starr)
(miembro-de who roger_daltrey pete_townsend keith_moon)
(miembro-de ebtg tracey_thorn ben_watt)

Una regla que se active ante cualquiera de estos hechos no es posible utilizando **?** al requerir la presencia de un único símbolo. Así (a ?x ?y) encaja con cada hecho de tres elementos comenzando con "a" ,p.e. (a b c), (a 1 2), (a a a) …, pero no si tienen más.

Para dar esa posibilidad utilizamos **$?** que equivale a ninguno o más símbolos, es decir variables multicampo. La desventaja es que aumenta el coste de procesamiento, por lo que no se debe abusar en su uso.

Primeros ejemplos: Comodines y variables

En el ejemplo de las bandas musicales la regla bandas podría aceptar un número no determinado a priori de miembros:

```
(defrule bandas
    (miembro-de ?banda $?)
    =>
    (printout t “Hay una banda llamada " ?banda crlf))
```

Primeros ejemplos: Comodines y variables

Podemos incluso obtener la lista de miembros de cada banda

```
(defrule miembros-banda
      (miembro-de ?banda $? ?miembro $?)
      =>
      (printout t ?miembro " es miembro de " ?banda crlf))
```

Observa que **$?** encaja con ninguno o más símbolos mientras que ?miembro sólo con uno (una variable puede referirse a múltiples campos p.e. $?valores)

Podemos guardar en una variable:

```
(defrule miembros-banda2
      (miembro-de ?banda $?miembros)
      =>
      (printout t “Los miembros de " ?banda " son " $?miembros crlf))
```

Primeros ejemplos: Comodines y variables

Ejercicios:

- Escribe una regla que se active con todos estos hechos

(dato AZUL amarillo rojo verde)
(dato AZUL rojo)
(dato rojo AZUL)
(dato AZUL)

Primeros ejemplos: Plantillas y condiciones

Los hechos de una plantilla, hecho no ordenado, pueden emplearse en una regla del mismo modo que uno ordinario. El siguiente ejemplo escribe el nombre y la edad:

```
(defrule edades
        (datos-personales (nombre ?nombre) (edad ?edad))
         =>
        (printout t ?nombre " tiene" ?edad " años." crlf) )
```

Combinado con operadores lógicos:

```
(defrule edades-peso
        (and
        (datos-personales (nombre ?nombre) (edad ?edad))
        (datos-personales (nombre ?nombre) (peso ?peso)) )
        =>
        (printout t ?nombre " pesa " ?peso " con " ?edad " años." crlf) )
```

Primeros ejemplos: Plantillas y condiciones

Como ya sabes, *test* permite especificar un elemento condicional basado en una expresión, en este caso, personas con peso superior a 100 kilos.

```
(defrule sobre-cien
    (datos-personales (nombre ?nombre) (peso ?peso))
    (test (> ?peso 100))
    =>
    (printout t ?nombre " pesa " ?peso " kg." crlf) )
```

Primeros ejemplos: Plantillas y condiciones

Recordemos la función *modify* que permite *modificar* un campo o *slot* de un hecho sin tener que emplear *retract* y *assert* para ello. Puede usarse en una regla.

Si no queremos eliminar el hecho anterior, sino copiar sus valores (quizás modificando alguno de sus campos) utilizaremos *duplicate*

```
(defrule cumpleanos
    ?cumpleanos <- (cumpleanos ?nombre)
    ?data-fact <- (datos-personales (nombre ?nombre) (edad ?edad) )
    =>
    (modify ?data-fact (edad (+ ? edad 1)))
    (retract ?cumpleanos) )
```

Al añadir el hecho *(cumpleanos Samuel)* se modifica su edad sin alterar el resto de sus registros. Se elimina finalmente el hecho para evitar que la regla pueda volver a activarse (p.e. si se modificara otra parte de la plantilla)

Primeros ejemplos: Plantillas y condiciones

Otras posibilidades para establecer condiciones, además de *test* ya mencionado previamente, son *exists* and *forall*. El primero se satisface cuando uno o más hechos verifican el predicado.

```
(defrule persona-existe
        (datos-personales (nombre ?nombre))
        =>
        (printout t “Hay una persona llamada " ?nombre crlf) )

(defrule hay-personas
        (exists (datos-personales (nombre ?nombre)))
        =>
        (printout t “Al menos hay una persona" crlf) )) )
```

Primeros ejemplos: Plantillas y condiciones

forall, se satisface si todos los hechos cumplen el patrón, por ejemplo:

```
(defrule prueba-cada-persona
(forall (datos-personales (nombre ?nombre)))
=>
(printout t “Todas las personas tienen nombre" crlf) )
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: El coche no arranca

El siguiente SE para actuar como experto (=conductor habitual) cuando no se consigue arrancar un coche por la mañana.

Se considera un conjunto de símbolos, por legibilidad utilizaremos formalmente un alias:

- A=[Coche-no-arranca]
- B=[Posible-fallo-eléctrico]
- C=[Carga-batería-<-10-volts]
- D=[Batería-baja-de-carga]

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: El coche no arranca

Cuando ocurre el problema, existe una base de hechos iniciales:

A=[Coche-no-arranca]

Las reglas consideradas son dos:

Regla 1: Si [Coche-no-arranca]
entonces [Posible-fallo-eléctrico]
Regla 2: Si [Posible-fallo-eléctrico] Y [Carga-batería-<-10-volts]
entonces [Batería-baja-de-carga]

Que de forma sucinta expresaremos:

Regla 1: Si A entonces B
Regla 2: Si B Y C entonces D

Primeros ejemplos: El coche no arranca

```
(deffacts hecho-inicial
        (Coche_No_Arranca)
)

(defrule primera
        (Coche_No_Arranca)
        =>
        (assert (Posible_Problema_Eléctrico))
        (printout t "Tensión Batería" crlf)
        (assert (tensión (read))) ; Solicita al usuario información respecto a la tensión de batería

(defrule  segunda
        (Posible_Problema_Eléctrico)
        (tensión ?x&: (< ?x 10))
        =>
        (printout t "Batería Baja" crlf)
        (assert ( La_Bateria_Está_Baja))    )
```

Constructor y nombre de hechos iniciales

Hecho inicial

Constructor y nombre de la primera regla

entonces

Premisa equivalente a Si A

Para mostrar información por la pantalla

Conclusión

Insertar un hecho en base

Lectura teclado

Comentario

Constructor y nombre de la segunda regla

entonces

Premisas equivalentes a Si A Y C

Conclusión

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Vamos al teatro

El siguiente SE determina el medio de transporte para acudir al teatro en base a la distancia, las condiciones meteorológicas, la localización del teatro y el tiempo disponible hasta que comience la función.

```
defrule Distancia
=>
(printout t "Distancia al teatro (en kilómetros)" crlf)
(assert (distancia (read)))
)

(defrule Tiempo_comienzo
=>
(printout t "Tiempo disponible hasta el comienzo de la función (en minutos)" crlf)
(assert (tiempo (read)))
)
```

Primeros ejemplos: Vamos al teatro

```
(defrule Meteo
(consulta_meteo)
=>
(printout t "¿Cómo está el tiempo (bueno o malo)?" crlf)
(assert (meteo (read)))
)

(defrule Lugar
(pregunta_loc)
=>
(printout t "¿Dónde es la función (centro o afueras)?" crlf)
(assert (loc (read)))
)

(defrule R1
(distancia ?dist)
(test (> ?dist 2))                    ; Está a más de 2 kilómetros
=>
(assert (transporte coche))
(assert (pregunta_loc))
)
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Vamos al teatro

```
(defrule R2
(distancia ?dist)
(tiempo ?temp)
(test (> ?dist 1))                    ; Está a más de 1 kilómetro
(test (< ?temp 10))                   ; Tenemos menos de 10  minutos
=>
(assert (transporte coche))
(assert (pregunta_loc))
)

(defrule R3
(distancia ?dist)
(tiempo ?temp)
(test (> ?dist 1))                    ; Está a más de 1 kilómetro
(test (< ?dist 2))                    ; Está a menos de 2 kilómetros
(test (> ?temp 10))                   ; Tenemos más de 10  minutos
=>
(assert (transporte a_pie))
)
```

Primeros ejemplos: Vamos al teatro

```
(defrule R4
(distancia ?dist)
(tiempo ?temp)
(test (< ?dist 1.5))                ; Está a menos de 1.5 kilómetros
(test (> ?temp 10))                 ; Tenemos más de 10  minutos
=>
(assert (transporte a_pie))
(assert (consulta_meteo))
)

(defrule R5
(transporte coche)
(loc centro)
=>
(printout t "Toma un taxi al centro" crlf)
(halt)
)
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Vamos al teatro

```
(defrule R6
(transporte coche)
(loc ~centro)
=>
(printout t "Toma tu coche" crlf)
(halt)  )

(defrule R7
(transporte a_pie)
(meteo bueno)
=>
(printout t "Ve a pie" crlf)
(halt)  )

(defrule R8
(transporte a_pie)
(meteo ~bueno)
=>
(printout t "Ve a pie, pero no olvides el paraguas" crlf)
(halt)  )
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Logs

Para la depuración de programas, es posible volcar la salida, incluyendo los errores, a un fichero. Esta característica se activa con *(dribble-on <nombre-fichero>)* siendo posible su desactivación con *(dribble-off)*

INSTITUTO UNIVERSITARIO SIANI INGENIERIA COMPUTACIONAL

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Primeros ejemplos: Consistencia

Usaremos el fichero *riesgo.clp*, tras abrirlo en el entorno y cargarlo *(load "riesgo.clp")* debemos usar *(reset)* para que se carguen los elementos por defecto

```
(deftemplate datos-personales
    (slot nombre) (slot edad)
    (slot peso) (slot fumador)
    (multislot fecha-nacimiento) )

(deffacts gente
    (datos-personales (nombre angel) (peso 60) (edad 30)
     (fumador no) (fecha-nacimiento 18 06 1970))
    (datos-personales (nombre belinda) (peso 120) (edad 45)
     (fumador si) (fecha-nacimiento 18 06 1955))
    (datos-personales (nombre carlos) (peso 120) (edad 60)
     (fumador si)(fecha-nacimiento 18 06 1940)) )

(deffacts datos
    (fecha 08 10 2008) )
```

Primeros ejemplos: Consistencia

Con la siguiente regla comprobamos si una persona tiene riesgo cardíaco (fumador o sobrepeso):

```
(defrule riesgo-cardiaco
    (datos-personales (nombre ?nombre) (fumador si)
    (peso ?peso)) (test (> ?peso 100))
    =>
    (assert (riesgo-cardiaco ?nombre)) )
```

Tras ejecutar *(run)*, por medio de *(facts)* o en la ventana de Facts (Windows->Fact) aparecen dos hechos nuevos:

```
f-5     (riesgo-cardiaco carlos)
f-6     (riesgo-cardiaco belinda)
```

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Consistencia

El problema con esa regla es que si una de esas personas dejara de fumar o perdiera peso, el hecho añadido seguirá ahí, a pesar de no ser consistente con la realidad dado que el hecho no almacena información sobre la regla que lo produjo.

Para evitar este efecto se emplea el elemento *logical*.

```
(defrule riesgo-cardiaco
      (logical (datos-personales (nombre ?nombre) (fumador si) (peso
         ?peso)))
      (logical (test (> ?peso 100)))
      =>
      (assert (riesgo-cardiaco ?nombre)) )
```

Primeros ejemplos: Consistencia

Al ejecutar la regla los resultados son idénticos, es decir, se añaden dos hechos nuevos. Pero si modificamos los datos de uno de los sujetos, p.e.

```
CLIPS> (modify 2 (peso 80))
```

(2 se corresponde con los datos de Belinda), observarás que los hechos se modifican, y el hecho *(riesgo-cardiaco belinda)* desaparece.

El elemento *logical* ha creado un enlace entre la regla y los hechos que provoca. Pero ojo, este elemento requiere más memoria y cómputo, por lo que debe controlarse su uso.

Primeros ejemplos: Resolución de conflictos

Cuando se activa una regla, ésta se coloca en la agenda según los siguientes criterios:

1. Las reglas más recientemente activadas se colocan encima de las reglas con menor prioridad, y debajo de las de mayor prioridad
2. Entre reglas de la misma prioridad, se emplea la estrategia configurada de resolución de conflictos
3. Si varias reglas son activadas por la aserción de los mismos hechos, y no se puede determinar su orden en la agenda según los criterios anteriores, se insertan de forma arbitraria (no aleatoria)

Primeros ejemplos: Resolución de conflictos

Estrategias de resolución de conflictos. Si un hecho A activa las reglas r1 y r2, y otro B activa las reglas r3 y r4:

- Estrategia en profundidad (depth): Es la estrategia por defecto, añadiendo r3, r4, r1, r2
- Estrategia en anchura (breadth): Añade r1, r2, r3, r4
- Estrategia de simplicidad/complejidad (complexity/simplicity): Según el número de comparaciones a realizar en el antecendente de la regla
- Estrategia aleatoria (random).
- Estrategia LEX (lex): Se da mayor prioridad a las reglas con un hecho más reciente, comparando los patrones en orden descendente
- Estrategia MEA (mea): Se aplica la misma estrategia de LEX, pero mirando sólo el primer patrón

Se activan con el comando *(set-strategy <estrategia>)*, *(get-strategy)* obtiene la actual

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Resolución de conflictos

El orden en que se ejecutan las reglas en CLIPS no es fácilmente controlable. Sin embargo la propiedad *salience* permite definir un valor entre -10000 y 10000, siendo por defecto 0. Las reglas con un valor mayor de la propiedad se disparan antes que aquellas con el valor menor.

```
(defrule molesta-fumadores
        (datos-personales (nombre ?nombre) (fumador si))
        => (printout t ?nombre " es un loco." crlf) )

(defrule preocupa-por-delgados
        (datos-personales (nombre ?nombre) (peso ?peso)) (test (< ?peso 80))
        ⇒ (printout t ?nombre " parece algo delgado." crlf) )
```

En este caso, el orden en que se disparan dependerá del orden en que los hechos de los que dependen fueron creados.

Primeros ejemplos: Resolución de conflictos

Por el contrario si utilizamos:

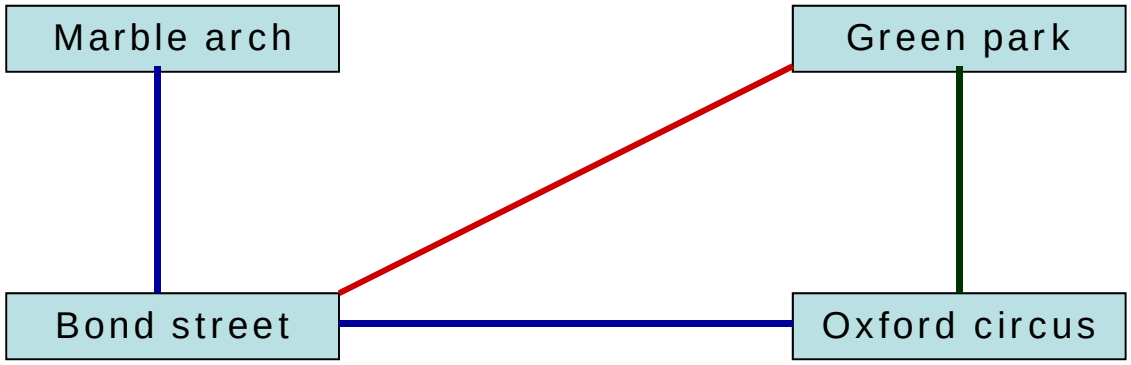
```
(defrule molesta-fumadores (declare (salience 10))
    (datos-personales (nombre ?nombre) (fumador si))
    => (printout t ?nombre " es un loco." crlf) )

(defrule preocupa-por-delgados (declare (salience 20))
    (datos-personales (nombre ?nombre) (peso ?peso)) (test (< ?peso
      80))
    ⇒ (printout t ?nombre " parece algo delgado." crlf) )
```

La regla preocupa-por-delgados tendrá prioridad. Se recomienda utilizar esta propiedad de forma comedida.

Primeros ejemplos: Resolución de conflictos

Abre el fichero metro.clp proporcionado por CLIPS Labs. Permite para una pequeña red de líneas (central, victoria y jubilee) de metro, preguntar la ruta entre dos estaciones.

De forma explícita mediante hechos iniciales se definen las estaciones que une una línea, las conexiones, así como las líneas que parten de una estación (plataforma).



Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Resolución de conflictos

Tras definir una estación de partida y una de llegada, se van marcando de forma sistemática las estaciones a partir de la salida. Luego se marca cada estación conectada con una ya marcada hasta alcanzar el destino. Un viajero en una estación se representa:

(location station marble-arch)

Un viajero en el andén de una línea en una estación

(location platform oxford-street central)

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Primeros ejemplos: Resolución de conflictos

Una vez cargado el fichero tras activar los hechos *(watch facts)*, ejecutar *(reset)* y *(run)* se puede observar que a pesar de alcanzar *bond-street*, el programa no se detiene.

Con el hecho *(location station bond-street)* se activan las reglas *go-to-platform y found-it* colocándose ambas en la agenda. Sin embargo no hay forma de saber la que se ejecutará primero.

Con el comando *(watch activations)* observarás que *found-it* se activa pero algo antes de ser disparada.

Primeros ejemplos: Resolución de conflictos

Si varias reglas se activan de forma simultánea el orden en que se añaden a la agenda depende de su saliencia, por defecto todas tienen 0. Redefiniendo

```
(defrule found-it
    (declare (salience 100))
    (end-station ?station)
    (location station ?station)
    =>
    (printout t "FOUND IT!" crlf) (halt))
```

Prueba de nuevo con *reset* y *run*

Ejemplos: auto.clp

Usaremos el fichero *auto.clp*.

El código está separado en varios grupos de reglas:

- Reglas de estado
- Preguntas
- Reglas de sugerencias
- Reglas de reparaciones
- Control de fase

Sugiero cargar, ejecutar e intentar en varios supuestos comprender el flujo del programa

Ejemplos: wine.clp

Usaremos el fichero *wine.clp*.

Al cargar el código observaremos que un par de reglas provocan errores, pero sin embargo el programa se ejecuta sin problemas aparentes. En esta ocasión el código se separa en:

- Selección de cualidades
- Selección de vino
- Preguntas
- Presentación
- Reglas de eliminación
- Reglas de combinación
- Control de fase

Sugiero cargar, ejecutar e intentar en varios supuestos.

Ejemplos: Consulta médica

Descargar desde consulta médica

Las siguientes transparencias presentan ejemplos de cursos anteriores que pueden descargarse desde el campus virtual.

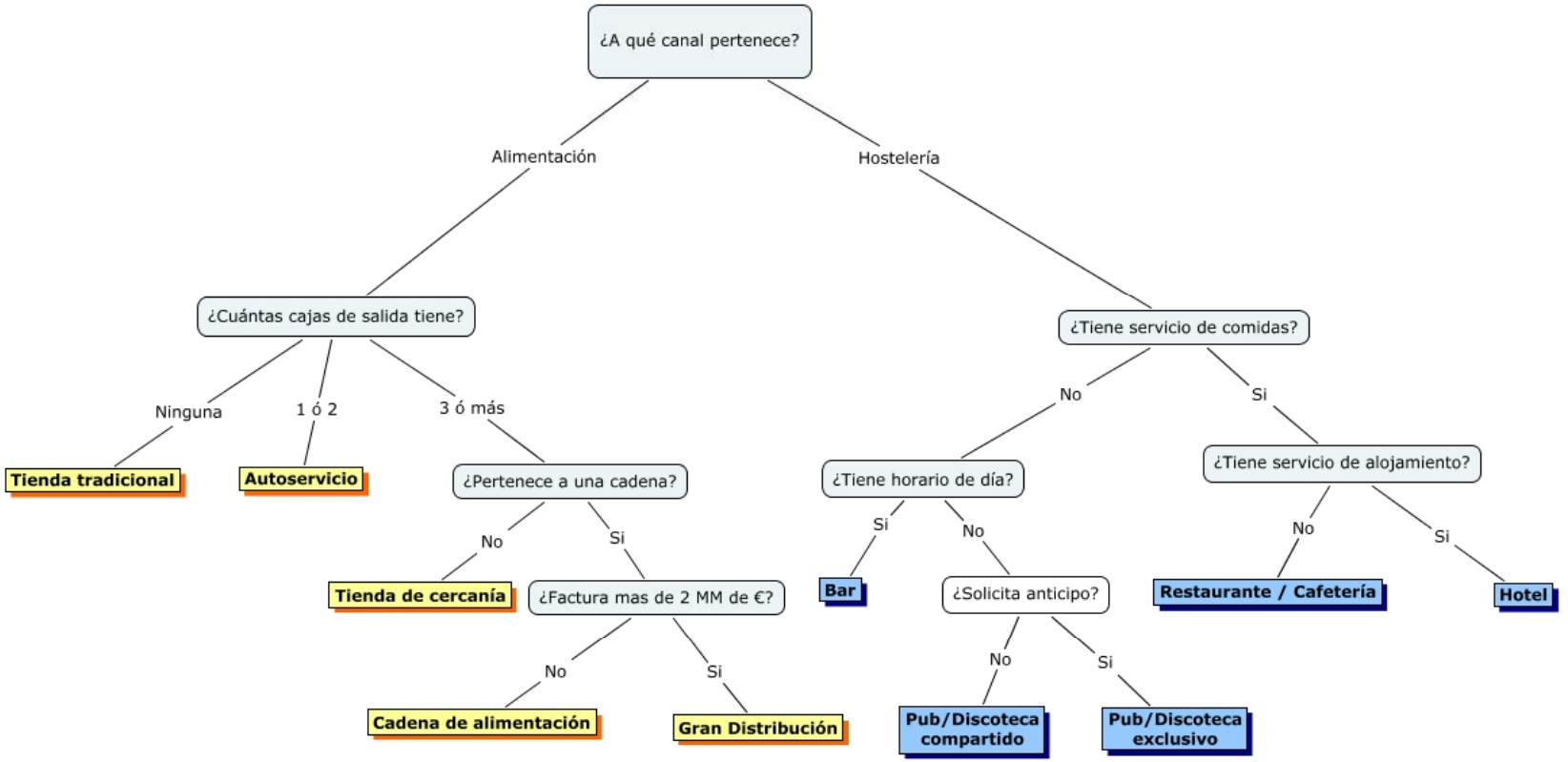
INSTITUTO UNIVERSITARIO SIANI
INGENIERIA COMPUTACIONAL

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Ejemplos: Curso 09/10

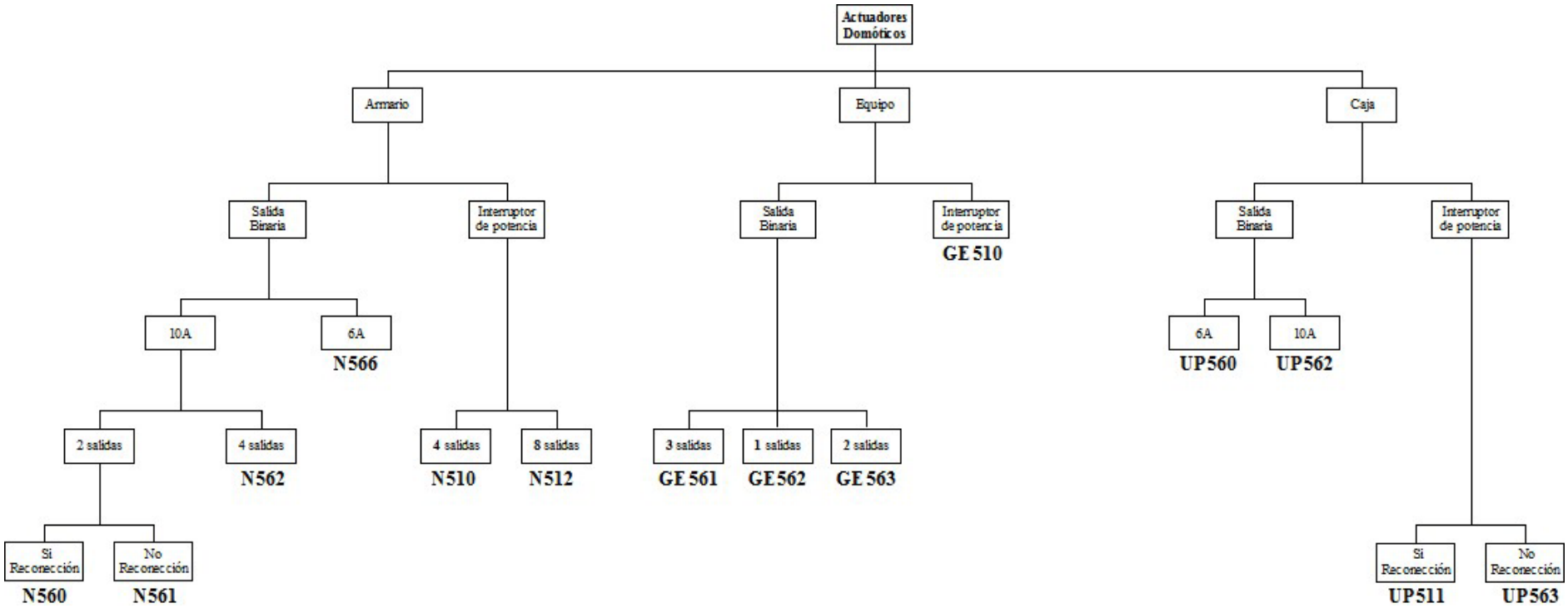
Alejandro Talavera



Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Ejemplos: Curso 09/10

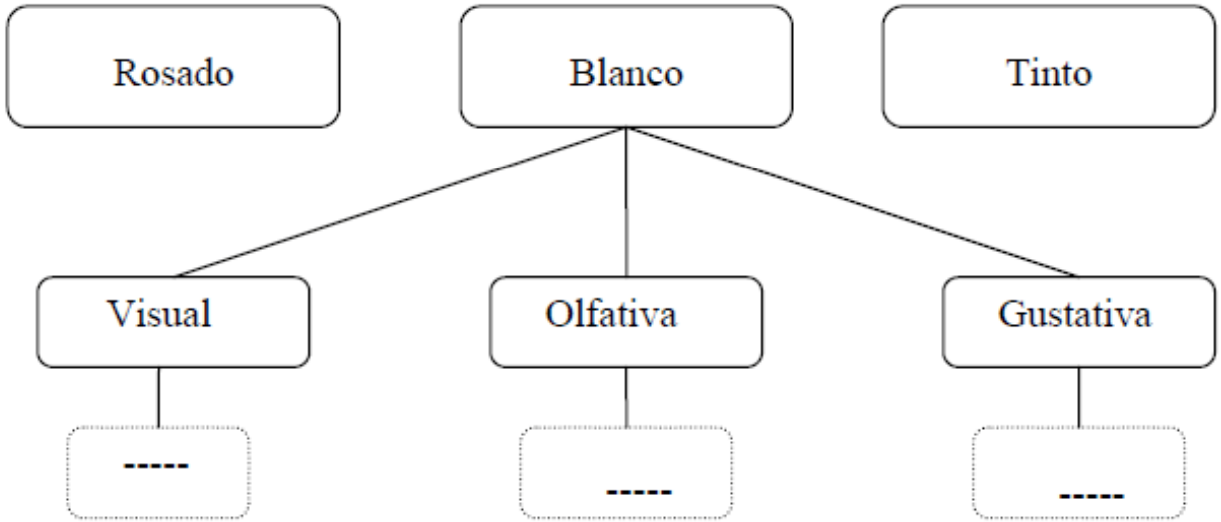
José Cristo González



Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Ejemplos: Curso 09/10

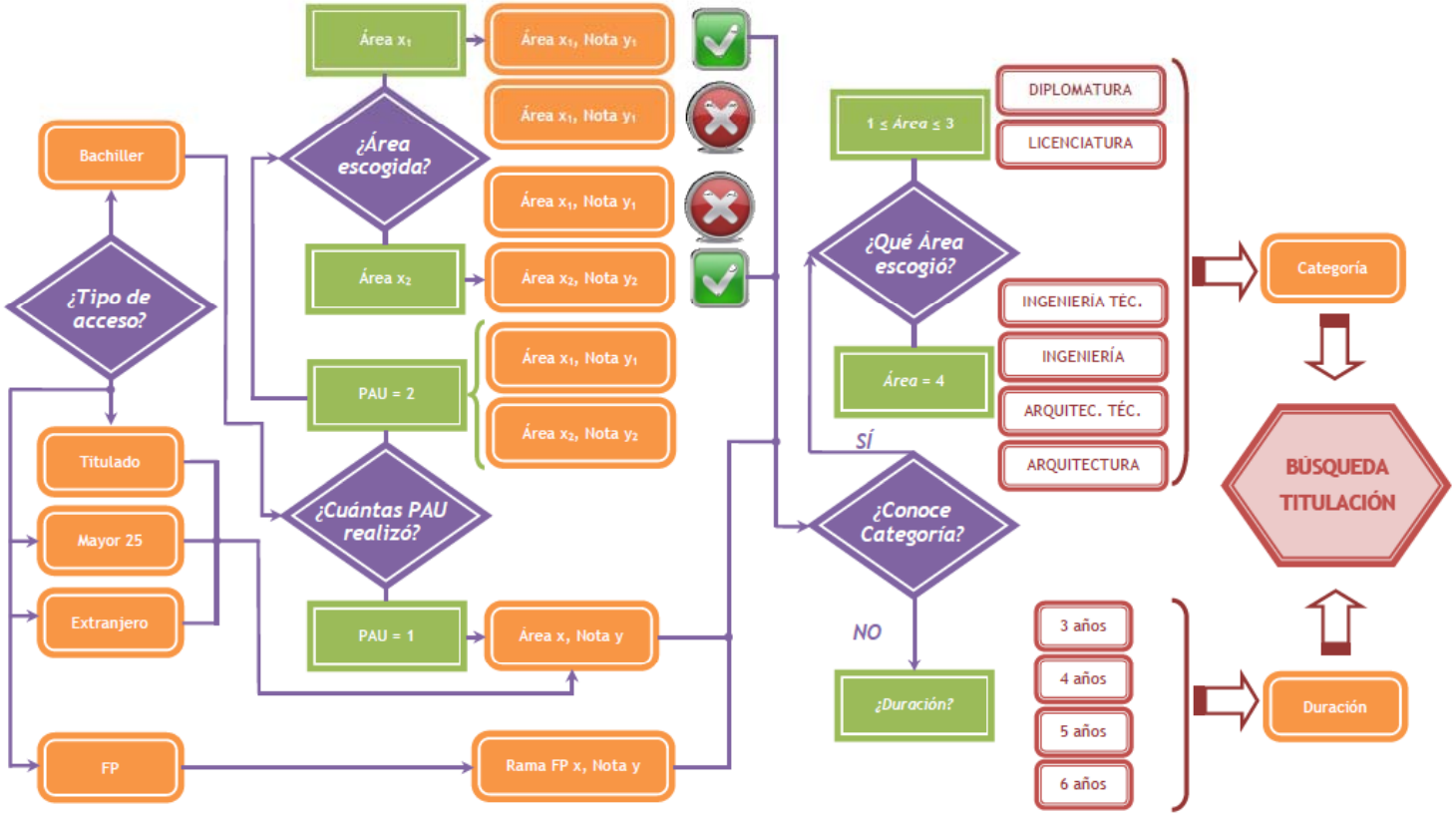
Desiré Santana



Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Ejemplos: Curso 09/10

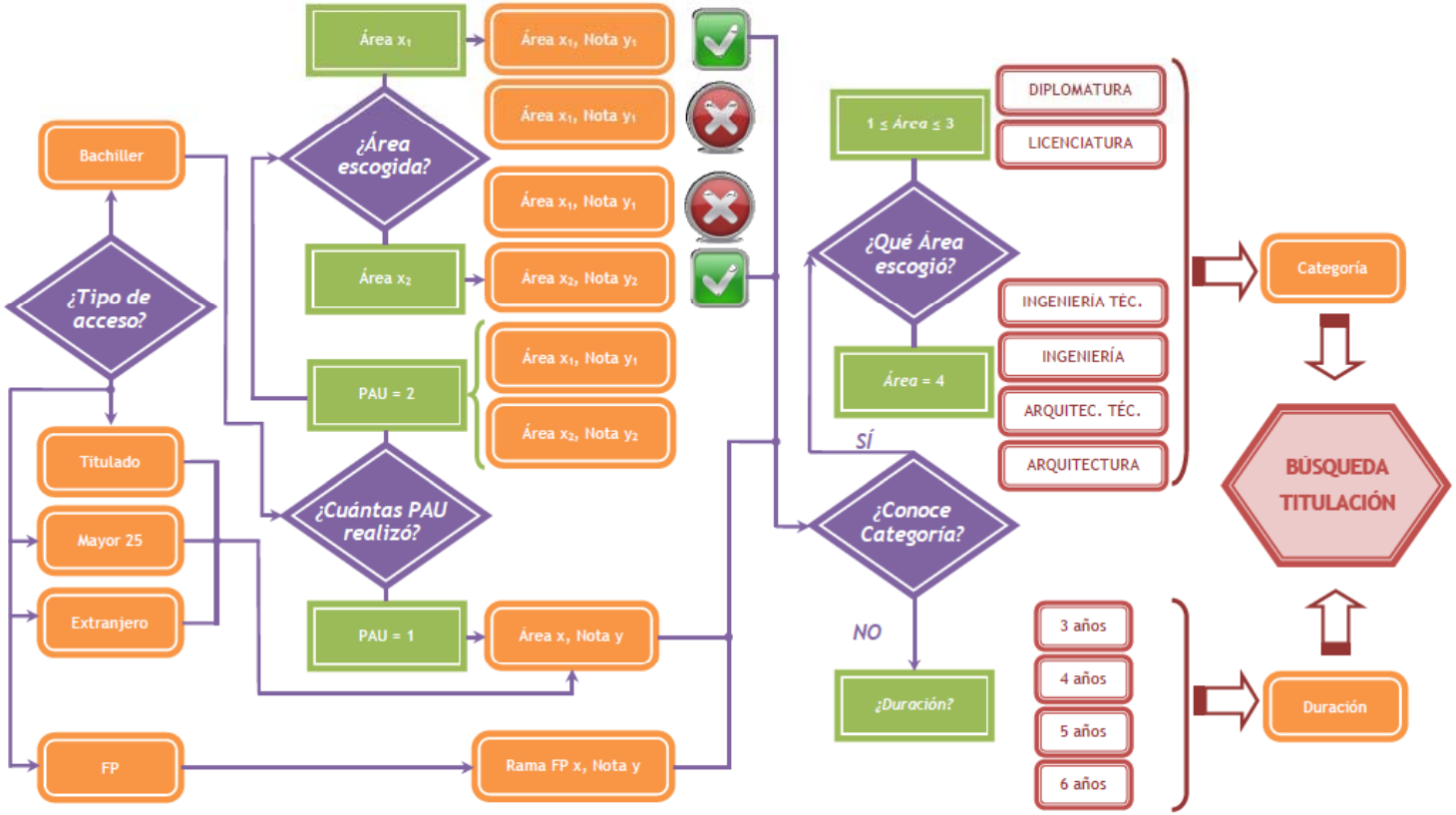
Rayco Toledo



INSTITUTO UNIVERSITARIO SIANI INGENIERIA COMPUTACIONAL
Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es
UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Ejemplos: Curso 09/10

Rayco Toledo

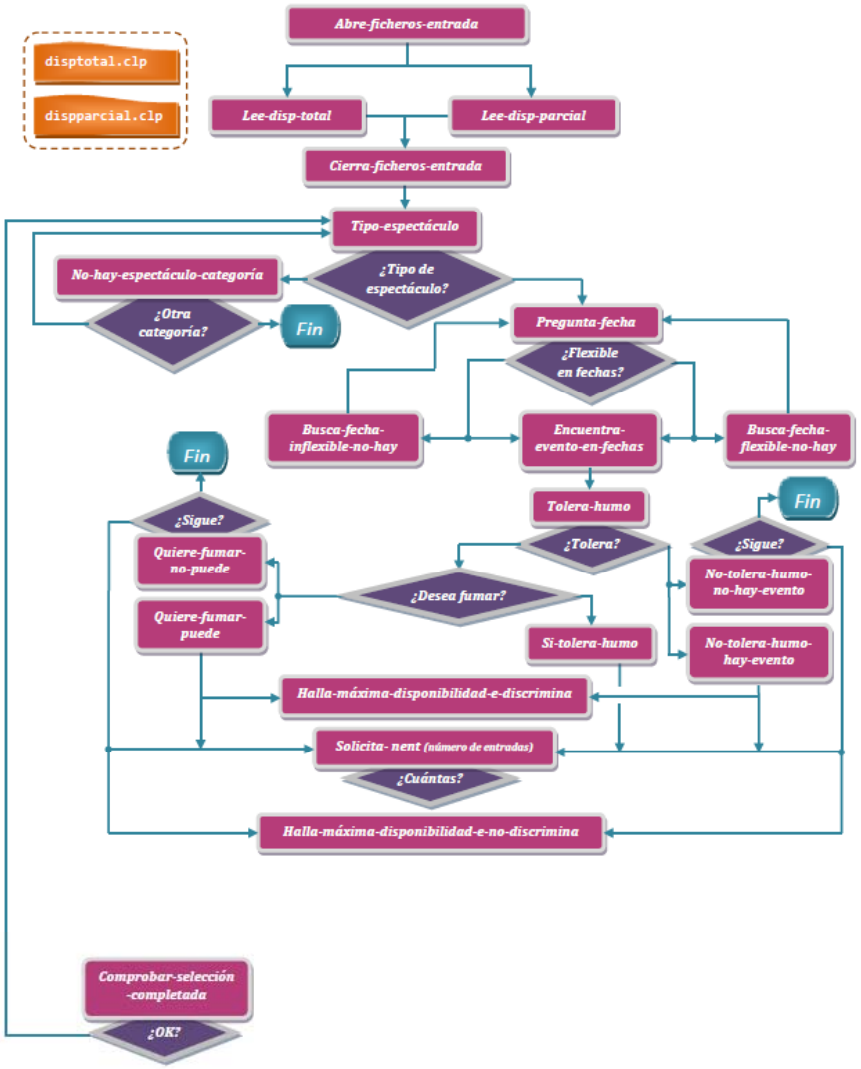


INSTITUTO UNIVERSITARIO SIANI INGENIERIA COMPUTACIONAL
Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es
UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

INSTITUTO UNIVERSITARIO SIANI
INGENIERIA COMPUTACIONAL

Ejemplos: Curso 09/10

Cristina Medina

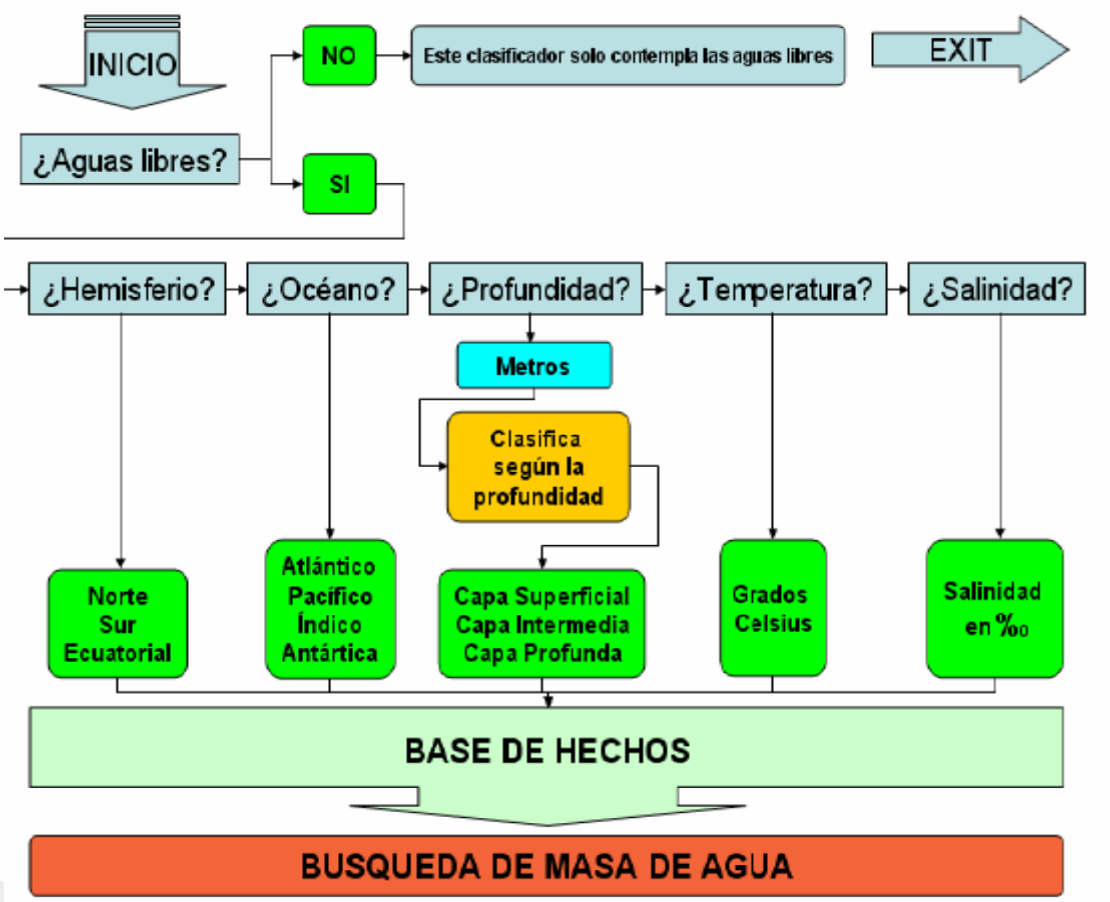


Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Ejemplos: Curso 10/11

Ninoska Cabrera: *Identificación de masas de agua oceánicas*.

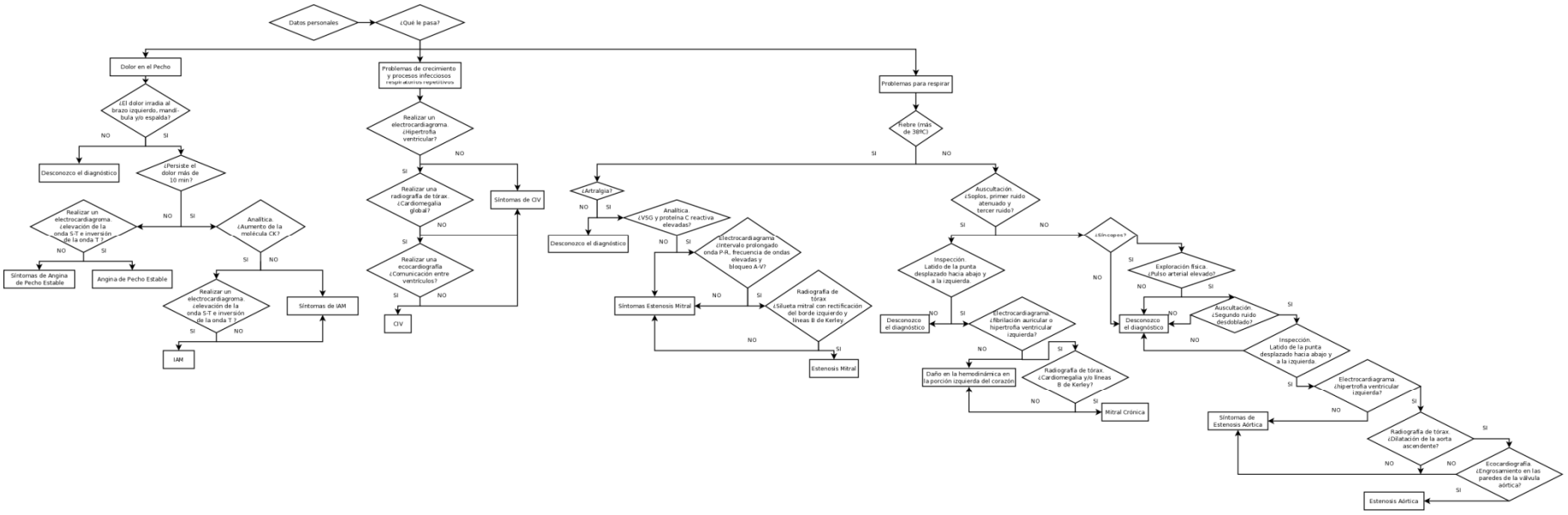


Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

INSTITUTO UNIVERSITARIO SIANI
INGENIERIA COMPUTACIONAL

Ejemplos: Curso 10/11

Moisés Díaz: Enfermedades cardíacas

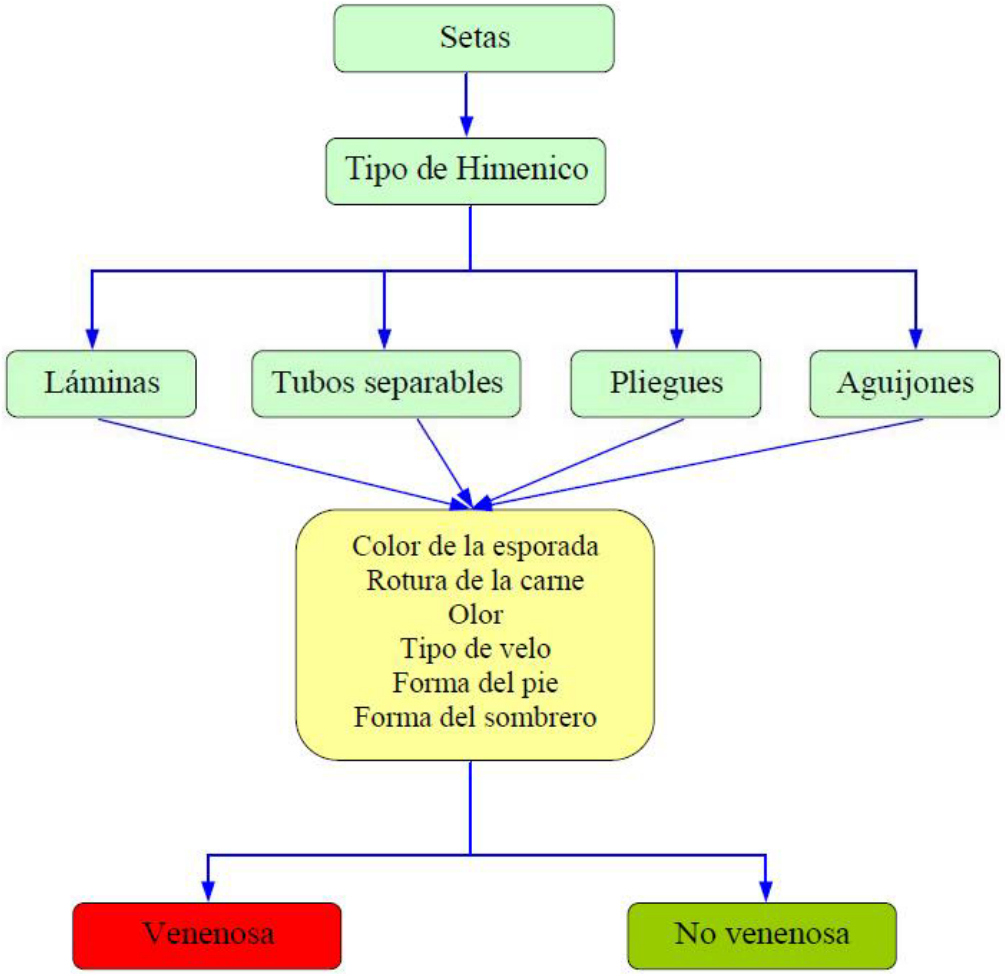


Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Ejemplos: Curso 10/11

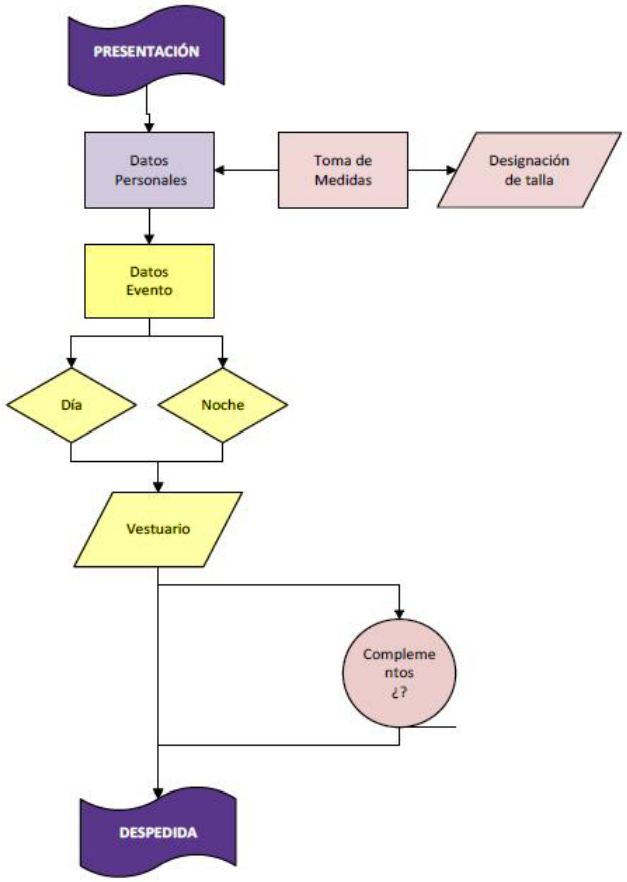
Rocío Espino: Clasificación de setas



INSTITUTO UNIVERSITARIO SIANI INGENIERIA COMPUTACIONAL
Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es
UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Ejemplos: Curso 10/11

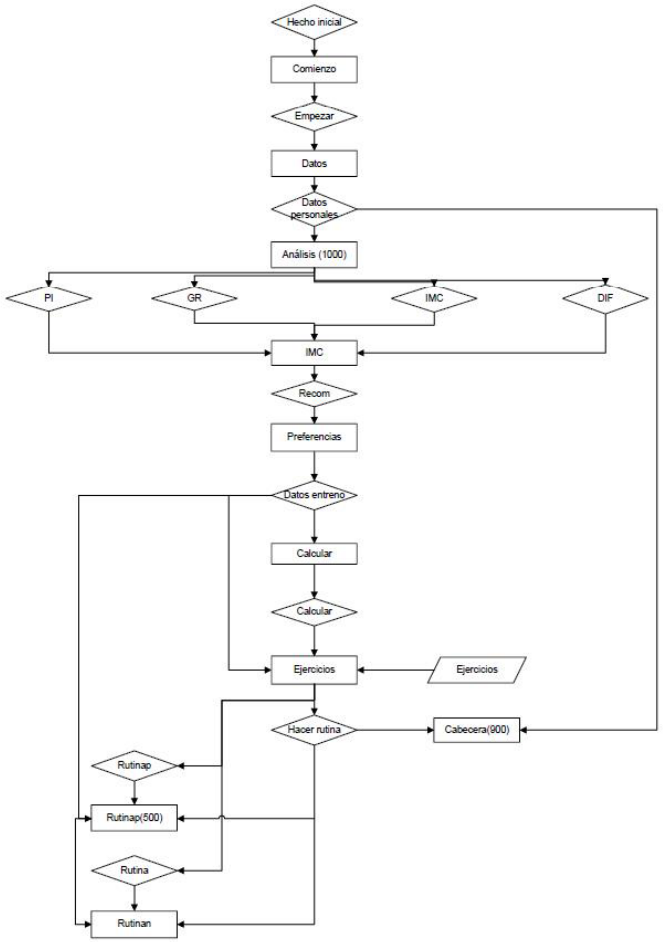
Cristina Freire: Estilista femenino



Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Ejemplos: Curso 10/11

Santiago Padrón: Entrenamiento personal



Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Ejemplos: Curso 10/11

José María Zarzalejos: Gestión del triaje enfermeril de embarazadas

INSTITUTO UNIVERSITARIO SIANI
INGENIERIA COMPUTACIONAL

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Trabajo de curso

Realizar un sistema basado en reglas

Incluir en la memoria:

- Descripción del sistema
- Una descripción esquemática o gráfica del conocimiento integrado en el sistema basado en reglas diseñado
- Significado de las principales reglas definidas
- Ejemplos de uso o ejecución el sistema

Fechas:

Definición temática del trabajo 18 Noviembre

Fecha tope de entrega 13 Enero

Fecha tope entrega revisión 27 de Enero

Edificio Central del Parque Tecnológico
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: info@siani.es · www.siani.es

Referencias/Documentación

- Tutorial CLIPS por Peter Robinson, Univ. de Hull, Reino Unido.
- Sistema CLIPS, por Aitor San Juan Sánchez
- Documentación CLIPS
- Introducción en español a CLIPS.
- Introducción a CLIPS, UNED
- Sistemas expertos con CLIPS.
- Introducción a CLIPS, Revisión de CLIPS, Universidad de Córdoba
- CLIPS Labs
- U1430: Expert Systems. Part I -- Programming in Expert Systems

- Ejemplo consulta médica