PROLOG

M. en C. Arturo Rodríguez García

1 INTRODUCCIÓN

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

PROLOG

- El lenguaje fue ideado por Alain Colmerauer y Philippe Roussel a principios de la década de 1970, en la Universidad de Provenza (Marsella, Francia).
- Tiene sus raíces en lógica de primer orden.
- Es un lenguaje declarativo.

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

PROLOG

- Es un lenguaje popular en la comunidad de Inteligencia Artificial, y ha sido utilizado para muchas aplicaciones:
  - Demostración Automática de Teoremas
  - Procesamiento de Lenguaje Natural
  - Sistemas Expertos
  - Etc.

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

IMPLEMENTACIONES DE PROLOG

- GNU Prolog
- BProlog
- P#
- Quintus
- SICStus
- SWI-Prolog
- ...

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

2 SWI PROLOG

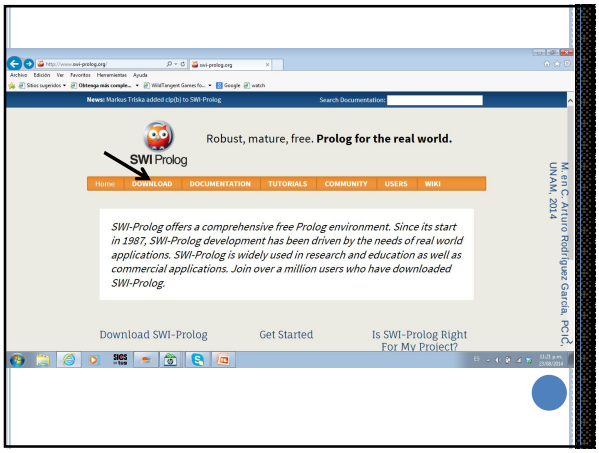
M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

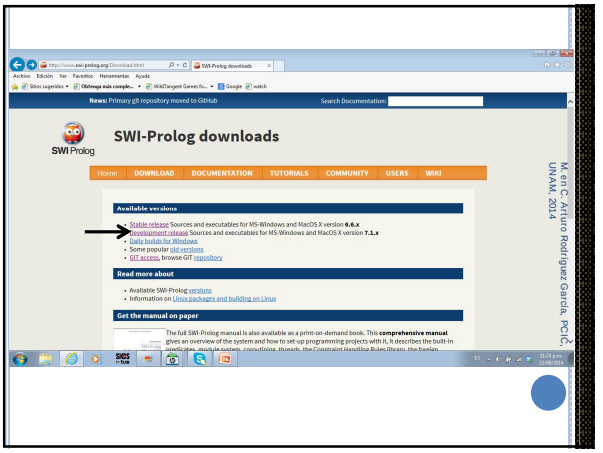
SWI Prolog

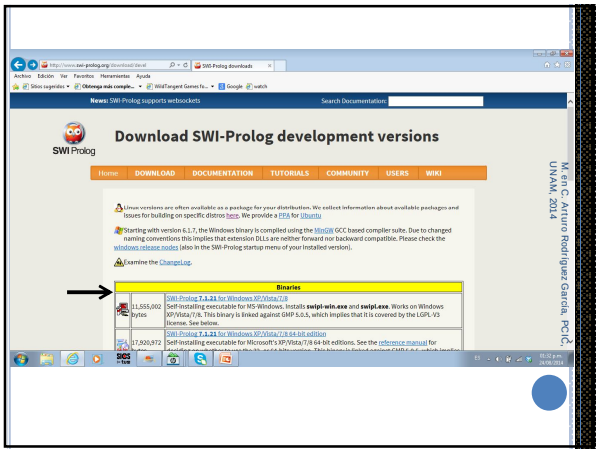
- Es una implementación en código abierto de PROLOG.
- Ha estado en desarrollo desde 1987, siendo Jan Wielemaker su autor principal.
- Las siglas SWI vienen del grupo de investigación *Sociaal-Wetenschappelijke Informatica* de la Universidad de Ámsterdam.

Instalación

- Usaremos la versión 7.1.20, que está disponible para:
  - Windows XP/Vista/7/8
  - MacOSX 10.6 (Snow Leopard) y posteriores
  - Linux (Debian, Ubuntu, Redhat, SuSE, Mageia)
- Entrar a la página oficial de SWI Prolog:
  - http://www.swi-prolog.org/
  - Dar clic en la pestaña DOWNLOAD, elegir la opción SWI-Prolog, para mostrar las versiones disponibles.



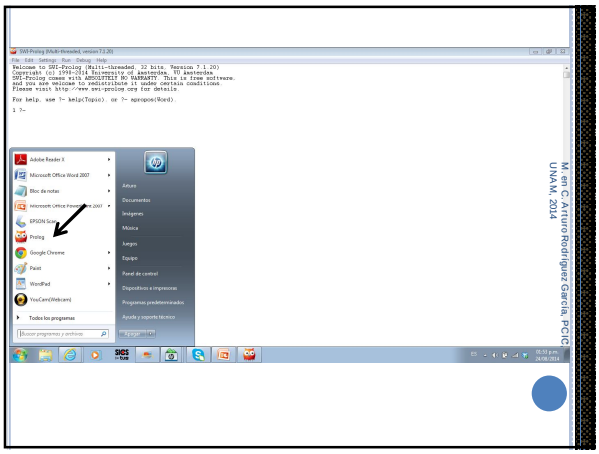




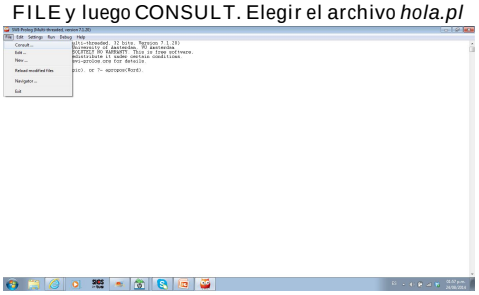
Hola Mundo

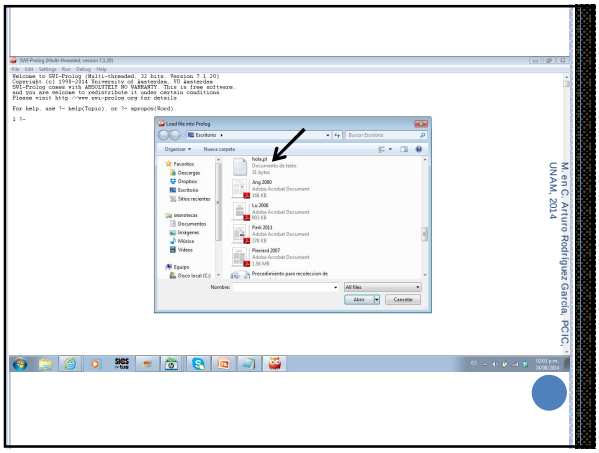
- Crear un archivo de texto llamado *hola.pl* que contenga lo siguiente:

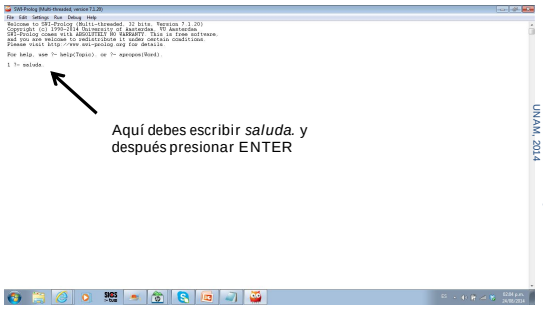
```
saluda:-write('hola mundo').
```



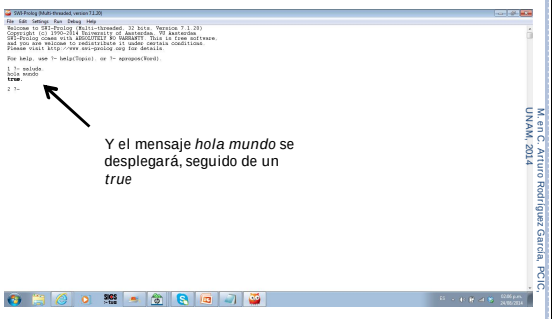
- Una vez abierto, en el menú de opciones elegir FILE y luego CONSULT. Elegir el archivo *hola.pl*



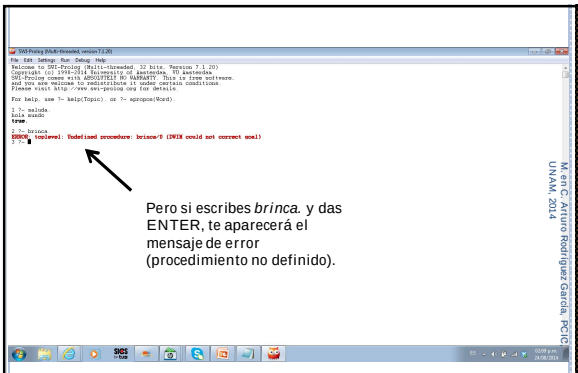




Aquí debes escribir *saluda.* y después presionar ENTER



Y el mensaje *hola mundo* se desplegará, seguido de un *true*



Pero si escribes *brinca.* y das ENTER, te aparecerá el mensaje de error (procedimiento no definido).

- Si modificas el archivo *hola.pl* deberás volver a cargar el archivo (ir al menú, seleccionar FILE, luego CONSULT y elegir el archivo de nuevo).

3 CONCEPTOS BÁSICOS

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

EL PROGRAMA

- Un programa en Prolog es una base lógica. Está compuesta de un conjunto de predicados almacenados en un archivo de texto.

ejemplo.pl

```
mortal(X):- persona(X).
persona(socrates).
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

EL *LISTENER*

- Es un programa que interactúa con el usuario. Realiza la consulta de la base lógica que se le indica y puede responder preguntas que haga el usuario.

```
?-
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

- Por ejemplo, el usuario (en rojo) le pide al *listener* que consulte la base almacenada en el archivo llamado *ejemplo.pl* , y el *listener* (en azul) le responderá que lo hizo exitosamente con un *true*.

```
?-consult('C:/Users/hola.pl').
true.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

- Ahora, cuando el usuario haga una pregunta, el *listener* revisará los predicados que han sido cargados para dar una respuesta.

ejemplo.pl

```
mortal(X):- persona(X).
persona(socrates).
```

```
?- persona(socrates).
true.

?- persona(platon).
false.

?- mortal(socrates).
true.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

4 Hechos

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

Hechos

- Son la forma más simple de los predicados en Prolog. Su sintaxis es:

pred(arg_1,...,arg_n).

- *pred* es el nombre del predicado
- *arg_1, ..., arg_n* son los argumentos
- *n* es la aridad del predicado
- . es el fin de cualquier cláusula en Prolog

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

Argumentos

- Pueden ser cualquier término legal en Prolog:
  - Entero.
  - Átomo (empieza con una letra minúscula).
  - Variable(empieza con una letra mayúscula o con un guión bajo _).
  - Estructura (Término complejo, que revisaremos más adelante).
  - Algunas implementaciones de Prolog permiten números de punto flotante y cadenas.

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

Conjunto de caracteres

- Mayúsculas: A-Z
- Minúsculas: a-z
- Dígitos: 0-9
- Símbolos: + - * / \ ^ , . ~ : ? @ # $ &

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

Ejemplos

- Átomos válidos:
  - abeja
  - r2d2
  - variasPalabrasJuntas
  - usando_guion_bajo
  - 'yo soy un atomo gracias a las comillas simples'
  - *-*??
- Variables válidas:
  - X
  - Maximo_Comun_Divisor
  - _3er_argumento

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

Ejemplos de hechos

misterio_a_la_orden.pl

```
persona(shaggy).
persona(fred).
persona(daphne).
persona(velma).
perro('scooby doo').
perro('scrappy doo').
enemigo('the black knight','mr wickles').
enemigo('charlie the robot','sarah jenkins').
enemigo(zombies).
where_are_you.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

5 CONSULTAS SIMPLES

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

CONSULTAS

- Las consultas de Prolog funcionan mediante emparejamiento de patrones (*pattern matching*).
- El patrón de consulta lo llamamos meta (*goal*).
- Si hay un hecho que se empareja con la meta, entonces la consulta tiene éxito y devuelve *true*. En caso contrario, devuelve *false*.

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

UNIFICACIÓN

- Es el emparejamiento de patrones usado en Prolog. Si la base lógica contiene sólo hechos, la unificación del predicado meta con un predicado en la base lógica tiene éxito si se cumplen estas tres condiciones:
  1. El nombre del predicado es el mismo.
  2. Ambos predicados tienen la misma aridad.
  3. Todos los argumentos son iguales.

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

misterio_a_la_orden.pl

```
persona(shaggy).
persona(fred).
persona(daphne).
persona(velma).
perro('scooby doo').
perro('scrappy doo').
enemigo('the black knight','mr wickles').
enemigo('charlie the robot','sarah jenkins').
enemigo(zombies).
where_are_you.
```

```
?- persona(shaggy).
true.

?- persona(shaggy,23).
false.

?- perro('scooby doo').
true.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

misterio_a_la_orden.pl

```
persona(shaggy).
persona(fred).
persona(daphne).
persona(velma).
perro('scooby doo').
perro('scrappy doo').
enemigo('the black knight','mr wickles').
enemigo('charlie the robot','sarah jenkins').
enemigo(zombies).
where_are_you.
```

```
?- enemigo('the black knight','mr wickles').
true.

?- enemigo('mr wickles','the black knight').
false.

?- where_are_you.
true.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

VARIABLES

- Las variables pueden reemplazar a uno o más argumentos del predicado meta.
- Cuando los argumentos correspondientes son comparados, una variable se empareja con cualquier término.
- Después de la unificación, la variable toma el valor del término con el que se emparejó. A esto se le llama enlazar la variable (*binding*).

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

misterio_a_la_orden.pl

```
persona(shaggy).
persona(fred).
persona(daphne).
persona(velma).
perro('scooby doo').
perro('scrappy doo').
enemigo('the black knight','mr wickles').
enemigo('charlie the robot','sarah jenkins').
enemigo(zombies).
where_are_you.
```

```
?- enemigo(X).
X=zombies.

?- enemigo('the black knight',Y).
Y='mr wickles'.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

- Al usar variables en el predicado meta, puede haber más de un predicado en la base lógica con el que se puede realizar la unificación.
- Prolog regresa el valor de la variable correspondiente al primer predicado de la base lógica con el que unificó. El usuario tiene dos opciones en este momento:
  - Presionar la tecla **N** .- Prolog cancela el enlace de la variable (unbinding) y busca otro predicado en la base lógica con el que se pueda unificar el predicado meta. A esto se le llama *backtracking*.
  - Presionar la tecla **Y** .- Termina la búsqueda en ese momento.

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

misterio_a_la_orden.pl

```
persona(shaggy).
persona(fred).
persona(daphne).
persona(velma).
perro('scooby doo').
perro('scrappy doo').
enemigo('the black knight','mr wickles').
enemigo('charlie the robot','sarah jenkins').
enemigo(zombies).
where_are_you.
```

```
?- persona(X).
X=shaggy. Y
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

misterio_a_la_orden.pl

```
persona(shaggy).
persona(fred).
persona(daphne).
persona(velma).
perro('scooby doo').
perro('scrappy doo').
enemigo('the black knight','mr wickles').
enemigo('charlie the robot','sarah jenkins').
enemigo(zombies).
where_are_you.
```

```
?- persona(X).
X=shaggy N ;
X=fred N ;
X=daphne N ;
X=velma.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

misterio_a_la_orden.pl

```
persona(shaggy).
persona(fred).
persona(daphne).
persona(velma).
perro('scooby doo').
perro('scrappy doo').
enemigo('the black knight','mr wickles').
enemigo('charlie the robot','sarah jenkins').
enemigo(zombies).
where_are_you.
```

```
?- enemigo(X,Y).
X='the black knight',
Y='mr wickles' N ;
X='charlie the robot',
Y='sarah jenkins'.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

MÁS SOBRE UNIFICACIÓN

- Los predicados en la base lógica también pueden tener variables en sus argumentos.
- Al realizar una consulta, un término en el predicado meta se puede enlazar con una variable en un predicado en la base lógica. Sin embargo, el *listener* sólo despliega las variables mencionadas en la consulta.
- Prolog también puede enlazar variables con variables (a partir de ese momento, si una de ellas toma un valor, la otra también lo hará).

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

misterio_a_la_orden.pl

```
persona(shaggy).
persona(fred).
persona(daphne).
persona(velma).
perro('scooby doo').
perro('scrappy doo').
enemigo('the black knight','mr wickles').
enemigo('charlie the robot','sarah jenkins').
enemigo(zombies).
where_are_you.
sospechoso(X).
```

```
?- sospechoso('Captain Cutler').
true.

?- sospechoso(Personaje_del_capitulo).
true.
```


6 CONSULTAS COMPUESTAS

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

CONSULTAS COMPUESTAS

- Una consulta compuesta se realiza combinando predicados meta separados por comas (que funcionan como *and*).
- Si el nombre de una variable aparece varias veces en una consulta, deberá tener el mismo valor en todas sus correspondencias con los predicados en la base lógica.
- El alcance de las variables es únicamente en la consulta en la que aparece.

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

misterio_a_la_orden.pl

```
protagonista(shaggy,masculino).
protagonista(fred,masculino).
protagonista(daphne,femenino).
protagonista(velma,femenino).
protagonista('scooby doo',masculino).
protagonista('scrappy doo',masculino).
humano(shaggy).
humano(fred).
humano(daphne).
humano(velma).
perro('scooby doo').
perro('scrappy doo').
```

```
?- protagonista(X,masculino),perro(X).
X='scooby doo' N ;
X= 'scrappy doo'.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

7 PREDICADOS EXTRA-LÓGICOS

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

PREDICADOS EXTRA-LÓGICOS

- También llamados *built-in predicates*.
- Cuando el *listener* encuentra un predicado meta que corresponde a uno de ellos, un procedimiento predefinido es invocado.
- Los predicados para escribir en la consola son un ejemplo: *write, nl, tab.*
- El predicado *fail* afecta el *backtracking*. Como su nombre lo indica, siempre falla.

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

misterio_a_la_orden.pl

```
protagonista(shaggy,masculino).
protagonista(fred,masculino).
protagonista(daphne,femenino).
protagonista(velma,femenino).
protagonista('scooby doo',masculino).
protagonista('scrappy doo',masculino).
humano(shaggy).
humano(fred).
humano(daphne).
humano(velma).
perro('scooby doo').
perro('scrappy doo').
```

```
?- humano(X),write('Humano:'),tab(3),
write(X),nl,fail.
Humano:   shaggy
Humano:   fred
Humano:   daphne
Humano:   velma
false.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

8 REGLAS

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

REGLAS

- Una regla es una consulta almacenada. Su sintaxis es:
  encabezado:-cuerpo
  - Encabezado.-es la definición de un predicado (como si fuera un hecho).
  - :-se llama "cuello" y se lee como un "si"
  - Cuerpo.-es una consulta, que puede ser simple o compuesta. Los predicados que la forman pueden ser también reglas.

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

misterio_a_la_orden.pl

```
protagonista(shaggy,masculino).
protagonista(fred,masculino).
protagonista(daphne,femenino).
protagonista(velma,femenino).
protagonista('scooby doo',masculino).
protagonista('scrappy doo',masculino).
humano(shaggy).
humano(fred).
humano(daphne).
humano(velma).
perro('scooby doo').
perro('scrappy doo').

heroe(X):-
      protagonista(X,masculino),
      humano(X).
```

```
?- heroe(X).
X=shaggy N ;
X=fred N ;
false
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

misterio_a_la_orden.pl

```
protagonista(shaggy,masculino).
protagonista(fred,masculino).
protagonista(daphne,femenino).
protagonista(velma,femenino).
protagonista('scooby doo',masculino).
protagonista('scrappy doo',masculino).
humano(shaggy).
humano(fred).
humano(daphne).
humano(velma).
perro('scooby doo').
perro('scrappy doo').

heroina(X):- protagonista(X,femenino),
      humano(X).
```

```
?- heroina(X).
X=daphne N ;
X=velma.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

- De la misma manera en que se pueden tener múltiples hechos para un mismo predicado, se pueden tener múltiples reglas para un predicado.
- Un mismo predicado puede tener cláusulas en las que está definido como un hecho, y otras donde es una regla.

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

misterio_a_la_orden.pl

```
persona(shaggy).
persona(fred).
persona(daphne).
persona(velma).
perro('scooby doo').
perro('scrappy doo').
enemigo('the black knight','mr wickles').
enemigo('charlie the robot','sarah jenkins').

protagonista(X):- persona(X).
protagonista(X):- perro(X).
protagonista('vincent van ghoul').
```

```
?- protagonista(X).
X=shaggy N ;
X=fred N ;
X=daphne N ;
X=velma N ;
X='scooby doo' N ;
X='scrappy doo' N ;
X='vincent van ghoul'.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

9 ARITMÉTICA

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

ARITMÉTICA EN PROLOG

- Prolog provee predicados extra-lógicos para evaluar expresiones aritméticas.
  X is <expresión aritmética>

```
?- X is 2+3.
X=5.

?- X is 3*(6/2).
X=9.

?- X is 7/5.
X=1.4.

?- 4 > 3.
true.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

convierte.pl

```
convierte(C,F):-
      F is C * 9 / 5 + 32.
```

```
?- convierte(100,X).
X=212.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

10 RECURSIÓN

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

RECURSIÓN

- Recursión, en cualquier lenguaje de programación, es la habilidad de una unidad de código de llamarse a si misma.
- En Prolog, la recursión ocurre cuando una regla contiene dentro de su cuerpo un predicado meta que se refiere a ella misma.

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

factorial.pl

```
factorial(0,1).
factorial(N,F):-
        N>0,
        N1 is N-1,
        factorial(N1,F1),
        F is N*F1.
```

```
?- factorial(0,1).
true.

?- factorial(0,2).
false.
```


factorial.pl

```
factorial(0,1).
factorial(N,F):-
        N>0,
        N1 is N-1,
        factorial(N1,F1),
        F is N*F1.
```

```
?- factorial(3,X).
X=6.
```


factorial.pl

```
factorial(0,1).
factorial(N,F):-
        N>0,
        N1 is N-1,
        factorial(N1,F1),
        F is N*F1.
```

```
?- factorial(3,F).
X=6.
```

```
factorial(3,F):-
        3>0,            <-true
        N1 is 3-1,      <-calcula N1
        factorial(2,F1), <-consulta
        F is 3*F1.
```


factorial.pl

```
factorial(0,1).
factorial(N,F):-
        N>0,
        N1 is N-1,
        factorial(N1,F1),
        F is N*F1.
```

```
?- factorial(3,F).
X=6.
```

```
factorial(3,F):-
        3>0,            <-true
        N1 is 3-1,      <-calcula N1
        factorial(2,F1), <-consulta
        F is 3*F1.

factorial(2,F):-
        2>0,            <-true
        N1 is 2-1,      <-calcula N1
        factorial(1,F1), <-consulta
        F is 2*F1.
```


factorial.pl

```
factorial(0,1).
factorial(N,F):-
        N>0,
        N1 is N-1,
        factorial(N1,F1),
        F is N*F1.
```

```
?- factorial(3,F).
X=6.
```

```
factorial(3,F):-
        3>0,            <-true
        N1 is 3-1,      <-calcula N1
        factorial(2,F1), <-consulta
        F is 3*F1.

factorial(2,F):-
        2>0,            <-true
        N1 is 2-1,      <-calcula N1
        factorial(1,F1), <-consulta
        F is 2*F1.

factorial(1,F):-
        1>0,            <-true
        N1 is 1-1,      <-calcula N1
        factorial(0,F1), <-consulta
        F is 1*F1.
```


factorial.pl

```
factorial(0,1).
factorial(N,F):-
        N>0,
        N1 is N-1,
        factorial(N1,F1),
        F is N*F1.
```

```
?- factorial(3,F).
X=6.
```

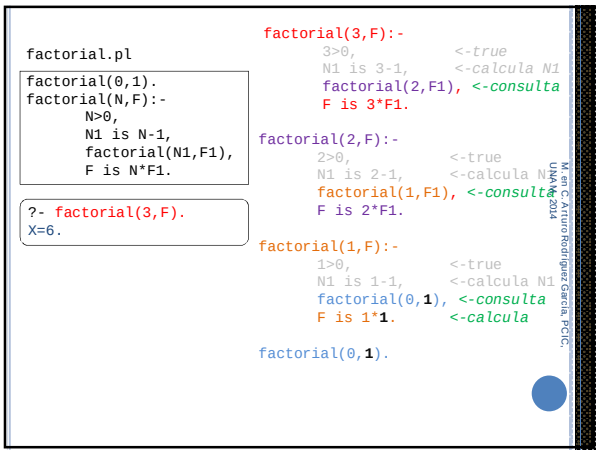
```
factorial(3,F):-
        3>0,            <-true
        N1 is 3-1,      <-calcula N1
        factorial(2,F1), <-consulta
        F is 3*F1.

factorial(2,F):-
        2>0,            <-true
        N1 is 2-1,      <-calcula N1
        factorial(1,F1), <-consulta
        F is 2*F1.

factorial(1,F):-
        1>0,            <-true
        N1 is 1-1,      <-calcula N1
        factorial(0,F1), <-consulta
        F is 1*F1.

factorial(0,1).
```

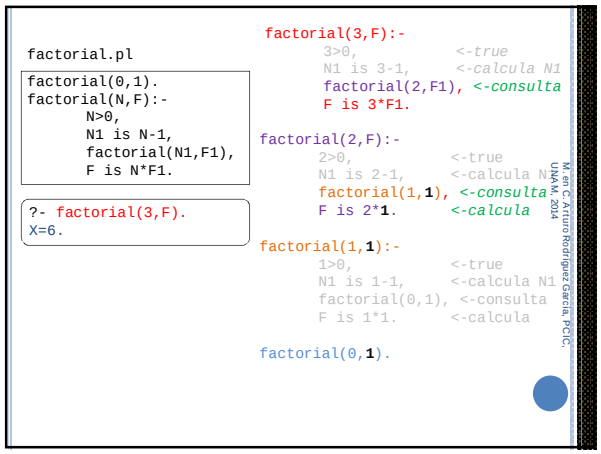
factorial.pl

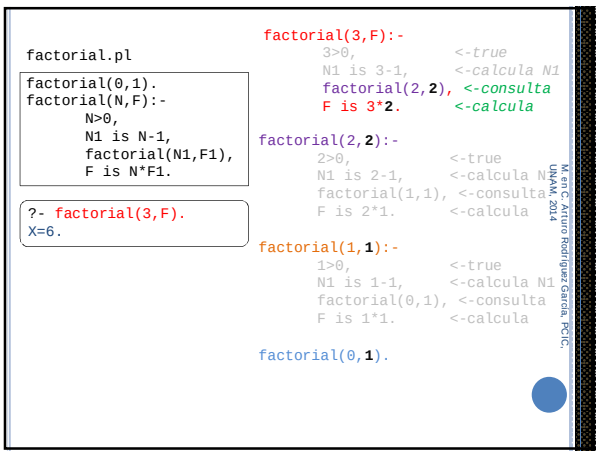
```
factorial(0,1).
factorial(N,F):-
        N>0,
        N1 is N-1,
        factorial(N1,F1),
        F is N*F1.
```

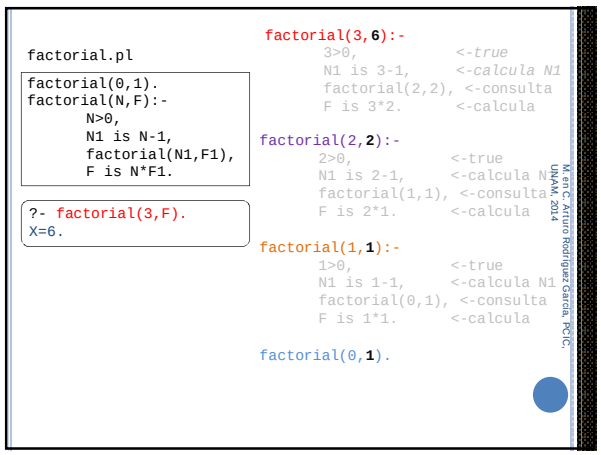
```
?- factorial(3,F).
X=6.
```

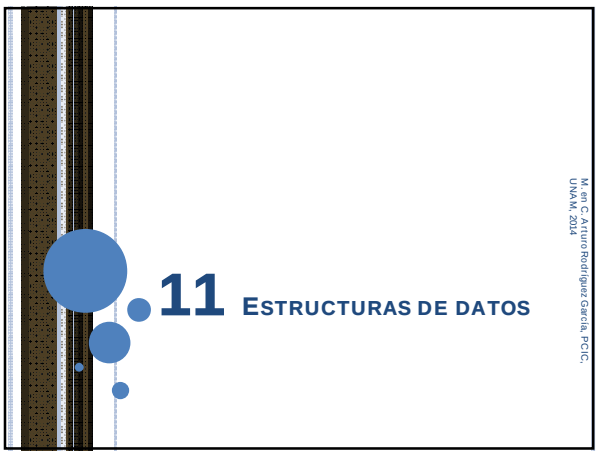
```
factorial(3,F):-
        3>0,            <-true
        N1 is 3-1,      <-calcula N1
        factorial(2,F1), <-consulta
        F is 3*F1.

factorial(2,F):-
        2>0,            <-true
        N1 is 2-1,      <-calcula N1
        factorial(1,F1), <-consulta
        F is 2*F1.

factorial(1,F):-
        1>0,            <-true
        N1 is 1-1,      <-calcula N1
        factorial(0,1), <-consulta
        F is 1*1.       <-calcula

factorial(0,1).
```

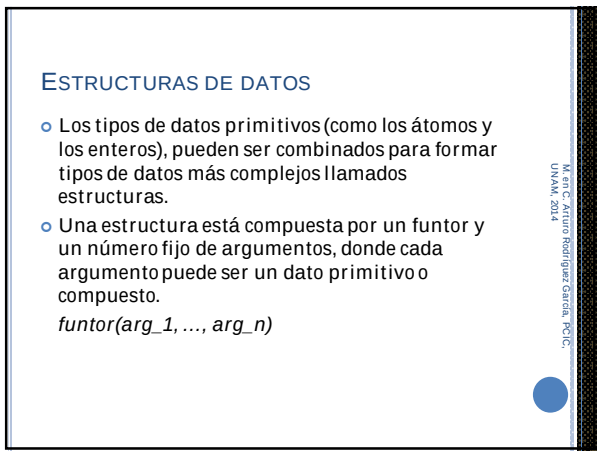








11 ESTRUCTURAS DE DATOS



ESTRUCTURAS DE DATOS

- Los tipos de datos primitivos (como los átomos y los enteros), pueden ser combinados para formar tipos de datos más complejos llamados estructuras.
- Una estructura está compuesta por un funtor y un número fijo de argumentos, donde cada argumento puede ser un dato primitivo o compuesto.

  funtor(arg_1, ..., arg_n)

estructura.pl

```
ubicacion(mesa,cocina).

ubicacion(silla(madera,nueva,cafe),sala).
ubicacion(silla(madera,vieja,cafe),cocina).
ubicacion(silla(metal,nueva,gris),sala).
ubicacion(silla(metal,vieja,gris),patio).
```

```
?- ubicacion(X,cocina).
X=mesa N ;
X=silla(madera,vieja,cafe).

?- ubicacion(silla(X,Y,gris),sala).
X=metal,
Y=nueva.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

UNIFICACIÓN

- Una variable unifica con cualquier término (sea primitiva, estructura o variable).
- Dos términos primitivos unifican sólo si son idénticos.
- Dos estructuras unifican si:
  - Tienen el mismo funtor.
  - Tienen la misma aridad.
  - Cada argumento de la primer estructura unifica con el argumento correspondiente en la segunda estructura.

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

OPERADOR DE UNIFICACIÓN =

- Como su nombre lo indica, provoca la unificación en Prolog.
- Error común.- usar el símbolo = para tratar de asignar un valor (por ejemplo, x = 3 + 4).

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

```
?- hola = hola.
true.

?- a = z.
false.

?- a(b,c,d) = a(b,c,d).
true.

?- a(b) = a(b,c).
false.

?- a(b(c(d,e))) = a(b(c(d,e))).
true.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

```
?- X = sol.
X=sol.

?- 3.1416 = Pi.
Pi=3.1416.

?- X = p(x,y,z).
X=p(x,y,z).

?- color(pez,azul) = color(pez,X).
X=azul.

?- color(pez,X) = color(Y,azul).
X=azul,
Y=pez.
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

```
?- X = Y.
X=Y.

?- X = Y , Y = 5.
X=Y, Y=5.

?- X = Y , Y = 5, Z = X.
X=Y, Y=5, Z=5
```

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

12 LISTAS

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

LISTAS

- Es una estructura de datos que consiste en una colección de términos (que pueden ser cualquier tipo de datos de Prolog, incluyendo estructuras y otras listas).

  [elemento1, elemento2, ... elementoN]

- La lista vacía (también llamada *nil*) es una lista especial que no contiene elementos, y se representa por un par de corchetes vacíos *[ ]* .

NOTACIÓN [H | T] (HEAD & TAIL)

- Es una notación especial para listas. Cuando esta estructura es unificada con una lista, H se enlaza con el primer elemento de la lista y T con el resto de los elementos.

```
?- [H|T] = [a,b,c,d].
H=a,
T=[b,c,d].

?- [H|T] = [ [a,b,c] , [d,e,f] , [g,h] ].
H=[a,b,c],
T=[[d,e,f],[g,h]].

?- [H|T] = [ uno, dos ].
H=uno,
T=[dos].

?- [H|T] = [uno].
H=uno,
T=[].

?- [H|T]=[].
false.
```

listas.pl

```
%Verifica si un elemento dado se encuentra en
una lista

es_elemento(H,[H|T]).

es_elemento(X,[H|T]):-
       es_elemento(X,T).
```

```
?- es_elemento(a,[b,a,t]).
true.

?- es_elemento(k,[h,o,l,a]).
false.
```

CAMBIANDO DE FUNTORES A LISTAS

```
?- X =.. [a,b,c,d].
X=a(b,c,d).

?- p(x,y,z) =.. X.
X=[p,x,y,z].
```

Recolección de soluciones en listas

- El predicado setof genera una lista ordenada de objetos que satisfacen una meta.

setof (Objeto , Meta , Lista)

misterio_a_la_orden.pl

```
protagonista(shaggy,masculino).
protagonista(fred,masculino).
protagonista(daphne,femenino).
protagonista(velma,femenino).
protagonista('scooby doo',masculino).
protagonista('scrappy doo',masculino).
humano(shaggy).
humano(fred).
humano(daphne).
humano(velma).
perro('scooby doo').
perro('scrappy doo').
```

```
?- setof(X,humano(X),List).
List=[daphne,fred,shaggy,velma].

?- setof(X,protagonista(X,femenino),List).
List=[daphne,velma].

?- setof(heroina(X),protagonista(X,femenino),L).
L=[heroina(daphne),heroina(velma)].
```


13 Operadores

Operadores

- Internamente, Prolog sólo tiene una estructura de datos:

  funtor(arg_1, ..., arg_n)
- Sin embargo, Prolog ofrece otras representaciones por cuestiones prácticas.

Por ejemplo, el operador +, utilizado en operaciones aritméticas como 2+2, corresponde a +(2,2).

- Un operador puede ser definido con el predicado *op*, cuyos tres argumentos son:
  - Precedencia.- un número entre 1 y 1200 (una alta precedencia está indicada con un menor número).
  - Asociatividad.- es un patrón que define el tipo de operador. Por ejemplo, *xfx* indica que el operador va entre los argumentos y que no es asociativo (no puedes ponerlos en cadena).
  - El nombre del operador.

operadores.pl

```
:- op(15,xfx,'=>').

a=>b.
```

```
?- a=>b.
true.

?- =>(a,b).
true.

?- a=>(b=>c) = =>(a,=>(b,c)).
true.
```

14 Cut

M. en C. Arturo Rodríguez García, PCIC, UNAM, 2014

Cut !

- Predicado que inhibe el backtracking.
- Se utiliza para evitar que después de encontrar una solución, el programa siga buscando más soluciones.

misterio_a_la_orden.pl

```
persona(shaggy).
persona(fred).
persona(daphne).
persona(velma).
perro('scooby doo').
perro('scrappy doo').
enemigo('the black knight','mr wickles').
enemigo('charlie the robot','sarah jenkins').
enemigo(zombies).
where_are_you.
```

```
?- persona(X).
X=shaggy N ;
X=fred N ;
X=daphne N ;
X=velma.

?- persona(X),!.
X=shaggy.
```

```
?- persona(X),persona(Y).
X=Y, Y=shaggy N ;
X=shaggy, Y=fred N ;
X=shaggy, Y=daphne N ;
X=shaggy, Y=velma N ;
X=fred, Y=shaggy N ;
X=Y, Y=fred N ;
X=fred, Y=daphne N ;
X=fred, Y=velma N ;
X=daphne, Y=shaggy N ;
X=daphne, Y=fred N ;
X=Y, Y=daphne N ;
X=daphne, Y=velma N ;
X=velma, Y=shaggy N ;
X=velma, Y=fred N ;
X=velma, Y=daphne N ;
X=Y, Y=velma.
```

```
?- persona(X),!,persona(Y).
X=Y, Y=shaggy N ;
X=shaggy, Y=fred N ;
X=shaggy, Y=daphne N ;
X=shaggy, Y=velma.

?- persona(X),persona(Y),!.
X=Y, Y=shaggy.
```

REFERENCIAS

- *Adventure in PROLOG*, Deniss Merrit, Amzi! Inc. http://www.amzi.com/AdventureInProlog/index.php
- *SWI Prolog, Reference Manual*, Jan Wielemaker, University of Amsterdam, 2014. http://www.swi-prolog.org/download/devel/doc/SWI-Prolog-7.1.21.pdf