



I/O

Prof. Dr. Carlos Lopes




Lendo Termos

- Prolog possui o predicado *read* para ler um termo do stream de dados de entrada corrente (por default é o teclado).
- *read(X)* faz com que o próximo termo seja lido e unificado a *X*. O termo deve ser expresso de acordo com as regras de sintaxe e deve ser terminado por um . (ponto).
- O predicado *read* é determinístico. não havendo sucesso não haverá tentativa da leitura de outro termo.
- Se *read(X)* é executado no fim do stream de entrada corrente então *X* é unificado com *end_of_file*.




Escrevendo Termos

- O predicado *write(X)* permite escrever termos no stream de saída corrente.
- Similar a *read*, *write* é determinístico.




Lendo e Escrevendo Termos: Exemplo

```
cubo :-  
    read(X), processa_cubo(X).  
  
processa_cubo(end_of_file) :- !.  
processa_cubo(N) :-  
    Y is N*N*N,  
    write(Y), nl,  
    cubo.  
  
?- cubo.
```



Lendo e Escrevendo Termos: Exemplo Revisto

```
cubo :-  
    write('Proximo Numero: '),  
    read(X), processa_cubo(X).  
  
processa_cubo(end_of_file) :- !.  
processa_cubo(N) :-  
    Y is N*N*N,  
    write('O cubo de '),  
    write(N), write(' eh '),  
    write(Y), nl, cubo.
```



Lendo e Escrevendo Termos: um outro exemplo

```
dialogo :- nl, nl,  
    write('Qual eh o seu nome?'),nl,  
    read(N),nl,  
    write('Bom Dia, '),write(N),nl.
```

Uso do programa:
?- dialogo.
Qual eh o seu nome?
Carlos
Bom Dia, Carlos

Processamento de Arquivos

- Até agora somente usamos entrada padrão (teclado) e saída padrão (tela).
- É possível especificar outros streams de entrada e saída de dados. Entretanto, em qualquer momento somente dois streams estarão ativos.
- Predicados a serem usados:
 - `see(Arq1)`: Arq1 é o novo stream de entrada.
 - `seen`: fecha o stream de entrada.
 - `tell(Arq2)`: Arq2 é o novo stream de saída.
 - `told`. Fecha o stream de saída.

Processamento de Arquivos - *template*

- O processamento de um arquivo tipicamente tem o seguinte *template*:

```
...,
see(F), processaArquivo, seen, see(user),
...
```

```
processaArquivo :-
    read(Termo), processa(Termo).
```

```
processa(end_of_file) :- !.
processa(Termo) :-
    trata(Termo), processaArquivo.
```

Processamento de Arquivos - *Exemplo*

- O programa exemplo consiste copiar o conteúdo de um arquivo X para um arquivo Y.

```
copia(X,Y) :-
    see(X), tell(Y), read(Frase),
    faz_copia_de(Frase), seen, told.
faz_copia_de(Frase) :-
    write(Frase), write(' '), nl,
    Frase=end_of_file, !.
faz_copia_de(Frase) :- read(Outra_Frase),
    faz_copia_de(Outra_Frase).
```

Manipulação de Caracteres

- É possível manipular caracteres usando três predicados:
 - `get0(C)`: lê um simples caractere
 - `get(C)`: lê o próximo caractere diferente de branco
 - `put(C)`: escreve um caractere
- C é o código ASCII para o caractere.
- Em SWI-prolog, usa-se entrada pelo teclado em buffer.

Manipulação de Caracteres - Exemplo

Desejamos definir um predicado que permita ao usuário digitar s para sim e n para não e este predicado devolva a opção digitada.

```
obtemsn(Resultado) :-
    get(Char),
    get0(_), % desconsidera Enter
    digitado(Char, Resultado), !.
obtemsn(Resultado) :- % se digitado fracassar
    nl, write('Favor Digitar S ou N: '),
    obtemsn(Resultado).
digitado(83, sim). % ASCII S
digitado(115, sim). % ASCII s
digitado(78, nao). % ASCII N
digitado(110, nao). % ASCII n
```

Manipulando Átomos

- O predicado `name` associa átomos com seus códigos ascii equivalentes.
 - Exemplo: `name(X,[49,50,51])` faz X=123
- Exemplo: definir o predicado `le_atomo(X)` que devolve um átomo da linha (string) lida.

```
le_atomo(Atomo) :-
    le_str(String),
    name(Atomo, String).
```

Manipulando Átomos (cont.)

```
le_str(String) :-
    get0(Char),
    le_str_aux(Char, String).

le_str_aux(-1, []) :- !. % EOF
le_str_aux(10, []) :- !. % EOL(UNIX)
le_str_aux(13, []) :- !. % EOL (DOS)
le_str_aux(Char, [Char|Resto]) :-
    le_str(Resto).
```

Inserindo os Programas na Base de Conhecimento de Prolog

- Para inserir o programa na base de conhecimento do Prolog você pode usar:

```
[testel]. % testel.pl é o arquivo
['testel']
['testel.pl']
consult(testel).
consult('testel').
consult('testel.pl').
```
- O caminho completo pode ser especificado e o diretório pode ser mudado com o predicado *chdir*.

Inserindo Programas (cont.)

- Durante qualquer sessão Prolog você pode usar *consultar* arquivos muitas vezes. Cada arquivo *consultado* tem suas cláusulas adicionadas ao conjunto de cláusulas existentes.
- Em geral *consult* lê um conjunto completo de termos Prolog usando *read* e inclui cada termo na base de conhecimento de Prolog como um fato ou regra.

Inserindo Programas: Observação

- Em algumas versões de Prolog (não é o caso de SWI) a repetição de um *consult* pode fazer com que cláusulas sejam repetidas. Para evitar isto você deve *reconsultar* o arquivo.

```
reconsult(testel).
reconsult('testel').
reconsult('testel.pro').
```
- SWI-Prolog não tem um predicado *reconsult*. *consult* assume também o significado de *reconsult*.

Consultas Embutidas

- Um programa Prolog, além de fatos e regras, pode conter consultas. Elas já foram usadas anteriormente para definir operadores.
- Exemplo:

```
:- write('Lendo a base de dados...'), nl
pai(pedro, manael).
pai(joaquim, pedro).
:- write('Fim'), nl.
?- [pai].
Lendo a base de dados...
Fim
```

Cuidado!

- A maioria das implementações Prolog proporcionam facilidades para a interação com o usuário.
- SWI-Prolog, por exemplo, possui predicados compatíveis com o padrão de Edimburgo e com Quintus Prolog.
- Para qualquer linguagem de programação a interface entre o que o sistema requer e aquilo que o usuário vê é sempre baseada em aspectos do sistema operacional e de detalhes de implementação da linguagem. Portanto tenha cuidado! Estes são aspectos que podem ser alterados de uma versão para outra.



Formas de Acesso a Arquivos

- A maioria das impl. Prolog possui uma forma de acessar arquivos através de identificadores de streams. Isto permite que os arquivos possam ser abertos de vários modos. Este método de manipulação de arquivos é parte do padrão ISO.
- O seguinte código lê um termo de um arquivo e o escreve em outro arquivo.

```
teste :-open('f1.txt', read, F1),  
        read(F1, Termo), close(F1),  
        open('f2.txt', write, F2),  
        write(F2, Termo), close(F2).
```

- Modos de acesso:
read, write, append, update.