

Tópicos em Inteligência Artificial: Programação em Lógica

José Carlos Bins Filho

Ciência da Computação
UFFS

25 de Abril de 2017



Universidade Federal da Fronteira Sul

Roteiro I

- ① Unidade 1: Administrativo
- ② Unidade 2: Introdução
- ③ Unidade 3: Prolog: Tipos de Dados
- ④ Unidade 4: Prolog: Predicados Pré-definidos
 - 4.1: Predicados Interpretador
 - 4.2: Predicados Listas
 - 4.3: Predicados Lógicos
 - 4.4: Predicados de Teste de Tipos
 - 4.5: Predicados Entrada e Saída
 - 4.6: Predicados de Controle
 - 4.7: Predicados de Criação de Operadores
 - 4.8: Predicados de Conversão de Tipos
 - 4.9: Predicados de Coleta de Soluções
- ⑤ Unidade 5: Prolog: Banco de Dados

Tópicos em Inteligência Artificial: Program

2 / 93

Roteiro I

- ① Unidade 1: Administrativo
- ② Unidade 2: Introdução
- ③ Unidade 3: Prolog: Tipos de Dados
- ④ Unidade 4: Prolog: Predicados Pré-definidos
 - 4.1: Predicados Interpretador
 - 4.2: Predicados Listas
 - 4.3: Predicados Lógicos
 - 4.4: Predicados de Teste de Tipos
 - 4.5: Predicados Entrada e Saída
 - 4.6: Predicados de Controle
 - 4.7: Predicados de Criação de Operadores
 - 4.8: Predicados de Conversão de Tipos
 - 4.9: Predicados de Coleta de Soluções
- ⑤ Unidade 5: Prolog: Banco de Dados

Ementa

- Introdução ao paradigma de programação lógica e representação de conhecimento usando lógica.
- Aplicações de programação em lógica e linguagem Prolog.

- Mensagens gerais serão dadas pelo Moodle apenas.
-

- $MF = \frac{2.5 \times P_1 + 2.5 \times P_2 + 5.0 \times T}{10} \geq 6.0$
- Prova Substitutiva no final do semestre para substituir a nota da prova.

	Aula	Data
Prova I		
Prova II		
Prova Sub		
Trabalho Final		

Bibliografia

- Básica
 - ▶ NILSSON, U.; MALUSZYNSKI, J. Logic, Programming and Prolog. John Wiley and Sons, 1995. 276 p.
 - ▶ BRATKO, I. PROLOG Programming for Artificial Intelligence. 2a.ed., New York, Addison-Wesley, 1990.
 - ▶ CASANOVA, M. A., GIORNO, F. A. C., FURTADO, A. L. Programação em lógica e a linguagem PROLOG. São Paulo, E. Blucher, 1987.
 - ▶ STERLING, L, SHAPIRO, E. The art of PROLOG. London, The MIT Press, 1986.

Bibliografia

- Complementar
 - ▶ Patrick Blackburn, Johan Bos, Kristina Striegnitz Learn Prolog Now, College Publications, 2006.
 - ▶ William F. Clocksin and Christopher S. Mellish Programming in Prolog: Using the ISO Standard, 5a ed. Springer Verlag, 2003.
 - ▶ STERLING, L. The Practice of Prolog. MIT PRESS, 1990. RUSSEL, S.; NORVIG, P. Inteligência Artificial. Rio de Janeiro: Campus, 2004.

1 Unidade 1: Administrativo

2 Unidade 2: Introdução

3 Unidade 3: Prolog: Tipos de Dados

4 Unidade 4: Prolog: Predicados Pré-definidos

4.1: Predicados Interpretador

4.2: Predicados Listas

4.3: Predicados Lógicos

4.4: Predicados de Teste de Tipos

4.5: Predicados Entrada e Saída

4.6: Predicados de Controle

4.7: Predicados de Criação de Operadores

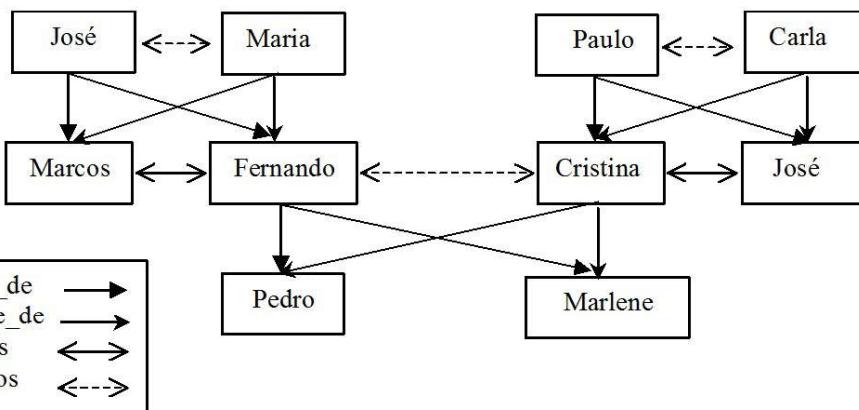
4.8: Predicados de Conversão de Tipos

4.9: Predicados de Coleta de Soluções

5 Unidade 5: Prolog: Banco de Dados

- Linguagem de programação baseada em lógica.
- Usa um paradigma de programação completamente diferente das linguagens procedurais como Fortran, C, Pascal, etc..., chamado de declarativo.

	Procedural	Declarativa
Programação	Descreve a solução do problema	Descreve o problema
Aplicação	Processamento Numérico	Processamento Simbólico
Execução	Sequencial	Por Demanda
Dados × Instruções	Diferentes	Difícil de diferenciar
Tradução	Compilada	Interpretada ou Kernel Compilado
Busca	Não há	Backtracking
Paralelismo	De difícil implementação	Inerentemente paralela
Determinismo	Determinista	Potencialmente não determinista



Como representar em Prolog

- Cláusula de Horn: disjunção de literais onde no máximo 1 literal é positivo.
 $\neg p \vee \neg q \vee \dots \vee \neg t \vee u$
- ou usando uma representação diferente:
 $u \leftarrow p \wedge q \wedge \dots \wedge t$
- onde :
 - ▶ u : é a cabeça da cláusula
 - ▶ $p \wedge q \wedge \dots \wedge t$: é o corpo da cláusula

- Em Prolog:
 - ▶ Uma clausula sem cabeça é um objetivo/pergunta.
 - ▶ Uma clausula sem corpo é um fato/dado.
 - ▶ Uma clausula completa é uma regra/instrução.
- O programa "parentesco.pl" é um programa que manipula/define relações de parentesco entre um grupo de pessoas.

- Prolog SWI
- Deve estar no C:\ProgramFiles\pl
- Se não, instalar do G:\swi
- Home page: <http://www.swi-prolog.org/>
- Dicas:
 - ▶ Edite o arquivo em editor e salve como texto com terminação ".pl".
 - ▶ Clique no arquivo que abrirá o SWI Prolog com o arquivo dentro ou alternativamente:
 - ▶ Dentro do swi (pl\bin\plwin.exe) leia o arquivo com ['<nomearq>']. Use apenas "/" para o path.

Embasamento teórico do Prolog

- Lógica proposicional
- Cálculo de predicados de primeira ordem
- Forma clausal de predicados de primeira ordem
- Clausulas de Horn
- Resolução (Robison 1965)
- Unificação

Lógica Proposicional

- Fórmula proposicional é um conjunto de proposições e símbolos lógicos
- Símbolos lógicos:
 - ▶ pontuação: {(,)}
 - ▶ conectivos: { \neg , \wedge (ou & ou .), \vee (ou +), \Rightarrow , \iff }
- Exemplo:
 - ▶ P : Está chovendo
 - ▶ Q : Está Nublado
 - ▶ $P \Rightarrow Q, P \vdash Q$
- Ou seja: Sabe-se que:

Se está chovendo então está nublado, e
Está chovendo, portanto conclui-se que;
Está nublado.

Lógica Proposicional

- P : Está chovendo
- Q : Está Nublado

P	Q	P	$P \Rightarrow Q$	T
F	F	F	V	F
F	V	F	V	F
V	F	V	F	F
V	V	V	V	V

Lógica Proposicional

- Expressse as proposições abaixo em lógica proposicional e ache a conclusão, se houver.
 - ▶ Sócrates está disposto a visitar Platão, se Platão estiver disposto a visitá-lo.
 - ▶ Platão não está disposto a visitar Sócrates, se Sócrates estiver disposto a visitá-lo, mas está disposto a visitar Sócrates, se Sócrates não estiver disposto a visitá-lo
 - ▶ Sócrates está disposto a visitar Platão, se Platão estiver disposto a visitá-lo.
 - ▶ Platão não está disposto a visitar Sócrates, se Sócrates estiver disposto a visitá-lo, mas está disposto a visitar Sócrates, se Sócrates não estiver disposto a visitá-lo
 - ▶ P : Sócrates está disposto a visitar Platão
 - ▶ Q : Platão está disposto a visitar Sócrates
 - ▶ $Q \Rightarrow P$
 - ▶ $P \Rightarrow \neg Q$
 - ▶ $\neg P \Rightarrow Q$
 - ▶ $T : Q \Rightarrow P, P \Rightarrow \neg Q, \neg P \Rightarrow Q \vdash ?$

Lógica Proposicional

- P : Sócrates está disposto a visitar Platão
- Q : Platão está disposto a visitar Sócrates

P	Q	$Q \Rightarrow P$	$P \Rightarrow \neg Q$	$\neg P \Rightarrow Q$	T
F	F	V	V	F	F
F	V	F	V	V	F
V	F	V	V	V	V
V	V	V	F	V	F

Lógica de Primeira Ordem ou Cálculo de Predicados

- Permite o uso de quantificadores e variáveis em fórmulas
- **Exemplo 2.1:** para todo X , existe Y inteiro que satisfaça $Y > X$
 $\forall X, \exists Y \mid Y \in \mathbb{N} \wedge Y > X$ ou mais precisamente
 $\forall X, \exists Y \mid \text{inteiro}(Y) \wedge \text{maior}(Y, X)$
- Fórmula de primeira ordem é um conjunto de termos e símbolos lógicos
- Símbolos lógicos:
 - ▶ pontuação: $\{(),\}$
 - ▶ conectivos: $\{\neg, \wedge(\text{ou } \& \text{ ou }), \vee(\text{ou } +), \Rightarrow, \Leftrightarrow\}$
 - ▶ quantificadores: $\{\forall, \exists, \nexists\}$
 - ▶ variáveis: $\{X, Y, Z, \dots\}$
- Termos:
 - ▶ constantes: exemplo: $\{a, b, joao, maria, 1, 2\}$
 - ▶ funções ou predicados (n-ários): exemplo: $\text{casado}(joao, maria)$

- Cálculo de predicados pode ser usado como modelo de execução
- **Exemplo 2.2:** Linguagem para descrição de listas
 - Alfabeto: $A = \{a_1, a_2, a_3, a_4, a_5, \dots\}$ e nil
 - $\text{nil} \in \text{Lista}(A)$
 - $\exists t(t \in A \vee t \in \text{Lista}(A)) \wedge \exists u(u \in \text{Lista}(A)) \Rightarrow (t, u) \in \text{Lista}(A)$
 - nil
 - (a_1, nil)
 - (a_2, nil)
 - $(a_2, (a_1, \text{nil}))$
 - $((a_2, \text{nil}), (a_1, \text{nil})), \dots$

- $p(t_1, t_2, \dots, t_n)$ é uma fórmula atômica se p é o símbolo de um predicado e t_1, t_2, \dots, t_n são símbolos do alfabeto.
- Um literal é uma fórmula atômica ou sua negação
- Uma cláusula é um conjunto de literais
- Um conjunto de cláusulas S é uma representação clausal de uma fórmula P se e somente se:
 - ▶ P é satisfazível se e somente se S é satisfazível
- Existe um conjunto de regras para transformar uma formula em uma cláusula

Forma Clausal de um Predicado

- Convertendo:
 $\forall x \forall y (\text{chama}(x, y) \Rightarrow (\exists u \text{ programa}(x, u) \wedge \exists v \text{ program}(y, v)))$
- Substitui segundo u por outra variável:
 $\forall x \forall y (\text{chama}(x, y) \Rightarrow (\exists u \text{ programa}(x, u) \wedge \exists v \text{ program}(y, v)))$
- Remove implicação:
 $\forall x \forall y (\neg \text{chama}(x, y) \vee (\exists u \text{ programa}(x, u) \wedge \exists v \text{ program}(y, v)))$
- Substitui u e v por predicados dependentes de x e y :
 $\forall x \forall y (\neg \text{chama}(x, y) \vee (\text{programa}(x, f(x)) \wedge \text{program}(y, g(y))))$
- ou:

```
programa(x,f(x)) :- chama(x,y)
programa(y,g(y)) :- chama(x,y)
```

Clausula de Horn

- Conforme dito anteriormente, uma cláusula de Horn é uma forma clausal com apenas um literal positivo.

- A resolução é uma regra que gera uma nova clausula a partir de outras clausulas:
- É dividida em :
 - ▶ Generalização do Modus ponens
 - ▶ Unificação
 - ▶ Fatoração

- Generalização do Modus ponens:
 - ▶ Modus Ponens: Se $p \Rightarrow q$ e p então q
 - ▶ Generalização: Se $\neg p \vee q$ e p então q
- Unificação:
 - ▶ Unificar é tornar dois literais iguais através da substituição mais geral possível.
 - ▶ **Exemplo 2.3:** $pred(f(X), Y, g(Y))$ unifica com $pred(Z, Z, g(W))$ pelas substituições:
 - ★ $Z = f(X)$
 - ★ $Y = Z = f(X)$
 - ★ $W = Y = f(X)$
 - ▶ e consequentemente ambos ficam iguais a: $pred(f(X), f(X), g(f(X)))$
 - ▶ **Observação:** X não é instanciado porque não é necessário.

Resolução

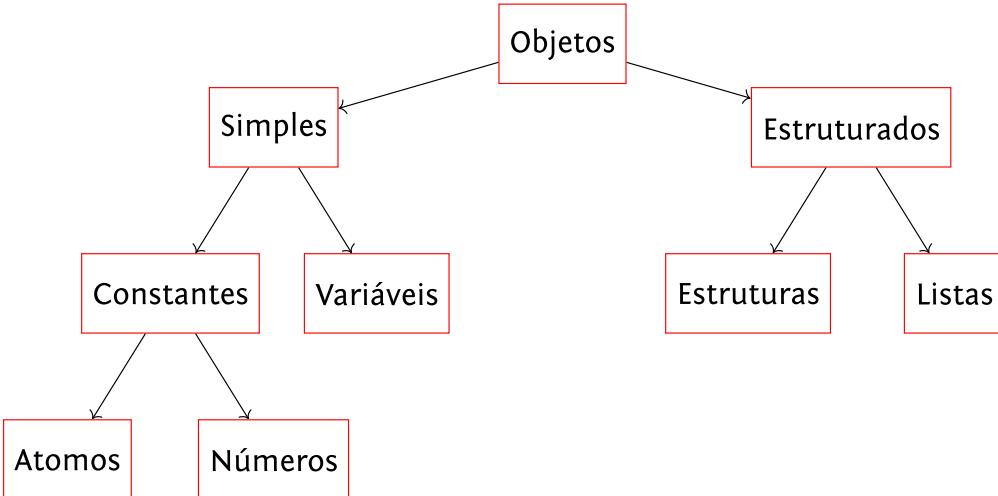
- Se A_1 e A_2 são fatores das clausulas B_1 e B_2 , L_1 é literal de B_1 e $\neg L_2$ é literal de B_2 , e existe uma substituição ξ que faz $L_1\xi = L_2\xi$ então derive $(B_1\xi - L_1\xi) \wedge (B_2\xi - \neg L_2\xi)$
- **Exemplo 2.4:** Dados:
 - ▶ 1: $chama(a, b)$
 - ▶ 2: $usa(b, e)$
 - ▶ 3: $\neg chama(X, Y) \vee depende(X, Y)$
 - ▶ 4: $\neg usa(X, Y) \vee depende(X, Y)$
- Para 1 e 3:
 - ▶ $chama(a, b)$ é fator de $chama(a, b)$
 - ▶ $depende(X, Y)$ é fator de $\neg chama(X, Y) \vee depende(X, Y)$
 - ▶ $L_1: chama(a, b)$
 - ▶ $\neg L_2: \neg chama(X, Y)$
- Portanto podemos derivar:
 - ▶ $depende(a, b)$ com $\xi = \{X = a, Y = b\}$
- Similarmente para 2 e 4 podemos derivar:
 - ▶ $depende(b, e)$ com $\xi = \{X = b, Y = e\}$

Linguagem Prolog

- Fatos \times Regras \times Objetivos
- Uma cláusula Prolog está dividida em cabeça e corpo
 $a(X, Y) : -b(X), c(X, Y)$
- Uma cláusula pode representar um fato, uma regra ou um objetivo.
 - ▶ Fato: é uma cláusula sem corpo, portanto ela é sempre verdadeira.
 - ▶ Regra: é uma cláusula completa, a cabeça da cláusula é verdadeira se os predicados do corpo forem verdadeiros.
 - ▶ Objetivo: é uma cláusula sem cabeça, logo os predicados do corpo são testados mas não gera nenhuma conclusão. Se houver mais de um objetivo, os objetivos são testados até que haja falha.

- 1 Unidade 1: Administrativo
- 2 Unidade 2: Introdução
- 3 Unidade 3: Prolog: Tipos de Dados
- 4 Unidade 4: Prolog: Predicados Pré-definidos
 - 4.1: Predicados Interpretador
 - 4.2: Predicados Listas
 - 4.3: Predicados Lógicos
 - 4.4: Predicados de Teste de Tipos
 - 4.5: Predicados Entrada e Saída
 - 4.6: Predicados de Controle
 - 4.7: Predicados de Criação de Operadores
 - 4.8: Predicados de Conversão de Tipos
 - 4.9: Predicados de Coleta de Soluções
- 5 Unidade 5: Prolog: Banco de Dados

- Em Prolog, tudo são objetos. Estes objetos podem ser subdivididos conforme abaixo:



Tipos de dados

Tipos de dados

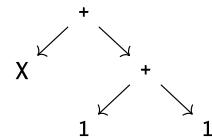
- Simples:
 - ▶ Átomo
 - ★ Qualquer cadeia de letras, dígitos e sublinha ‘_’ que comece por letra minúscula
 - ★ Qualquer cadeia de caracteres entre apóstrofes
 - ★ **Exemplo 3.1:** aBC, texto, variavel_livre, ‘Jose Carlos’
 - ▶ Números:
 - ★ Inteiros: -3, 569
 - ★ Reais: -3.47, 623.0

- Simples:
 - ▶ Variáveis:
 - ★ Qualquer cadeia de letras, dígitos e sublinha ‘_’ que comece por letra maiúscula ou sublinha
 - ★ **Exemplo 3.2:** X, Nome, _
 - ★ Variáveis no Prolog não tem tipo até que seja feita uma atribuição (instanciamento), neste caso a variável assume o tipo do dado atribuído.
 - ★ O escopo das variáveis é a clausula, e neste escopo só uma atribuição pode ser feita à variável a não ser que a atribuição seja desfeita através de backtracking.

- Estruturados

- Estruturas:

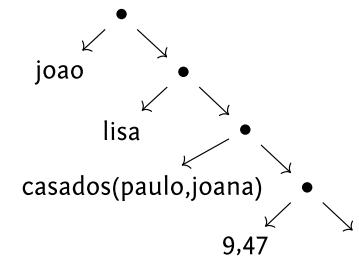
- ★ Objetos que contêm vários componentes. Os componentes podem por sua vez ser estruturas ou dados simples.
 - ★ **Exemplo 3.3:** `data(05,04,2002), +(X, +(1, 1))`
 - ★ Uma estrutura é definida pelo seu nome (funtor) e pelo número de argumentos (aridade).
 - ★ **Exemplo 3.4:** `data(13,4)` e `data(13,4,02)` são estruturas diferentes.
 - ★ Uma estrutura pode ser vista como uma árvore n-ária onde a raiz da árvore é o nome da estrutura.



- Estruturados

- Listas:

- ★ Uma lista é uma sequência de itens. Os itens podem ser de qualquer tipo, inclusive outras listas.
 - ★ Uma lista pode ser vista como uma árvore binária onde a raiz da árvore é o operador de lista “.”, o nodo esquerdo é a cabeça (primeiro elemento) da lista e o nodo direito é o resto da lista.
 - ★ O último elemento de uma lista é sempre a lista vazia, representada por `[]`.
 - ★ **Exemplo 3.5:** `[joao, lisa, casados(paulo, joana), 9.47]`



- Trabalhando com listas:

- Predicado Membro: Testa se um elemento é membro de uma lista
 - % se a lista chegou ao fim então o item não pertence a lista.
 - `membro(_, []) :- fail.`
 - % se o item é igual ao primeiro elemento da lista então % ele pertence a lista
 - `membro(X, [X|R]).`
 - % se o item não é igual ao primeiro elemento da lista % então testa se ele pertence ao resto da lista
 - `membro(X,[P|R]) :- membro(X,R).`

- **Exercícios:** Faça predicados para as seguintes tarefas:

- Ex 3.1:** Imprime uma lista
- Ex 3.2:** Retorna o primeiro elemento de uma lista
- Ex 3.3:** Retorna o último elemento de uma lista
- Ex 3.4:** Remove o último elemento de uma lista
- Ex 3.5:** Inverte uma lista
- Ex 3.6:** Conta o número de elementos é de uma lista
- Ex 3.7:** Conta quantas vezes um dado elemento aparece numa lista

Observação: Ver uma possível solução no arquivo `listas.pl` no Moodle.

- **Exercícios:** Faça predicados para as seguintes tarefas:

Ex 3.8: Acha a posição de um item numa lista

Ex 3.9: Retorna o item na posição especificada de uma lista

Ex 3.10: Recebe duas listas e testa se a primeira lista pertence a segunda lista

Ex 3.11: Recebe duas listas e testa se uma lista pertence a outra lista e retorna a lista e a sublista nesta ordem

Ex 3.12: Inclui um elemento no fim de uma lista

Ex 3.13: Recebe duas listas e as concatena

Ex 3.14: Recebe duas listas e subtrai a segunda da primeira

- **Soluções:** Faça predicados para as seguintes tarefas:

► **Ex 3.8:** Acha a posição de um item numa lista

★ **Solução 3.8:** Com inicialização:

%% 1: constante usada para iniciar a contagem

%% X: item desejado (entrada)

%% L: Lista (entrada)

%% N: nro do elemento desejado para o item (saída)

nro_membro(_, _, [], 0) :- fail.

nro_membro(N, X, [X|R], N).

nro_membro(N, X, [Y|R], N1) :- N2 is N + 1, nro_membro(N2, X, R, N1).

★ **Solução 3.8:** Sem inicialização:

%% X: item desejado (entrada)

%% L: Lista (entrada)

%% N: nro do elemento desejado para o item (saída)

nro_membro(X, L, N) :- busca_membro(1, X, L, N).

busca_membro(_, _, [], 0) :- fail.

busca_membro(N, X, [X|R], N).

busca_membro(N, X, [Y|R], N1) :- N2 is N + 1,

busca_membro(N2, X, R, N1).

Tipos de dados

- **Soluções:** Faça predicados para as seguintes tarefas:

► **Ex 3.12:**] Inclui um elemento no fim de uma lista

★ **Solução 3.12:**

%% X: elemento à incluir (entrada)

%% L: Lista inicial (entrada)

%% NL: Nova lista após inclusão de X (saída)

incluir_no_fim(X, [], [X]).

incluir_no_fim(X, [Y|R], [Y|R1]) :- incluir_no_fim((X,R),R1).

Tipos de dados

- **Trabalho I:** Implemente os exercícios

{3.7, 3.9, 3.10, 3.11, 3.13, 3.14} e mais algum predicado usando listas que você ache interessante, e entregue o código na próxima aula. Inclua um predicado chamado teste que chame todos os predicados criados pelo menos uma vez.

- Aritmética em Prolog:

- ▶ O operador "`=`" é reservado em Prolog para unificação, por isto o operador de atribuição usado é o operador "`is`".

- ▶ **Exemplo 3.6:**

?- A is 3.5, B is 1 + A , C = 1 + A.

retorna:

A = 3.5

B = 4.5

C = 1 + 3.5

- ▶ Operadores de comparação forciam a avaliação da expressão e compararam os resultados.`{>, <, >=, <=, =:=, \=}`

- ▶ **Exemplo 3.7:**

?- 1 + 3 =:= 3 + 1.

yes

?- 1 + 3 = 3 + 1.

no

?- 1 + 3 =3 + 1.

no

- Aritmética em Prolog:

- ▶ Na comparação o Prolog segue uma ordem padrão que é dada por:

- ★ Variáveis < Números < Strings < Átomos < Termos compostos

- ▶ Além disto, dentro de cada um existe uma ordem que é:

- ★ Variáveis: Ordem de endereço.

- ★ Números: ordem dos números; se um inteiro e um real representam o mesmo número o real é maior.

- ★ Strings: Ordem alfabética.

- ★ Átomos: Ordem alfabética

- ★ Termos compostos: Primeiro a aridade; depois o nome alfabeticamente; depois recursivamente os parâmetros.

Tipos de dados

- Aritmética em Prolog:

- ▶ O operador "`\=`" é usado para testar falha na unificação.
- ▶ Podem também ser usados operadores de multiplicação, divisão, etc.

- ▶ **Exemplo 3.8:**

?- A is 7/3.

A = 2.333333333333335

?- A is 7//3.

A = 2

?- A is 3 * 5.

A = 15

?- A is 7 ^ 3.

A = 343

?- A is 7 mod 3.

A = 1

Tipos de dados

- Aritmética em Prolog:

- ▶ **Exercício 3.15:** Dê o resultado das pesquisas abaixo:

- ★ ?- A is 4+3, A =:= 7.

- ★ ?- A is 4+3, A = 4+3.

- ★ ?- A is 4+3, A is 7.

- ★ ?- A is 4+3, A is 8.

- ★ ?- A = 4+3, A = 7.

- ★ ?- A = 4+3, A = 4+3.

- ★ ?- A = 4+3, A = 3+4.

- ★ ?- A = 4+3.

- ?- A = 3+4.

- **Exercícios:** Faça predicados para as seguintes tarefas:

Ex 3.16: Retorne o maior elemento de uma lista

Ex 3.17: Retorne o somatório da lista

Ex 3.18: Retorne a média da lista

Ex 3.19: Retorne se a lista está ordenada. A direção da ordenação é dada por um parâmetro de entrada

- **Soluções:** Faça predicados para as seguintes tarefas:

► **Ex 3.19:**] Retorne se a lista está ordenada. A direção da ordenação é dada por um parâmetro de entrada

★ **Solução 3.19:**

%% O: Ordem a-ascendente d-descendente(entrada)

%% L: Lista inicial (entrada)

ordenada(_, []).

ordenada(a, [X,Y|R]) :- X <= Y, ordenada(a,[Y|R]).

ordenada(d, [X,Y|R]) :- X >= Y, ordenada(d,[Y|R]).

Tipos de dados

- Prolog possui uma série imensa de operadores, a tabela abaixo mostra todos eles e a precedência dos mesmos.

Prec.	Assoc.	Operadores
1200	xfx	-->, :-
1200	fx	:-, ?-
1150	fx	dynamic, discontiguous, initialization, meta_predicate, module_transparent, multifile, public, thread_local, thread_initialization, volatile
1100	xfy	;
1050	xfy	->, *->
1000	xfy	,
990	xfx	:=
900	fy	\+
700	xfx	<, =, =.., =@=, @=, =:=, =<, ==, =, >, >=, @<, @=<, @>, @>=, \=, \==, as, is, ><, ::
600	xfy	:
500	yfx	+, -, /, \, xor
500	fx	?
400	yfx	*, /, //, div, rdiv, <<, >>, mod, rem
200	xfx	**
200	xfy	^
200	fy	+, -, \
100	yfx	.
1	fx	\$

Tipos de dados

- Além disto Prolog permite que se crie e modifique a prioridade de operadores.
- Isto será visto mais adiante.

- 1 Unidade 1: Administrativo
- 2 Unidade 2: Introdução
- 3 Unidade 3: Prolog: Tipos de Dados
- 4 Unidade 4: Prolog: Predicados Pré-definidos
 - 4.1: Predicados Interpretador
 - 4.2: Predicados Listas
 - 4.3: Predicados Lógicos
 - 4.4: Predicados de Teste de Tipos
 - 4.5: Predicados Entrada e Saída
 - 4.6: Predicados de Controle
 - 4.7: Predicados de Criação de Operadores
 - 4.8: Predicados de Conversão de Tipos
 - 4.9: Predicados de Coleta de Soluções
- 5 Unidade 5: Prolog: Banco de Dados

- O prolog possui uma série de predicados pré-definidos que ajudam o programador.
 - ▶ Predicados que manipulam dados (conversão, teste, etc ...)
 - ▶ Predicados que manipulam listas (tamanho, cabeça da lista, etc ...)
 - ▶ Predicados que auxiliam a depuração do programa
 - ▶ Predicados de entrada e saída
 - ▶ Predicados que manipulam o banco de cláusulas, etc...

- 1 Unidade 1: Administrativo
- 2 Unidade 2: Introdução
- 3 Unidade 3: Prolog: Tipos de Dados
- 4 Unidade 4: Prolog: Predicados Pré-definidos
 - 4.1: Predicados Interpretador
 - 4.2: Predicados Listas
 - 4.3: Predicados Lógicos
 - 4.4: Predicados de Teste de Tipos
 - 4.5: Predicados Entrada e Saída
 - 4.6: Predicados de Controle
 - 4.7: Predicados de Criação de Operadores
 - 4.8: Predicados de Conversão de Tipos
 - 4.9: Predicados de Coleta de Soluções
- 5 Unidade 5: Prolog: Banco de Dados

- Predicados úteis do interpretador:
 - ▶ help/0
 - ▶ help/1: help(+Pred)
 - ▶ listing/0
 - ▶ listing/1: listing(+Pred)
 - ▶ edit/1: edit(+Spec)
 - ▶ halt/0
 - ▶ trace/0
 - ▶ notrace/0
 - ▶ spy/1: spy (+Pred).
 - ▶ nospy/1: nospy (+Pred)
 - ▶ nospyall/0

- 1 Unidade 1: Administrativo
- 2 Unidade 2: Introdução
- 3 Unidade 3: Prolog: Tipos de Dados
- 4 Unidade 4: Prolog: Predicados Pré-definidos
 - 4.1: Predicados Interpretador
 - 4.2: Predicados Listas
 - 4.3: Predicados Lógicos
 - 4.4: Predicados de Teste de Tipos
 - 4.5: Predicados Entrada e Saída
 - 4.6: Predicados de Controle
 - 4.7: Predicados de Criação de Operadores
 - 4.8: Predicados de Conversão de Tipos
 - 4.9: Predicados de Coleta de Soluções
- 5 Unidade 5: Prolog: Banco de Dados

- Predicados com listas: Os principais são:
 - ▶ append(?List1, ?List2, ?List3)
 - ▶ member(?Elem, ?List)
 - ▶ delete(+List1, ?Elem, ?List2)
 - ▶ nth1(?Index, ?List, ?Elem)
 - ▶ flatten(+NestedList, -FlatList)
- Mas existem muitos mais predicados pré-definidos que manipulam listas, a lista completa é:
 - ▶ member(?Elem, ?List)
 - ▶ append(?List1, ?List2, ?List1AndList2)
 - ▶ append(+ListOfLists, ?List)
 - ▶ prefix(?Part, ?Whole)
 - ▶ select(?Elem, ?List1, ?List2)
 - ▶ selectchk(+Elem, +List, -Rest)

- Continuação:
 - ▶ select(?X, ?XList, ?Y, ?YList)
 - ▶ selectchk(?X, ?XList, ?Y, ?YList)
 - ▶ delete(+List1, @Elem, -List2)
 - ▶ nth0(?Index, ?List, ?Elem)
 - ▶ nth1(?Index, ?List, ?Elem)
 - ▶ nth0(?N, ?List, ?Elem, ?Rest)
 - ▶ nth1(?N, ?List, ?Elem, ?Rest)
 - ▶ last(?List, ?Last)
 - ▶ proper_length(@List, -Length)
 - ▶ same_length(?List1, ?List2)
 - ▶ reverse(?List1, ?List2)
 - ▶ permutation(?Xs, ?Ys)
 - ▶ flatten(+NestedList, -FlatList)
 - ▶ max_member(-Max, +List)
 - ▶ min_member(-Min, +List)
 - ▶ sum_list(+List, -Sum)
 - ▶ max_list(+List:list(number), -Max:number)
 - ▶ min_list(+List:list(number))
 - ▶ numlist(+Low, +High, -List)

- Continuação:
 - ▶ is_set(@Set)
 - ▶ list_to_set(+List, ?Set)
 - ▶ intersection(+Set1, +Set2, -Set3)
 - ▶ union(+Set1, +Set2, -Set3)
 - ▶ subset(+SubSet, +Set)
 - ▶ subtract(+Set, +Delete, -Result)

Roteiro I

- 1 Unidade 1: Administrativo
- 2 Unidade 2: Introdução
- 3 Unidade 3: Prolog: Tipos de Dados
- 4 Unidade 4: Prolog: Predicados Pré-definidos
 - 4.1: Predicados Interpretador
 - 4.2: Predicados Listas
 - 4.3: Predicados Lógicos**
 - 4.4: Predicados de Teste de Tipos
 - 4.5: Predicados Entrada e Saída
 - 4.6: Predicados de Controle
 - 4.7: Predicados de Criação de Operadores
 - 4.8: Predicados de Conversão de Tipos
 - 4.9: Predicados de Coleta de Soluções
- 5 Unidade 5: Prolog: Banco de Dados

- **Predicados Lógicos:**

- ▶ true
- ▶ fail
- ▶ not(+Term)

- **Operadores Lógicos:**

- ▶ , (e)
- ▶ ; ou | (ou)

Roteiro I

- 1 Unidade 1: Administrativo
- 2 Unidade 2: Introdução
- 3 Unidade 3: Prolog: Tipos de Dados
- 4 Unidade 4: Prolog: Predicados Pré-definidos
 - 4.1: Predicados Interpretador
 - 4.2: Predicados Listas
 - 4.3: Predicados Lógicos**
 - 4.4: Predicados de Teste de Tipos**
 - 4.5: Predicados Entrada e Saída
 - 4.6: Predicados de Controle
 - 4.7: Predicados de Criação de Operadores
 - 4.8: Predicados de Conversão de Tipos
 - 4.9: Predicados de Coleta de Soluções
- 5 Unidade 5: Prolog: Banco de Dados

- **Predicados que checam tipos:**

- ▶ var(+Term)
- ▶ nonvar(+Term)
- ▶ integer(+Term)
- ▶ float(+Term)
- ▶ number(+Term)
- ▶ atom(+Term)
- ▶ string(+Term)
- ▶ atomic(+Term) : atom, string, integer or floating point number.
- ▶ ground(+Term): não tem variáveis livres
- ▶ is_list(+Term)
- ▶ compound(+Term)

Roteiro I

1 Unidade 1: Administrativo

2 Unidade 2: Introdução

3 Unidade 3: Prolog: Tipos de Dados

4 Unidade 4: Prolog: Predicados Pré-definidos

4.1: Predicados Interpretador

4.2: Predicados Listas

4.3: Predicados Lógicos

4.4: Predicados de Teste de Tipos

4.5: Predicados Entrada e Saída

4.6: Predicados de Controle

4.7: Predicados de Criação de Operadores

4.8: Predicados de Conversão de Tipos

4.9: Predicados de Coleta de Soluções

5 Unidade 5: Prolog: Banco de Dados

- Entrada e Saída:

- ▶ Manipulação de arquivos. os básicos são:

- ★ see(+SrcDest)

- ★ tell(+SrcDest)

- ★ append(+File)

- ▶ Outros:

- ★ seeing(?SrcDest)

- ★ telling(?SrcDest)

- ★ seen

- ★ told

- ★ with_output_to(+Output, Goal)

- Entrada e Saída:

- ▶ Outros:

- ★ open(+SrcDest, +Mode, --Stream, +Options)

- SrcDest: Nome do arquivo

- Mode: append;update

- Stream: Nome do fluxo de dados

- Opções: read, write, execute, etc...

- ★ close(+Stream)

- ★ current_stream(?Object, ?Mode, ?Stream)

- ★ is_stream(+Term)

- ★ seek(+Stream, +Offset, +Method, -NewLocation)

- ★ set_input(+Stream)

- ★ set_output(+Stream)

- ★ current_input(-Stream)

- ★ current_output(-Stream)

- ★ at_end_of_stream

- ★ at_end_of_stream(+Stream)

- ★ set_end_of_stream(+Stream)

- ★ copy_stream_data(+StreamIn, +StreamOut, +Len)

- ★ copy_stream_data(+StreamIn, +StreamOut)

- ★ read_pending_input(+StreamIn, -Codes, ?Tail)

- Entrada e Saída:

- ▶ Entrada e Saída de Termos:

- ★ read(-Term)

- ★ write(+Term)

- ★ writeq(+Term)

- ▶ Outros:

- ★ write_term(+Term, +Options)

- ★ write(+Stream, +Term) [ISO]

- ★ write_term(+Stream, +Term, +Options)

- ★ write_length(+Term, -Length, +Options)

- ★ writeq(+Stream, +Term)

- ★ writeln(+Term)

- ★ writeln(+Stream, +Term)

- Entrada e Saída:

- ▶ Outros:

- ★ print(+Term)
 - ★ print(+Stream, +Term)
 - ★ portray(+Term)
 - ★ read(-Term)
 - ★ read(+Stream, -Term)
 - ★ read_clause(+Stream, -Term, +Options)
 - ★ read_term(-Term, +Options)
 - ★ read_term(+Stream, -Term, +Options)
 - ★ read_term_from_atom(+Atom, -Term, +Options)
 - ★ read_history(+Show, +Help, +Special, +Prompt, -Term, -Bindings)
 - ★ prompt(-Old, +New)
 - ★ prompt1(+Prompt)

- Entrada e Saída:

- ▶ Entrada e Saída de Caracteres:

- ★ put(+Char)
 - ★ get(-Char)
 - ★ nl

- ▶ Outros:

- ★ nl(+Stream)
 - ★ put(+Stream, +Char)
 - ★ put_byte(+Byte)
 - ★ put_byte(+Stream, +Byte)
 - ★ put_char(+Char)
 - ★ put_char(+Stream, +Char)
 - ★ put_code(+Code)
 - ★ put_code(+Stream, +Code)
 - ★ tab(+Amount)
 - ★ tab(+Stream, +Amount)
 - ★ flush_output
 - ★ flush_output(+Stream)
 - ★ ttyflush

- Entrada e Saída:

- ▶ Outros:

- ★ get_byte(-Byte)
 - ★ get_byte(+Stream, -Byte)
 - ★ get_code(-Code)
 - ★ get_code(+Stream, -Code)
 - ★ get_char(-Char)
 - ★ get_char(+Stream, -Char)
 - ★ get0(-Char)
 - ★ get0(+Stream, -Char)
 - ★ get(+Stream, -Char)
 - ★ peek_byte(-Byte)
 - ★ peek_byte(+Stream, -Byte)
 - ★ peek_code(-Code)
 - ★ peek_code(+Stream, -Code)
 - ★ peek_char(-Char)
 - ★ peek_char(+Stream, -Char)
 - ★ peek_string(+Stream, +Len, -String)
 - ★ skip(+Code)
 - ★ skip(+Stream, +Code)
 - ★ get_single_char(-Code)

Trabalho II

- **Trabalho II:** Dada a base de dados abaixo, implemente os seguintes predicados:

- ▶ Predicado que lista os clientes de uma filial específica
 - ▶ Predicado que lista os clientes de toda a empresa
 - ▶ Predicado que lista as profissões dos clientes de toda a empresa
 - ▶ Predicado que lista todos os bens adquiridos por um cliente
 - ▶ Predicado que conta quantos cliente a empresa possui
 - ▶ Predicado que calcula o custo médio dos bens

- Continuação:

- Base de Dados:

```
filial('POA', [2345,2346,2347]).  
filial('Curitiba',[3567,3568,3569,3570]).
```

```
cliente(2345, 'Carlos', 45, 'Médico', [bem(c, 'Mercedes', 80000)]).  
cliente(2346, 'Rogério', 23, 'Estudante', [bem(m, 'Kawasaki 1000',  
24000)]).  
cliente(2347, 'Marcelo', 52, 'Advogado', [bem(c, 'Audi A4', 57000),  
bem(c, 'Ford Focus', 25000)]).  
cliente(3567, 'Ana', 35, 'Modelo', [bem(c, 'BMW Serie 3', 42000)]).  
cliente(3568, 'Marcos', 20, 'Operário', [bem(c, 'Fiat Uno', 20000)]).  
cliente(3569, 'José', 45, 'Professor', [bem(c, 'Fiat Pálio 1.6', 30000)]).  
cliente(3570, 'João Carlos', 38, 'Médico', [bem(c, 'Ford Fox', 35000),  
bem(m, 'CB 450', 15000)]).
```

1 Unidade 1: Administrativo

2 Unidade 2: Introdução

3 Unidade 3: Prolog: Tipos de Dados

4 Unidade 4: Prolog: Predicados Pré-definidos

4.1: Predicados Interpretador

4.2: Predicados Listas

4.3: Predicados Lógicos

4.4: Predicados de Teste de Tipos

4.5: Predicados Entrada e Saída

4.6: Predicados de Controle

4.7: Predicados de Criação de Operadores

4.8: Predicados de Conversão de Tipos

4.9: Predicados de Coleta de Soluções

5 Unidade 5: Prolog: Banco de Dados

- Predicados de controle:

- call(:Goal): Executa um objetivo

Exemplo 4.1: : Exemplo de call: call.pl

- repeat: Sempre dá verdadeiro, diferente de true.

Exemplo 4.2: : Exemplo de repeat: ascii.pl

- Para exemplificar o repeat temos primeiro que adiantar a explicação do predicado flag.

Predicado flag(+Key,-Old,+New)

- ★ O predicado flag é um mecanismo para armazenar dados no banco de dados.

- ★ Key é um atomo, inteiro ou termo que é usado para localizar a flag.

- ★ Old é o valor atual da flag, se não houver valor ele é unificado com 0.

- ★ New é o novo valor a ser armazenado.

- Predicados de controle:

- ! (cut): Corta o backtracking

- ★ Quando o Prolog está processando clausulas para frente o cut é desconsiderado.

- ★ Quando o Prolog está fazendo backtracking o cut faz com que o predicado falhe (todo o predicado e não só a clausula sendo executada)

- ★ De forma geral, a não ser que se queira que o backtracking aconteça se usa cut no fim do predicado.

Exemplo 4.3: : Exemplo de cut: predicado max

- **Exercício 4.1:** Faça um predicado que implemente um comando 'for' de uma linguagem procedural. `for(I, L, X)` ; I é o valor de início, L é o limite e X é o valor atual da variável de controle

Observação: Use o cut e o fail para controlar a execução

- **Exemplo:** Exemplo de como será usado:

?- `for(1,4,X), write(X), nl, fail.`

```
1  
2  
3  
fail  
?-
```

- **Exercícios:** Faça predicados para as seguintes tarefas:

Ex 4.2: Converte uma hora no formato "H:M" para minutos

Ex 4.3: Converte um número de minutos em hora no formato "H:M"

Ex 4.4: Testa se uma data está correta (dias × mês × ano Bissextos)¹

¹Ano bissexto são todos os anos múltiplos de 4 exceto os múltiplos de 100 e de 400 ao mesmo tempo

Roteiro I

- 1 Unidade 1: Administrativo
- 2 Unidade 2: Introdução
- 3 Unidade 3: Prolog: Tipos de Dados
- 4 Unidade 4: Prolog: Predicados Pré-definidos
 - 4.1: Predicados Interpretador
 - 4.2: Predicados Listas
 - 4.3: Predicados Lógicos
 - 4.4: Predicados de Teste de Tipos
 - 4.5: Predicados Entrada e Saída
 - 4.6: Predicados de Controle
 - 4.7: Predicados de Criação de Operadores
 - 4.8: Predicados de Conversão de Tipos
 - 4.9: Predicados de Coleta de Soluções
- 5 Unidade 5: Prolog: Banco de Dados

Predicado para criar operadores

- Em Prolog é muito fácil criar operadores novos
- Novos operadores não são associados a nenhuma ação mas servem apenas para manter componentes de estruturas.
- A sintaxe para definir operadores é:
`:- op(<precedência>, <associatividade>, <operador>)`
- A precedência dos operadores é dada por um inteiro. Quanto menor o valor deste inteiro maior a precedência dos operadores

- A associatividade é dada por:
 - ▶ Operadores infixados
 - ★ xfx : sem associatividade)
 - ★ yfx : associatividade à esquerda
 - ★ xfy : associatividade à direita
 - ▶ Operadores pré-fixados
 - ★ fx : sem associatividade
 - ★ fy : associatividade à direita
 - ▶ Operadores pós-fixados
 - ★ xf : sem associatividade
 - ★ yf : associatividade à esquerda

- **Exemplo:** de Operadores pré-definidos:


```
: - op(500, fx, [+,-]).  
:- op(500, yfx, [+,-]).  
:- op(400, yfx, [*,/]).
```

Criação de operadores

- **Exemplo:** de criação de operadores.
Deseja-se poder representar equações da lógica booleana usando operadores, como em: A & ~B (A e não B)
- Definindo operadores:
`: - op(600, yfx, [&, v]).
:- op(500, fy, ~).`
- Agora pode-se usar equações booleanas conforme definidas anteriormente, mas para avaliarmos estas equações precisamos de um predicado de avaliação.

Criação de operadores

- No entanto se quisermos imprimir o valor da expressão devemos usar uma variável para atribuir o valor e um operador de atribuição.
`: - op(1000, fx, ':=').`

```
eval(A := B) :- eval(B), A = true, !.  
eval(A := B) :- A = false, !.
```

Roteiro I

- 1 Unidade 1: Administrativo
- 2 Unidade 2: Introdução
- 3 Unidade 3: Prolog: Tipos de Dados
- 4 Unidade 4: Prolog: Predicados Pré-definidos
 - 4.1: Predicados Interpretador
 - 4.2: Predicados Listas
 - 4.3: Predicados Lógicos
 - 4.4: Predicados de Teste de Tipos
 - 4.5: Predicados Entrada e Saída
 - 4.6: Predicados de Controle
 - 4.7: Predicados de Criação de Operadores
 - 4.8: Predicados de Conversão de Tipos
 - 4.9: Predicados de Coleta de Soluções
- 5 Unidade 5: Prolog: Banco de Dados

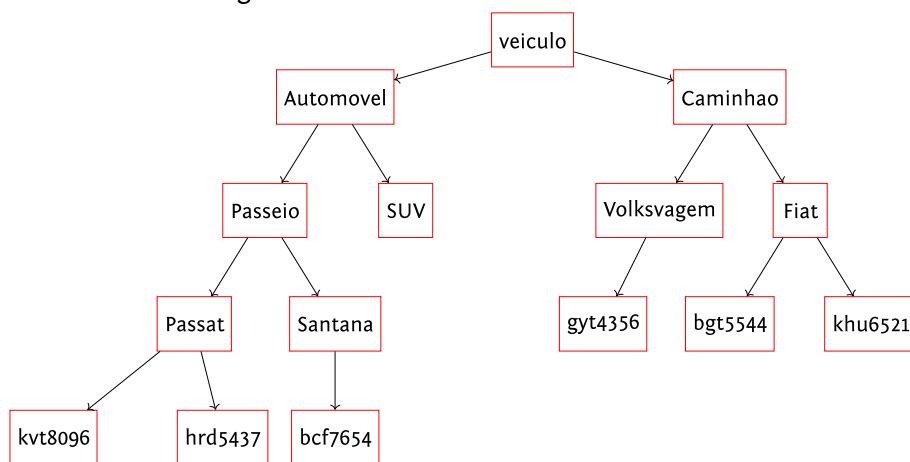
- Predicados para conversão entre tipos
 - ▶ atom_codes(?Atom, ?CodeList)
 - ▶ atom_chars(?Atom, ?CharList)
 - ▶ char_code(?Atom, ?ASCII)
 - ▶ number_chars(?Number, ?CharList)
 - ▶ number_codes(?Number, ?CodeList)
 - ▶ name(?AtomOrInt, ?CodeList)
 - ▶ atom_string(?Atom, ?String)
 - ▶ number_string(?Number, ?String)
 - ▶ term_string(?Term, ?String)
 - ▶ int_to_atom(+Int, +Base, -Atom)
 - ▶ int_to_atom(+Int, -Atom)
 - ▶ term_to_atom(?Term, ?Atom)
 - ▶ atom_to_term(+Atom, -Term, -Bindings)
- Exemplo 4.4: `?- atom_to_terms('casados(jose,X)',casados(Y,Z),L).`
`Y = jose`
`Z = _G429`
`L = ['X'=_G429]`

- Outros Predicados com strings:
 - ▶ string_chars(?String, ?Chars)
 - ▶ string_codes(?String, ?Codes)
 - ▶ text_to_string(+Text, -String)
 - ▶ string_length(+String, -Length)
 - ▶ string_code(?Index, +String, ?Code)
 - ▶ get_string_code(+Index, +String, -Code)
 - ▶ string_concat(?String1, ?String2, ?String3)
 - ▶ split_string(+String, +SepChars, +PadChars, -SubStrings)
Exemplo 4.5: `split_string("/ab/.b./c/d///de", ".", "/", L).`
`L = ["ab", "b", "c/d", , "de"].`

- Outros Predicados com strings:
 - ▶ sub_string(+String, ?Before, ?Length, ?After, ?SubString)
 - ▶ atomics_to_string(+List, -String)
 - ▶ atomics_to_string(+List, +Separator, -String)
 - ▶ string_upper(+String, -UpperCase)
 - ▶ string_lower(+String, LowerCase)
 - ▶ read_string(+Stream, ?Length, -String)
 - ▶ read_string(+Stream, +SepChars, +PadChars, -Sep, -String)

- **Exercício 4.5:** : Represente a árvore abaixo em Prolog e faça predicados para percorre-la usando:

- ▶ Busca em profundidade
- ▶ Busca em largura



Roteiro I

- 1 Unidade 1: Administrativo
- 2 Unidade 2: Introdução
- 3 Unidade 3: Prolog: Tipos de Dados
- 4 Unidade 4: Prolog: Predicados Pré-definidos
 - 4.1 Predicados Interpretador
 - 4.2 Predicados Listas
 - 4.3 Predicados Lógicos
 - 4.4 Predicados de Teste de Tipos
 - 4.5 Predicados Entrada e Saída
 - 4.6 Predicados de Controle
 - 4.7 Predicados de Criação de Operadores
 - 4.8 Predicados de Conversão de Tipos
 - 4.9 Predicados de Coleta de Soluções
- 5 Unidade 5: Prolog: Banco de Dados

- Predicados para testar/coletar soluções:
 - ▶ forall(+Cond,+Action): Executa uma ação para todos as possíveis instâncias da condição
 - ▶ findall(+Var, +Goal, -Bag): Cria uma lista com todas as instâncias da variável de controle que satisfazem o objetivo
 - ▶ bagof(+Var, +Goal, -Bag) Cria uma lista com todas as instâncias da variável de controle que satisfazem o objetivo para cada instância das outras variáveis
 - ▶ setof(+Var, +Goal, -Bag) Cria uma lista ordenada e sem duplicação com todas as instâncias da variável de controle que satisfazem o objetivo para cada instância das outras variáveis
 - ▶ **Observação 1:** Em bagof e setof as variáveis podem ser eliminadas da unificação (não assumem valores) usando o operador '^'.
 - ▶ **Observação 2:** Ver exemplos destes predicados em collect.pl

Trabalho III

- **Trabalho III:** Fazer um banco que modele a estrutura de uma faculdade com professores, alunos, e disciplinas com seus respectivos horários.
- Faça predicados para responder a perguntas sobre este banco de dados:
 - ▶ Quais disciplinas um professor/aluno leciona/assiste
 - ▶ Quais são os horários livres de um professor/aluno
 - ▶ Quais são os horários comuns entre um professor e um aluno
 - ▶ Quais são os horários livres comuns entre um professor e um aluno
 - ▶ Quantas disciplinas são ministradas na parte da manhã
 - ▶ Invente 3 outras perguntas interessantes.

Faça um predicho que teste os predicados criados.

Observação: Os predicados de coleta de soluções devem ser usados, embora em alguns casos outros predicados serão também necessários.

- 1 Unidade 1: Administrativo
- 2 Unidade 2: Introdução
- 3 Unidade 3: Prolog: Tipos de Dados
- 4 Unidade 4: Prolog: Predicados Pré-definidos
 - 4.1: Predicados Interpretador
 - 4.2: Predicados Listas
 - 4.3: Predicados Lógicos
 - 4.4: Predicados de Teste de Tipos
 - 4.5: Predicados Entrada e Saída
 - 4.6: Predicados de Controle
 - 4.7: Predicados de Criação de Operadores
 - 4.8: Predicados de Conversão de Tipos
 - 4.9: Predicados de Coleta de Soluções
- 5 Unidade 5: Prolog: Banco de Dados

- Existem duas maneiras de armazenar informações em bancos de dados em Prolog
 - ▶ Banco de clausulas: neste banco as novas clausulas (clausulas dinâmicas) adicionadas são compiladas e funcionam como se fossem clausulas do programa original (clausulas estáticas).

Observação: O Prolog SWI não aceita remover clausulas estáticas.
 - ▶ Banco de dados: neste banco os dados são associados a uma chave que pode ser um átomo, um inteiro ou um termo. Mais de um dado pode ser associado a mesma chave.

Banco de Clausulas

- Trabalhando com o banco de clausulas.
 - ▶ Incluindo no BC:
 - ★ assert(+Term)
 - ★ asserta(+Term)
 - ★ assertz(+Term)
 - ▶ Removendo do BC:
 - ★ abolish(+Name, +Arity)
 - ★ retract(+Term)
 - ★ retractall(+Head)

Banco de Dados

- Trabalhando com o banco de dados.
 - ▶ Incluindo no BD:
 - ★ record(+Key, +Term, -Reference)
 - ★ record(+Key, +Term)
 - ★ recordz(+Key, +Term, -Reference)
 - ★ recordz(+Key, +Term)assert(+Term)
 - ▶ Pesquisando no BD:
 - ★ recorded(+Key, -Value, -Reference)
 - ★ recorded(+Key, -Value)
 - ▶ Removendo do BD:
 - ★ erase(+Reference)
 - ▶ Alterando o BD:
 - ★ flag(+Key, -Old, +New)

Observação: Pode ser usado para implementar contadores em Prolog (ver exemplo ascii.pl)

- **Trabalho IV:** Programa Animal:

- ▶ Este programa implementa um jogo de adivinhação.
- ▶ É feita uma pergunta ao jogador e baseado nas respostas do jogador o programa tenta adivinhar que animal o jogador está pensando.
- ▶ Caso o programa não consiga adivinhar ele incorpora o animal ao seu banco de dados de forma a poder adivinhar o animal da próxima vez. Isto é feito perguntando-se ao jogador qual era o animal que ele estava pensando e qual característica diferencia o animal pensado e o animal sugerido pelo programa.