Recursão, condicionais e I/O em Prolog

Por João Zucchi e Raul Steinmetz

Paradigmas de Programação



Input e Output em Prolog é bem simples.

Para leitura e escrita de dados no terminal existem 3 comandos.

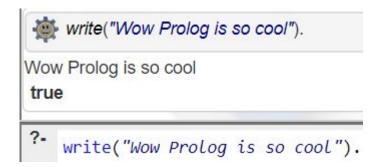
write() e print()

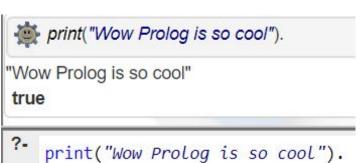
Com write() é necessário utilizar aspas para indicar texto, com print() as aspas são printadas;

Aspas simples destacam o texto;

Variáveis são printadas sem aspas;

print() é mais indicado para printar variáveis, write() é um comando mais geral.





read()

As vezes é necessário interagir com o usuário, com read(X): Lê um valor e o atribui à variável X.

```
% lendo variavel
reading() :-
    read(Z),
    write(Z).
```

Output em Files

CONDICIONAIS

Os condicionais em Prolog são muito similares aos já vistos em Haskell.

Existem duas formas de fazer condicionais em Prolog.

RECURSIVIDADE

Novamente, muito parecido com Haskell.

- 1. Condição de parada
- Função com Recursividade (com chamada a si própria

```
% contar e printar com recursao
printar(10) :- write(10), nl.
printar(X) :-
   write(X), nl,
   Y is X + 1,
    printar(Y).
% printar e contar com recursao inversamente
printarReverse(10) :- write(10), nl.
printarReverse(X) :-
   Y is X + 1,
    printarReverse(Y),
   write(X), nl.
```

Criamos uma cadeia alimentar usando a relação come(A, B), que indica que um animal 'A' come o animal 'B'.

```
% criação da cadeia alimentar
come(grilo, vegetal).
come(sapo, grilo).
come(cobra, sapo).
come(gaviao, cobra).
come(caramujo, vegetal).
come(peixe, caramujo).
come(pelicano, peixe).
come(besouro, vegetal).
come(sapo, besouro).
come(aquia, sapo).
come(rato, vegetal).
come(cobra, rato).
come(coelho, vegetal).
come(veado, vegetal).
come(raposa, coelho).
come(aguia, coelho).
come(aguia, raposa).
come(lobo, raposa).
come(lobo, veado).
```

Função que determina se um animal A está acima de B em uma certa cadeia alimentar.

```
?- cadeiaAlimentar(aguia, sapo).
true .
```

```
cadeiaAlimentar(A, B) :- come(A, B).
cadeiaAlimentar(A, B) :-
   come(A, X),
   cadeiaAlimentar(X, B).
```

```
?- cadeiaAlimentar(aguia, X).
X = sapo;
X = coelho;
X = raposa;
X = grilo;
X = besouro;
X = vegetal;
X = vegetal;
X = vegetal;
X = coelho;
X = vegetal
```

Escrever um caminho dentro da Cadeia Alimentar, de um animal X (entrada do usuário) até a base da cadeia alimentar (vegetal).

```
escreverCadeiaPredador() :-
    open("predador.txt", write, Stream),
   write("Digite o nome do animal: "),
    read(X).
    cadeiaAlimentarEscrita(X, vegetal, Stream),
   write(Stream, X),
    close(Stream).
cadeiaAlimentarEscrita(A, B, Stream) :-
    come(A, B),
   write(Stream, B),
    nl(Stream).
   write(Stream, " ^"),
   nl(Stream),
   write(Stream, " |"),
    nl(Stream).
cadeiaAlimentarEscrita(A, B, Stream) :-
    come (A, X),
    cadeiaAlimentarEscrita(X, B, Stream),
   write(Stream, X),
    nl(Stream).
   write(Stream, " ^"),
   nl(Stream).
   write(Stream, " |"),
    nl(Stream).
```

Output

.txt —

```
?- escreverCadeiaPredador().
Digite o nome do animal: pelicano.
```

true .

```
vegetal

caramujo

l
peixe

pelicano
```

DESAFIO

Implementar o algoritmo proposto;

Primeira equipe que implementar o algoritmo ganha um BOMBOM;

Em 2 minutos cronometrados;

ARE YOU READY?



TERMINAL: fibonacci(1,0,10).

 1
 8

 1
 13

 2
 21

 3
 34

 5
 55

```
fibonacci(X,Y,N) :-
    N > 0
    Z is X + Y,
    M is N - 1,
    write(X),
    nl,
    fibonacci(Z,X,M).
```