TRABAJANDO CON ARCHIVOS

División de programas sobre archivos

- Muchos predicados de Prolog hacen uso de los mismos predicados básicos.
- Por ejemplo: member/2, append/3
- Redefinirlos no seria una opción.
- La forma más sencilla de decirle a Prolog que lea las definiciones de predicados que se almacenan en un archivo es usar los corchetes.
- ?- [miarchivo].
  - {consulting(miarchivo.pl)…}
  - {miarchivo.pl consulted, 124 bytes}

Leyendo archivos

- También puede consultar más de un archivo a la vez.
- ?- [miarchivo1, miarchivo2, miarchivo3].
  - *{consulting miarchivo1.pl...}*
  - *{consulting miarchivo2.pl...}*
  - *{consulting miarchivo3.pl...}*
- No es necesario hacer esto de forma interactiva.
- En su lugar, puede utilizar una directiva en la base de datos.
  - *:- [miarchivo1, miarchivo2].*

Leyendo archivos

- Tal vez varios archivos, independientemente, consulten el mismo archivo.
- Por lo cual, es importante una comprobación adicional: ensure_loaded/1
- :- ensure_loaded([miarchivo1, miarchivo2]).

Módulos

- Imagine que está escribiendo un programa que administra una base de datos de películas.
- Ha diseñado dos predicados:
  - *printActors/1*
  - *printMovies/1*
- Se almacenan en diferentes archivos y ambos utilizan un predicado auxiliar:
  - *displayList/*

Actores y películas

```
% Este es el archivo: printActors.pl
printActors(Film):-
        setof(Actor,starring(Actor,Film),List),
        displayList(List).
displayList([]):- nl.
displayList([X|L]):-
        write(X), tab(1),
        displayList(L).
```

```
% Este es el archivo: printMovies.pl
printMovies(Director):-
        setof(Film,directed(Director,Film),List),
displayList(List).
displayList([]):- nl.
displayList([X|L]):-
        write(X), nl,
        displayList(L).
```

Principal

- % Este el archivo main.pl
- :- [printActors].
- :- [printMovies].

- ?- [main].
  - *{consulting main.pl}*
  - *{consulting printActors.pl}*
  - *{printActors.pl consulted}*
  - *{consulting printMovies.pl}*
  - *The procedure displayList/1 is*
  - *being redefined.*
  - *Old file: printActors.pl*
  - *New file: printMovies.pl*
  - *Do you really want to redefine it?*
  - *(y, n, p, or ?)*

Usando module

- Module de predicado incorporado:
  - *module/1 y module/2*
- Para crear un módulo/biblioteca: use_module:
  - *use_module/1 y use_module/2*
  - *Para importar predicados desde una biblioteca*
- Argumentos
  - *El primer argumento da nombre del módulo*
  - *El segundo argumento [opcional] es una lista de predicados exportados*
- No todos los intérpretes de Prolog son compatibles con el sistema de módulos
  - *SWI Prolog y Sicstus si*

Usando module

- Module es un predicado incorporado:
  - *module/1 y module/2*
  - *Usado para crear un módulo/biblioteca*
- El predicado incorporado use_module:
  - *use_module/1 y use_module/2*
  - *Para importar predicados desde una biblioteca*
- Argumentos
  - *El primer argumento da nombre del módulo*
  - *El segundo argumento [opcional] es una lista de predicados exportados*

Actores y películas

```
% This is the file: printActors.pl
:- module(printActors,[printActors/1]).

printActors(Film):-
        setof(Actor,starring(Actor,Film),List),
        displayList(List).

displayList([]):- nl.
displayList([X|L]):-
        write(X), tab(1),
        displayList(L).
```

```
% This is the file: printMovies.pl
:- module(printMovies,[printMovies/1]).

printMovies(Director):-
        setof(Film,directed(Director,Film),List),
        displayList(List).

displayList([]):- nl.
displayList([X|L]):-
        write(X), nl,
        displayList(L).
```

Main

```
% This is the revised file main.pl
        :- use_module(printActors).
        :- use_module(printMovies).
```

```
% This is the revised revised file main.pl
        :- use_module(printActors,[printActors/1]).
        :- use_module(printMovies,[printMovies/1]).
```

Librerías

- Muchos de los predicados más comunes están predefinidos por los intérpretes de Prolog
- Por ejemplo, en SWI prolog, **member/2** y **append/3** son como parte de una biblioteca
- Una biblioteca es un módulo que define predicados, y se pueden cargar usando los predicados normales para importar Módulos.

Importando librerías

- Al especificar el nombre de una biblioteca que desea utilizar,
- Puede decir que este módulo es una biblioteca
- Prolog buscará en el lugar correcto, es decir, un directorio donde todas las bibliotecas se almacenan:
  - *:- use_module(library(lists)).*

Escribiendo archivos

- Para escribir en un archivo tenemos que abrir una secuencia,
- Para escribir la cadena**'Hogwarts'** a un archive con el nombre**hogwarts.txt** hacemos:
  - *open(‘hogwarts.txt‘, write, Stream),*
  - *write(Stream, 'Hogwarts'),*
  - *close(Stream),*

Incorporando en archivos

- Para incorporar a un archivo existente tenemos que abrir una secuencia en el modo append.
- Para incorporar el string **'Harry'** en el archive **hogwarts.txt** temenos que hacer:
  - *open('hogwarts.txt', append, Stream),*
  - *write(Stream, 'Harry'),*
  - *close(Stream).*

Escribiendo en archivos

- Resumen de predicados:
  - open/3
  - write/2
  - close/1
- Otros predicados útiles:
  - tab/2
  - nl/1
  - format/3

Leyendo de archivos

- La lectura de información de archivos es directo en Prolog si la información se proporciona en forma de términos de Prolog seguidos de puntos completos
- Leer información de archivos es más difícil si la información no se proporciona en el formato adecuado, nuevamente utilizamos streams y el open y predicados de cierre.
- Consideremos el archivo houses.txt:
  - *gryffindor.*
  - *hufflepuff.*
  - *ravenclaw.*
  - *slytherin.*
- Vamos a escribir un programa Prolog que lea esta información y la muestre en la pantalla.

Leyendo de archivos (casas.pl)

main:-

open('houses.txt',read,S),

read(S,H1),

read(S,H2),

read(S,H3),

read(S,H4),

close(S),

write([H1,H2,H3,H4]), nl.

Resumen de predicados

- Resumen de predicados
  - *open/3*
  - *read/2*
  - *close/1*
- Mas read/2
  - *El predicado read/2 solo funciona en términos de Prolog.*
  - *También causará un error en tiempo de ejecución cuando uno intenta leer al final de un archivo.*

Llegar al final de un stream

- El predicado integrado **at_end_of_stream/1** comprueba si Se ha alcanzado el final de un stream.
- Tendrá éxito cuando finalice el stream (dado a él como argumento) es alcanzado, de lo contrario si fallará,
- Podemos modificar nuestro código para leer en un archivo usando este predicado.

Llegar al final de un stream

```
main:-
        open('houses.txt',read,S),
        readHouses(S,Houses),
        close(S),
        write(Houses), nl.
readHouses(S,[]):-
        at_end_of_stream(S).
readHouses(S,[X|L]):-
        \+ at_end_of_stream(S),
        read(S,X),
        readHouses(S, L).
```

Lectura de entradas arbitrarias

- El predicado **get_code/2** lee la siguiente línea disponible del stream.
  - Primer argumento: un stream.
  - Segundo argumento: el codigo del caracter.
- Ejemplo: un predicado **readWord/2** que lee átomos de un archivo:

```
readWord(Stream,Word):-
        get_code(Stream,Char),
        checkCharAndReadRest(Char,Chars,Stream),
        atom_codes(Word,Chars).

checkCharAndReadRest(10, [], _):- !.
checkCharAndReadRest(32, [], _):- !.
checkCharAndReadRest(-1, [], _):- !.
checkCharAndReadRest(Char,[Char|Chars],S):-
        get_code(S,NextChar),
        checkCharAndRest(NextChar,Chars,S).
```