Aula 11: Trabalhando com arquivos

- Este aula trabalhará os vários aspectos da manipulação de arquivos e da modularidade
- Nós aprenderemos três coisas:
  - Como definições de predicados podem ser dispersadas em diferentes arquivos
  - Como escrever sistemas modulares de software
  - Como escrever resultados para arquivos e como obter entradas a partir dos arquivos

Dividindo programas em arquivos

- Muitos predicados Prolog fazem uso dos mesmos predicados básicos
  - Por exemplo: **member/2**, **append/3**
- É claro que você não quer redefini-los a cada vez que necessitar deles
  - Prolog oferece muitos modos de fazer isto

Leitura de programas

- A forma mais simples de dizer ao Prolog para ler as definições de predicados armazenadas em um arquivo é usar os colchetes

```
?- [meuArq].
{consulting(meuArq.pl)…}
{meuArq.pl consulted, 233 bytes}
true
?-
```

Leitura de programas

- Você também pode consultar mais de um arquivo por vez

```
?- [meuArq1, meuArq2, meuArq3].
{consulting meuArq1.pl…}
{consulting meuArq2.pl…}
{consulting meuArq3.pl…}
```

Leitura de programas

- Você não precisa fazer isto interativamente
- Ao invés disto, você pode usar uma diretiva na base de dados

```
:- [meuArq1, meuArq2].
```

Leitura de programas

- Talvez, muitos arquivos, independentemente, consultem o mesmo arquivo.
- Verificação extra se as definições dos predicados já são conhecidas: ensure_loaded/1

```
:- ensure_loaded([meuArq1, meuArq2]).
```

Módulos

- Imagine que você está escrevendo um programa que gerencie um banco de dados sobre filmes
- Você projetou dois predicados:
  - **imprimeAtores/1**
  - **imprimeFilmes/1**
- Eles estão armazenados em arquivos diferentes
- Ambos usam um predicado auxiliar:
  - **exibeLista/1**

O arquivo imprimeAtores.pl

```
% Este é o arquivo:  imprimeAtores.pl

imprimeAtores(Filme):-
    setof(Ator,elenco(Ator,Filme),Lista),
    exibeLista(Lista).

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

O arquivo imprimeFilmes.pl

```
% Este é o arquivo:  imprimeFilmes.pl

imprimeFilmes(Diretor):-
    setof(Filme,direcao(Diretor,Filme),Lista),
    exibeLista(Lista).

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

O arquivo principal.pl

```
% Arquivo principal.pl

:- [imprimeAtores].
:- [imprimeFilmes].
```

O arquivo principal.pl

```
% Arquivo principal.pl


:- [imprimeAtores].
:- [imprimeFilmes].
```

```
?- [principal].
```

O arquivo principal.pl

```
% Arquivo principal.pl


:- [imprimeAtores].
:- [imprimeFilmes].
```

```
?- [principal].
{consulting principal.pl}
```

O arquivo principal.pl

```
% Arquivo principal.pl


:- [imprimeAtores].
:- [imprimeFilmes].
```

```
?- [principal].
{consulting principal.pl}
{consulting imprimeAtores.pl}
```

O arquivo principal.pl

```
% Arquivo principal.pl


:- [imprimeAtores].
:- [imprimeFilmes].
```

```
?- [principal].
{consulting principal.pl}
{consulting imprimeAtores.pl}
{imprimeAtores.pl consulted}
```

O arquivo principal.pl

```
% Arquivo principal.pl

:- [imprimeAtores].
:- [imprimeFilmes].
```

```
?- [principal].
{consulting principal.pl}
{consulting imprimeAtores.pl}
{imprimeAtores.pl consulted}
{consulting imprimeFilmes.pl}
```

O arquivo principal.pl

```
% Arquivo principal.pl

:- [imprimeAtores].
:- [imprimeFilmes].
```

```
?- [principal].
{consulting principal.pl}
{consulting imprimeAtores.pl}
{imprimeAtores.pl consulted}
{consulting imprimeFilmes.pl}
The procedure exibeLista/1 is
    being redefined.
Old file: imprimeAtores.pl
New file: imprimeFilmes.pl
Do you really want to redefine it?
(y, n, p, or ?)
```

Usando módulos em Prolog

- Predicado pré-construído **module**:
  - **module/1** e **module/2**
  - Para criar um módulo/biblioteca
- Predicado pré-construído **use_module**:
  - **use_module/1** e **use_module/2**
  - Para importar predicados de uma biblioteca
- Argumentos
  - O primeiro argumento é o nome do módulo
  - O segundo e opcional argumento é uma lista dos predicados exportados

Nota sobre módulos em Prolog

- Nem todos os interpretadores Prolog possuem este sistema de módulos.
- SWI Prolog e Sicstus possuem.
- O sistema de módulos do Prolog ainda não é compatível com a norma ISO.

O módulo imprimeAtores.pl

```
% Este é o arquivo:  imprimeAtores.pl
:- module(imprimeAtores,[imprimeAtores/1]).

imprimeAtores(Filme):-
    setof(Ator,elenco(Ator,Filme),Lista),
    exibeLista(Lista).

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

O módulo imprimeFilmes.pl

```
% Este é o arquivo:  imprimeFilmes.pl
:- module(imprimeFilmes,[imprimeFilmes/1]).

imprimeFilmes(Diretor):-
    setof(Filme,direcao(Diretor,Filme),Lista),
    exibeLista(Lista).

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

O arquivo revisado principal.pl

```
% Este é o arquivo revisado principal.pl

:- use_module(imprimeAtores).
:- use_module(imprimeFilmes).
```

```
% Este é o arquivo revisado principal.pl

:- use_module(imprimeAtores,[imprimeAtores/1]).
:- use_module(imprimeFilmes,[imprimeFilmes/1]).
```

Bibliotecas

- Muitos dos predicados mais comuns já vem pré-construídos nos interpretadores Prolog
- Por exemplo, em SWI prolog, **member/2** e **append/3** vem como parte de uma biblioteca
- Uma biblioteca é um módulo que define predicados comuns e pode ser carregada usando os predicados normais para importar módulos

Importando bibliotecas

- Ao especificar o nome de uma biblioteca que você quer usar, você pode informar que este módulo é uma biblioteca
- Prolog procurará no lugar certo, ou seja, em um diretório onde todas as bibliotecas estão guardadas

```
:- use_module(library(lists)).
```

Escrevendo para arquivos

- Para escrever em um arquivo temos que abrir um fluxo (*stream*)
- Para escrever a string **'Hogwarts'** para um arquivo com o nome **hogwarts.txt** faremos assim:

```
...
open(‘hogwarts.txt‘, write, Fluxo),
write(Fluxo, 'Hogwarts'),
close(Fluxo),
...
```

Acrescentado informação aos arquivos

- Para estender um arquivo existente, temos que abrir um fluxo em modo *append*
- Para acrescentar a string **'Harry'** ao arquivo de nome **hogwarts.txt**, faremos:

```
...
open(‘hogwarts.txt‘, append, Fluxo),
write(Fluxo, 'Harry'),
close(Fluxo),
...
```

Escrevendo para arquivos

- Resumo dos predicados:
  - open/3
  - write/2
  - close/1
- Outros predicados úteis:
  - tab/2
  - nl/1
  - format/3

Lendo de arquivos

- Ler informações de arquivos é simples em Prolog se as informações for dada na forma de termos do Prolog seguida por pontos finais.
- Ler a informação dos arquivos é mais difícil se a informação não for dada no formato Prolog.
- Novamente, usaremos fluxos e os predicados *open* e *close.*

Exemplo: lendo dos arquivos

- Considere o arquivo casas.txt:

```
grifinoria.
lufa_lufa.
corvinal.
sonserina.
```

- Escreveremos um programa em Prolog que lê esta informação e a exibe na tela.

Exemplo: lendo dos arquivos

- Aqui está o programa:

casas.txt:

```
grifinoria.
lufa_lufa.
corvinal.
sonserina.
```

```
principal:-
    open('casas.txt',read,F),
    read(F,C1),
    read(F,C2),
    read(F,C3),
    read(F,C4),
    close(F),
    write([C1,C2,C3,C4]), nl.
```

Lendo de arquivos

- Resumo dos predicados
  - open/3
  - read/2
  - close/1

- Mais sobre read/2
  - O predicado read/2 somente funciona com termos do Prolog.
  - Causará um erro em tempo de execução quando se tentar ler o fim do arquivo.

Alcançando o fim de um fluxo

- O predicado pré-construído **at_end_of_stream/1** verifica se o fim de um fluxo foi alcançado.
- Ele sucederá quando o final do fluxo (dado como seu argumento) for alcançado e falhará em caso contrário.
- Podemos modificar nosso código para ler de um arquivo usando este predicado.

Usando at_end_of_stream/1

```
principal:-
    open('casas.txt',read,F),
    leiaCasas(F,Casas),
    close(F),
    write(Casas), nl.

leiaCasas(F,[]):-
     at_end_of_stream(F).

leiaCasas(F,[X|L]):-
     \+ at_end_of_stream(F),
     read(F,X),
     leiaCasas(F, L).
```

Com cortes verdes

```
principal:-
    open('casas.txt',read,F),
    leiaCasas(F,Casas),
    close(F),
    write(Casas), nl.

leiaCasas(F,[]):-
     at_end_of_stream(F), !.

leiaCasas(F,[X|L]):-
     \+ at_end_of_stream(F), !,
     read(F,X),
     leiaCasas(F, L).
```

Com um corte vermelho

```
principal:-
    open('casas.txt',read,F),
    leiaCasas(F,Casas),
    close(F),
    write(Casas), nl.

leiaCasas(F,[]):-
     at_end_of_stream(S), !.

leiaCasas(F,[X|L]):-
     read(F,X),
     leiaCasas(F, L).
```

Lendo uma entrada qualquer

- O predicado **get_code/2** lê o próximo caracter disponível de um fluxo
  - Primeiro argumento: um fluxo
  - Segundo argumento: o código do caracter
- Exemplo: um predicado leiaPalavra***/2*** que lê átomos de um arquivo

Usando get_code/2

```
leiaPalavra(Fluxo,Palavra):-
    get_code(Fluxo,Caracter),
    verificaELeiaResto(Caracter,Caracteres,Fluxo),
    atom_codes(Palavra,Caracteres).

verificaELeiaResto(10, [], _):- !.
verificaELeiaResto(32, [], _):- !.
verificaELeiaResto(-1, [], _):- !.
verificaELeiaResto(Caracter,[Caracter|Caracteres],F):-
      get_code(F,ProxCaracter),
      verificaELeiaResto(ProxCaracter,Caracteres,F).
```

Leitura adicional

- Bratko (2001): Prolog Programming for Artificial Intelligence
  - Aplicações práticas
- O`Keefe (1990): The Craft of Prolog
  - Para hackers avançados de Prolog
- Sterling (1994): The Art of Prolog
  - Com orientação mais teórica