GSI010 - Programação Lógica
Controle

Aula de hoje

- Tratamento de entrada e saída

Arquivos de dados

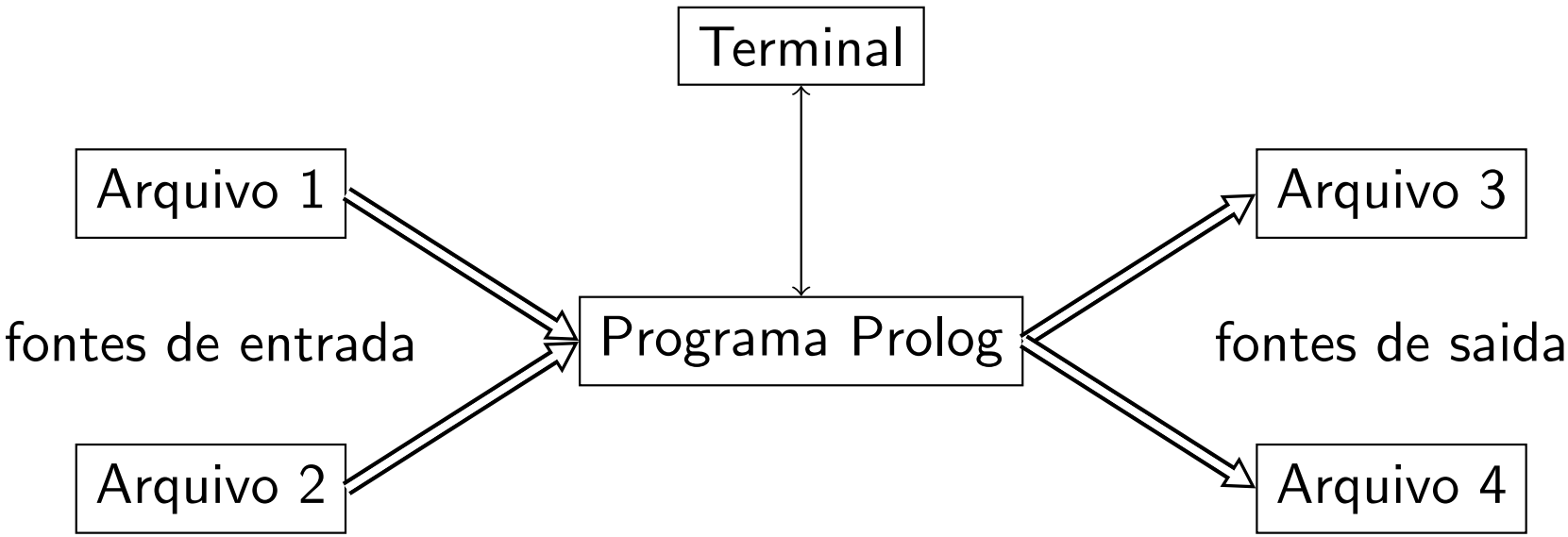
Comunicação em um programa

Para armazenar e recuperar dados de arquivos.

- leitura de arquivos
- leitura de periféricos (teclado, mouse)
- escrita de arquivos

Predicados para entrada e saída

- em Prolog predicados de entrada/saída são dependente da implementação da linguagem
- é possível gerenciar vários arquivos simultâneamente



Arquivos-padrão

Originalmente

- Arquivo entrada padrão: “teclado”
- Arquivo saída padrão: “terminal”

Mudar arquivo entrada padrão:

```
1 see(nomeDoArquivo).
2 faz_leitura(VariavelComConteudo).
3 see(user).
```

Término de leitura de arquivo

Fato especial `end_of_file/0` significa que o arquivo de leitura aberto com o predicado `see/1` chegou ao fim.
O predicado `see(user).` faz voltar para o padrão. Também é possível usar o predicado `seen/0`

```
1 ?- see('entrada.pl').
2 true.
3
4 ?- read(X).
5 X = conteudo.
6
7 ?- read(X).
8 X = end_of_file.
```

Arquivo padrão de escrita

O arquivo padrão de saída é o terminal.

```
?- tell('nome_do_arquivo.pl')
?- write(fato_no_arquivo).
?- told.

?- tell(a).
true.

?- write(fato).
true.

?- write('.').
true.

?- nl.
true.

?- told.
true.
```

Para voltar ao terminal, usar `tell/0`. Exemplo: `tell(user).`

Trabalhando com fluxos

```
1          open('arquivo.txt', write, Fluxo),
2          write('escreve algo'),
3          close(Fluxo).
4          open('arquivo.txt', append, Fluxo),
5          write('escreve algo2'),
6          close(Fluxo).
7
8          open('arquivo.txt', read, Fluxo),
```

`at_end_of_stream(Fluxo)`, verifica se foi atingido o fim do arquivo representado por `Fluxo`.

`read`

`read(Arquivo, X)`

- usado para leitura de dados a partir da fonte corrente
  - leitura do próximo termo que irá unificar com `X`
  - Se `X` é uma variável então `X` será instanciada com conteúdo de Arquivo
  - Se a unificação não for possível então o objetivo `read(X)` irá falhar
  - Cada termo deve ser seguido por um ponto e um espaço ou enter
  - `Arquivo` representa o arquivo e não é obrigatório

É determinístico, ou seja, não permite *backtracking*

read e write

```
1 ?- open('entrada.pl',read,F), read(F,A).
2 F = <stream>(0x1955cd0),
3 A = conteudo.
4
5 ?- open('entrada.pl',append,F), write(F, 'teste.'),
      close(F).
6 F = <stream>(0x195ce00).
```

Outros comandos

- `tab/1` – insere tabulações no arquivo de saída
- `nl` – insere um retorno de linha (enter)
- `flush/0` – confirma a escrita de conteúdo da memória buffer para o arquivo

Escrevendo uma lista na tela:

```
1          escreveLista2([]).
2          escreveLista2([L | LL]) :-
3              imprime(L), nl, escreveLista2(LL).
4
5          imprime([]).
6          imprime([X|L]) :-
7              write(X), tab(1), imprime(L).
```

casas.txt

```
1 grifinoria.
2 lufa_lufa.
3 corvinal.
4 sonserina.
```

```
1 principal:-
2     open('casas.txt',read,F
          ),
3     read(F,C1),
4     read(F,C2),
5     read(F,C3),
6     read(F,C4),
7     close(F),
8     write([C1,C2,C3,C4]),
          nl.
```

```
1  principal:-
2      open('casas.txt',read,F),
3      leiaCasas(F,Casas),
4      close(F),
5      write(Casas), nl.
6
7  leiaCasas(F,[]):-
8      at_end_of_stream(F).
9
10 leiaCasas(F,[X|L]):-
11     \+ at_end_of_stream(F),
12     read(F,X),
13     leiaCasas(F, L).
```

Problema: fornecedor

Em um arquivo temos fatos no seguinte formato: `item(Nro, Descrição, Preço, Fornecedor).`

Problema

Produzir outro arquivo que tenha somente os produtos de um determinado Fornecedor.

```
1          ...
2          read(ArquivoEntrada,item(Nro,D,P,NomeForn)),
3          write(ArquivoSaida, item(Nro,D,P)),
4          ...
```

Processamento de caracteres individuais

Também é possível trabalhar com caracteres individuais em ASCII: `get` e `put`

Um caractere é escrito na fonte de saída corrente por meio do objetivo: `put(C)`

```
1  ?- put(48).
2  0
3  true.
4
5  ? put(65), put(66), put(67).
6  ABC
7  true.
8
9  ?- get(C).
10 |: .
11
12 C = 46.
13
14 ?- put(46).
15 .
16 true.
```

`get(C)` ignora espaços em branco e caracteres não imprimíveis

`get_code/2` não ignora.

Processamento de caracteres: get_code/2

```
1  leiaPalavra(Fluxo,Palavra):-
2      get_code(Fluxo,Caracter),
3      verificaELeiaResto(Caracter,Caracteres,Fluxo),
4      atom_codes(Palavra,Caracteres).
5
6  verificaELeiaResto(10,[],_):- .
7  verificaELeiaResto(32,[],_):- .
8  verificaELeiaResto(1,[],_):- .
9
10 verificaELeiaResto(Caracter,[Caracter|Caracteres],F):-
11     get_code(F,ProxCaracter),
12     verificaELeiaResto(ProxCaracter,Caracteres,F).
```

Conversão de ASCII para atomo (e vice-versa)

Como usar os caracteres lidos na forma de átomos.

```
1 ?- atom_codes(X,[77,77,78]).
2 X = 'MMN'.
3
4 ?- atom_codes(asdd,X).
5 X = [97, 115, 100, 100].
```

Predicados básicos do Prolog

Cada implementação tem um conjunto de predicados básicos para evitar reescrita de funcionalidades comuns.

```
?- append([2],[3],X).
X = [2, 3].

?- append([2],[3,4],X).
X = [2, 3, 4].

?- member(2,[2,3,4,2]).
true ;
true.

[arq1,arq2,arq2]. % sao arquivos de uma biblioteca de
    predicados e fatos
```

Usando módulos

Predicado préconstruído `module`:

- `module/1` e `module/2`
- para criar um módulo/biblioteca

o Predicado préconstruído `use_module`:

- `use_module/1` e `use_module/2`
- Para importar predicados de uma biblioteca

o Argumentos

Definindo módulos em um arquivo `imprimeAtores.pl`

```
1 %o primeiro argumento é o nome do módulo
2 %o segundo é uma lista dos predicados exportados
3 :- module(imprimeAtores,[imprimeAtores/1]).
```

É possível dizer quais predicados são exportados, e.g., `imprimeAtores/1`. Evita conflito entre predicados com mesmo nome em diferentes arquivos.

Carregando bibliotecas

Consultar ajuda de bibliotecas:

```
:- use_module(library(lists)).
?- help(append).
```

Bibliotecas variadas e úteis:

```
:- use_module(library(http/http_client)).
:- use_module(library('http/http_sgml_plugin')).

encontrar(X, [X|_]) .
encontrar(X, [Y|L]) :- encontrar(X,L) ; encontrar(X, Y)
    .
encontrar(X, Y) :- not(Y=[_|_]), compound(Y), Y=..L,
    encontrar(X, L).

pegar_url(URL) :-
http_get(URL, DOM, [proxy('proxy.ufu.br',3128)]), DOM=[
    H|_], assert(H).


pegar_titulo(URL, Titulo) :-
    pegar_url(URL),
    element(X,Y,Z),
    encontrar(element(meta,A,_),Z),
    A=[H|T],
    H=..L,
    L=[_,name,title],
    T=..TL,
    TL=[_,TL2|_],
    TL2=..[_,_,Titulo|_],
    retract(element(X,Y,Z)).

% exemplo de uso
% pegar_titulo('http://www.imdb.com/title/tt0068646/',
    Titulo).
```

Referências

- User guide, Programming in XPCE/Prolog, Wielemaker e Anjewierden (2005).
- Luis, A. M. Palazzo, Introdução à programação prolog, Educat, 1997
- Slides profs. Elaine Faria, Hiran Nonato e Gabriel Coutinho - UFU
- Slides da Profa. Solange - ICMC - USP