Tipos de Dados

4.3 Tipos de dados estruturados (TDE)

- Tipos e objetos de dados estruturados (ODE)
- Especificação de tipo de dados estruturado (TDE)
- Implementação de TDE
- Declaração e checagem de tipo para TDE
- Vetores e Arranjos
- Registros
- Listas
- Cadeia de caracteres
- Ponteiros e OD construído pelo programador
- Conjuntos
- Objetos executáveis
- Arquivos

Objeto de dados estruturado (ODE) e tipos de dados estruturados (TDE)

- ODE: construído como um agregado de outros OD, chamados componentes.
  - componente pode ser elementar ou um ODE.
    - Exemplo: componente de arranjo: outro arranjo, registro, caracter, real, inteiro, ponteiro.
- Semelhança com OD elementares
  - maior *complexidade*: especificações, implementação, declarações, checagem de tipos e ligações de atributos:
    - como indicar componentes e seus relacionamentos de modo que a seleção de um componente da estrutura seja *direta*?
    - algumas Operações envolvem aspectos da gerencia de armazenamento não presentes em OD elementares.

Especificação de TDE

Principais atributos de TDE

- Número de componentes invariável ou não:
  - ODE de tamanho fixo (invariável)
    - em geral arranjos, registros, *strings*.
  - ODE de tamanho variável (variável durante a execução).
    - pilhas, listas, conjuntos, tabelas, arquivos, *strings*.
- Tipo de cada componente
  - homogêneo: seus componentes são do mesmo tipo
    - em geral arranjos, strings, conjuntos, arquivos.
  - heterogêneo: possui componentes de tipos diferentes.
    - registros, listas.

Principais atributos de TDE

Nome usado para selecionar componentes

- Arranjos
  - É um subscrito *inteiro* ou uma *seqüência* de subscritos

```
type ThorasDia = array [1..24] of integer;
     TdiasMes = array [1..31] of ThorasDia;
     TmesesAno = array [1..12] of TdiasMes;
(* alternativamente *)
     ThorasAno  = array [1..12, 1..31, 1..24] of integer;
var dados: TmesesAno;   dadosM: ThorasAno;
```

  - Mapeamento

    m: índices → componentes

    ThorasAno: {1,...,12} x {1,...,31} x {1,...,24} → I

  - Seleção de componentes

```
dados [im,id,ih] := dadosM [im,id,ih] + valorObservado;
```

Principais atributos de TDE

Nome usado para selecionar componentes

- Registros
  - É um identificador ou ponteiro definido pelo programador
  - Declaração

    type tdia = 1..31; tmes = 1..12; tano=1900..2050;
    type tdata = record dia: tdia; mês: tmes; ano: tano end ;
    var data1, data2: tdata;
  - Seleção

    data1.dia := 15; data1.mês = 5; data1.ano = 2000;
    data2 := data1;
  - Produto cartesiano não corresponde a um índice:

    tdia x tmes x tano = {(x, y, z) | x←tdia, y ← tmes, z ← tano}

Principais atributos de TDE

Nome usado para selecionar componentes

- Listas (caso geral de filas e pilhas)
  - Acesso para componentes específicos:
    - previstas operações para mudar os componentes acessíveis.
  - Declaração

```
type tinfo=record nome: string [30]; fone: string[15] end;
     tnodo = record info: tinfo ;  prox: ^tnodo end ;
     tfila = record primeiro, ultimo: ^tnodo end;
var pilha: ^tnodo; fila: tfila;
```

  - Seleção

```
new(pilha) ;
pilha.info.nome := “Wagner Teixeira”;
pilha.info.fone := “(61)307-2702”;
pilha.prox := nil;
```

Principais atributos de TDE

- Número de componentes (variável ou fixo)
- Tipo de cada componente (homo ou heterogêneo)
- Nome usado para selecionar componentes
- Número máximo de componentes
  - TDE de tamanho variável: pode-se limitar o máximo de componentes permitido: pilha, string, arquivo, etc.
- Organização dos componentes
  - Seqüência linear (mais comum)
    - Usada em vetores, registros, strings, pilhas, listas, arquivos.
  - Formas multidimensionais
    - Usada arranjos multidimensionais, registros com componentes registros e listas com componentes listas.

Especificação de TDE

Operações em TDE

- Seleção de cada componente de per si:
  - Aleatória: acessa um componente arbitrário.
    - subscrito em vetores, tal como em mat [4,3].
  - Seqüencial: acessa os elementos em ordem predefinida
    - **for** l:=1 **to** nl **do**
      for k:=1 **to** nk **do** mat [l,k] := ...
  - Referência x seleção (i.e. ter acesso)
    - V[5] := ... implica em duas operações
      - referência: retorna ponteiro para a localização de V.
      - seleção: usa o ponteiro e o subscrito para ter acesso ao 5o elemento de V.
    - referenciar: determinar a localização atual (l-value) do OD.
    - selecionar: escolher um componente do OD.

Especificação de TDE

Operações em TDE

- Operações sobre todos os componentes da estrutura:

  Registrox : = Registroy; (* sem detalhar componentes *)

  MatA := MatB;

  VetorC := VetorA + VetorB ; (* soma de vetores *)

  - APL e SNOBOL4 são muito ricas nesse tipo de operação.
- Inserção e eliminação de componentes:
  - mudam o número de componentes $\Rightarrow$ impacto sobre a RA e sobre a gerência de armazenagem
- Criação e destruição de ODE:
  - impacto sobre a gerência de armazenagem.

Tipos de Dados Estruturados

Implementação de TDE

- Fatores a considerar:
  - Representação da armazenagem (criadas por software):
    - eficiência na seleção de componentes do ODE.
    - eficiência global da gerência de armazenamento.
  - Operações sobre a RA (simuladas por software)
- Representação de Armazenagem
  - Área para componentes e descritor opcional.
  - Representação seqüencial ⇒ estruturas tamanho fixo
    - único bloco continuo que inclui descritor e componentes.
  - Representação encadeada ⇒ estruturas tamanho variável
    - vários blocos não contínuos ligados por ponteiros.
    - descritor no bloco inicial e os componentes nos demais.

Implementação de TDE

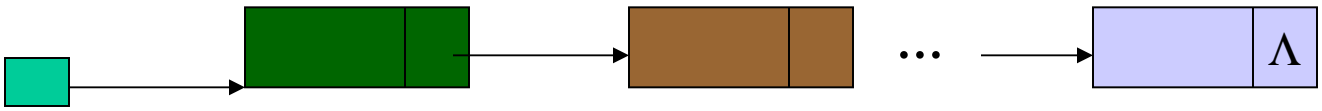
Implementação de operações em TDE

- TDE requer acesso aleatório e seqüencial eficientes
- RA seqüencial
  - seleção aleatório: fórmula ⇔ endereço de base + offset:
    var A: **array** [-1..10] **of string** [5];
    l-value(A[k]) = l-value(A) + (k-(-1))*5.
    em geral, local(A[k]) = l-value(A)+(k-LB)*E, onde
    LB = limite inferior do subscrito de A,
    E = tamanho de um elemento de A.
  - seleção seqüencial: endereço atual + E.
  - domínio de subscritos
    - pode ser qualquer, por exemplo -5..-2; -3..20, 1990..2000.
    - a expressão de seleção acima funciona para qualquer caso.

Implementação de operações TDE

RA encadeada

- Seleção aleatória
  - segue os ponteiros desde o bloco inicial até o componente desejado.
  - a posição do ponteiro dentro de cada componente deve ser conhecida.
- Seleção seqüencial
  - consiste em acessar o primeiro componente e seguir o ponteiro desse para o próximo e assim por diante.



Implementação de TDE

Gerência de armazenagem e TDE

- *Tempo de vida* de um ODE
  - início: bloco de armazenamento alocado e RA iniciada.
  - término: fim da ligação para seu local de armazenagem.
- ODE de tamanho variável
  - cada componente tem seu próprio tempo de vida.
  - início: quando ele é criado e inserido na estrutura.
  - término: quando ele é eliminado da estrutura.
- *Caminho de acesso* (criado junto com o ODE)
  - nome (em um ambiente de referência) ou ponteiro (em alguma estrutura pré-existente). Podem existir vários.
  - a interação tempo de vida e caminho de acesso pode criar *lixo* ou *referência pendente*.

Implementação de TDE

Gerência de armazenagem e TDE

- Lixo
  - ODE sem nenhum caminho de acesso, mas ainda ligado ao seu local de armazenagem ⇒ existente e inacessível.
  - problema: alocação de desnecessária de memória.
- Referência pendente (dangling) (mais sério)
  - caminho de acesso para um ODE que não existente.
    - ODE morre e é *liberado* o bloco de armazenagem para realocação de outros objetos, mas *sem* destruir todos caminhos de acesso associados ao ODE eliminado.
  - problema: pode violar a segurança de checagem de tipo
    - uma atribuição pode alterar um ODE de tipo diferente que agora ocupe o mesmo local de armazenamento.

Tipos de Dados Estruturados

Declarações e checagem de tipo para TDE

- RA e seleção
  - declaração de um ODE permite definir sua RA e a fórmula adequada de seleção dos componentes.

    A: **array** [1..10,-5..5] **of real**;

    1. o tipo de dado: arranjo
    2. o número de dimensões: 2
    3. domínio do primeiro subscrito: 1 a 10
    4. domínio do segundo subscrito: -5 a 5
    5. número de componentes: 110 (10 x 11)
    6. tipo de cada componente: real

    $\Rightarrow$ RA seqüencial: 10 blocos de 11 valores reais cada.

    seleção: l-value(A[j,k])=l-value(A)+[(j-1)*11+(k+5)]*E

Declarações e checagem de tipo TDE

Questões relativas a seleção de componente

- Existência de um componente selecionado
  - tipos dos argumentos da seleção certos: a fórmula pode levar a um componente não existente (subscript range).
    - requer checagem *dinâmica* $\Rightarrow$ testar existência componente
    - *sem* checagem dinâmica: similar a erro de tipo (endereço obtido pode conter qualquer coisa que, se usada como argumento de operação, leva a resultados imprevisíveis).
- Tipo do componente selecionado
  - com ponteiro, nem sempre é garantido que o objeto apontado existe (pois pode haver referência pendente).
  - a checagem estática apenas garante que *se* o ODE *existir*, é do tipo correto.

Tipos de Dados Estruturados

Vetores e arranjos

- Vetor
  - estrutura de tamanho fixo com componentes do mesmo tipo, organizada como uma seqüência linear simples.
  - RA e atributos
    - tipo de dados: vetor
    - número de componentes
    - subscritos válidos: [LB (menor), UB (maior subscrito)]
    - tipo de dados de cada componente
    - tamanho de cada componente
    - componentes: A[LB], A[LB+1], ... A[UB]
  - **var** V:**array** [-5..20] **of real**; A: **array**[tcores] **of integer**;

    float V [10]; /* C*/

Vetores e Arranjos

Operações sobre vetores

- Subscrição
  - seleciona um componente (retorna o seu l-value).

    l-value(A[I]) = l-value(A) +(I-LB)*E
- Outras
  - criar e eliminar vetores.
  - atribuir valores para componentes do vetor.
  - operações aritmética com pares de vetores.
  - inicialização de vetores (algumas LP).

    APL fornece um conjunto rico de operações com vetores
  - inserção e eliminação de componentes não são implementadas.

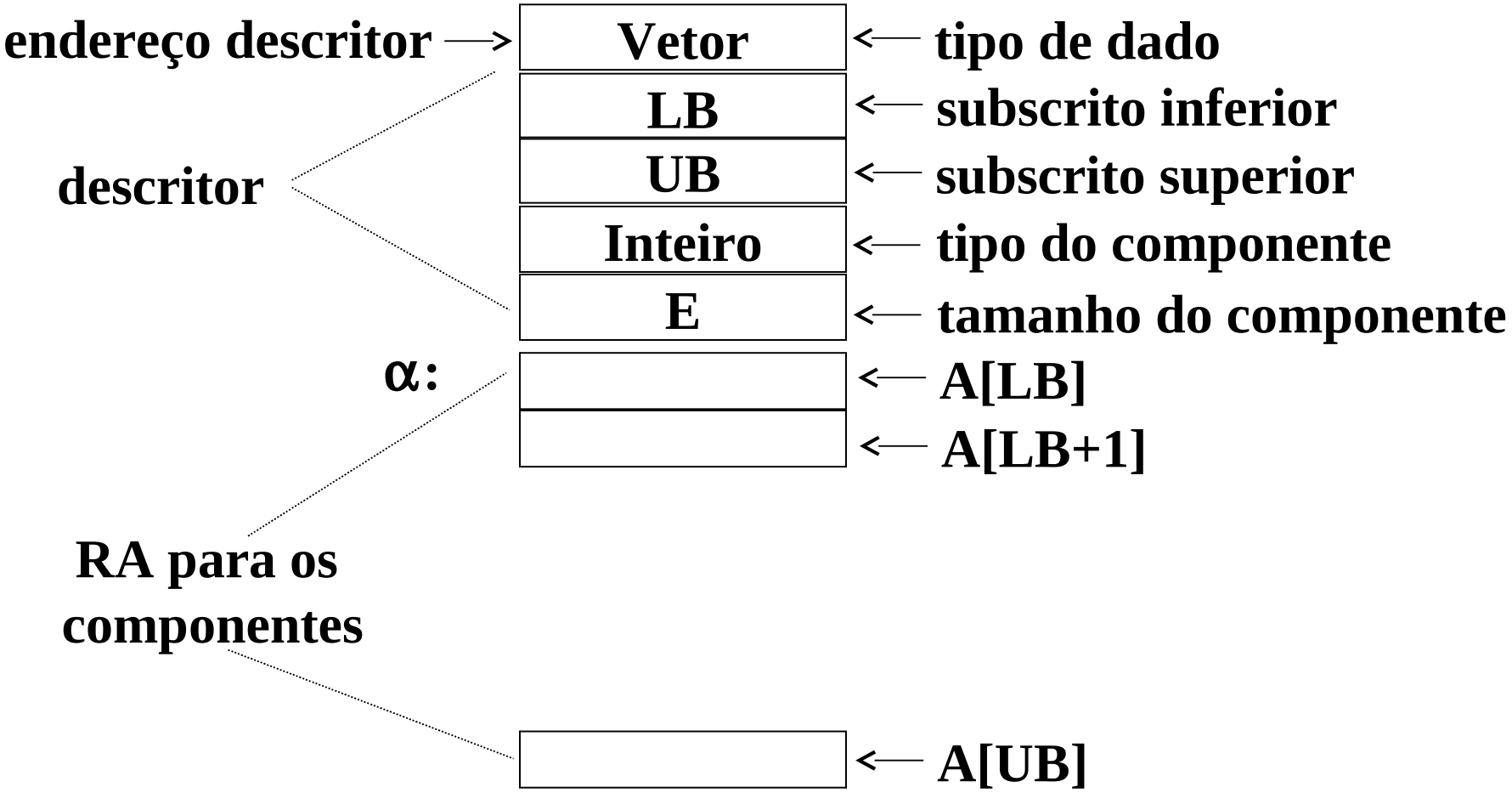
Vetores e Arranjos

Implementação de vetores

- Vetores são objetos homogêneos e de tamanho fixo
  - a quantidade e posição dos componentes é invariante.
  - RA mais adequada: armazenagem seqüencial
  - Atributos necessários para checagem dinâmica
    - Os únicos requeridos em tempo de execução são: α (endereço de base), E (tamanho de cada componente) e LB e UB (limites inferior e superior).
- Seleção
  - l-value(A[j]) = α +(j-LB)*E = (α-LB*E)+(j*E)
    - Se α for conhecido então l-value(A[j]) = VO + j*E, onde VO≡ (α-LB*E) é uma constante,
    - Em C, l-value(A[j]) = α +j*E porque LB = 0.

Vetores e Arranjos

Implementação de vetores



Vetores e Arranjos

Implementação de Vetores

- Na criação da RA
  - Aloque área de memória de tamanho D + N x E. Onde
    - D é o tamanho da área para o descritor do vetor
    - N é a quantidade de elementos
    - E é o tamanho de um elemento.
  - Calcule a origem virtual VO = α - LB x E
- Na seleção do j-ésimo componente
  - l-value(A[j])=VO + j x E
  - Verifique erro de subscrito: LB $\leq$ j $\leq$ UB.
- Checagem estática
  - Não requer representar tipos, apenas: VO, LB, UB, E, e os elementos A[LB], A[LB+1], ..., A[UB].

Vetores e Arranjos

Arranjos multidimensionais

- Arranjos n-dimensionais tem componentes de (n-1) dimensões.
  - vetor de vetores ....
- Especificação e Sintaxe:
  - um subscrito para cada dimensão

    Pascal: B: **array**[LB_1..UB_1,LB_2..UB_2, ... ,LB_n..UB_n] **of** t1;

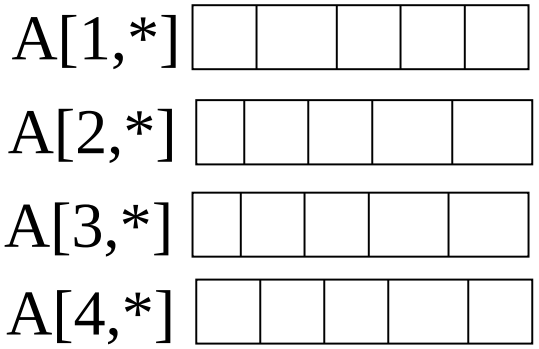
    C: t1 B[N_1][N_2][N_3]...[N_n];

    Fortran: t1 B(N_1,N_2,N_3,...,N_n)
- A seleção de um componente requer o informe de um subscrito para cada dimensão.

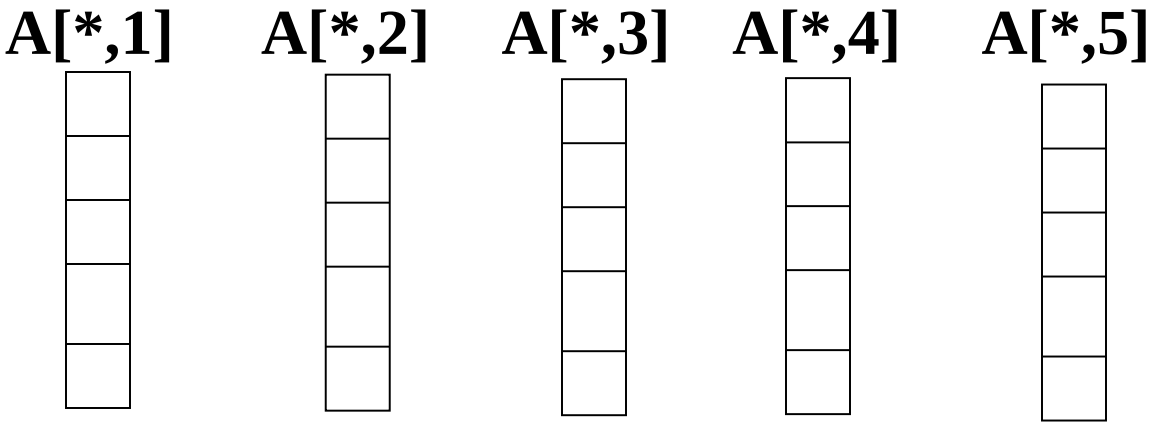
Vetores e Arranjos
Multidimensionais: implementação

- Ordem de linhas uma matriz é vista como uma coluna de linhas.

Dado o tamanho constante dos elementos e o endereço do 1º elemento, a fórmula de acesso é imediata.



- Ordem de Colunas matriz vista como uma linha de colunas.



Vetores e Arranjos
Multidimensionais: implementação

- Na criação da RA
  - Aloque área de memória de tamanho D + N x E. Onde
    - $N = N_1 \times N_2 \times ... \times N_n$ é a quantidade total de elementos.
- Checagem estática
  - Descritor separado (da área de dados) contendo:
    - $VO, LB_1, UB_1, m_1, LB_2, UB_2, m_2, ... LB_n, UB_n, m_n, E$
    - $A[LB_1,*], A[LB_1+1,*], ..., A[UB_1,*]$
- Seleção
  - Cálculo de multiplicadores (constantes):
    - $m_n = E$; $m_i = (UB_{i+1} - LB_{i+1} + 1) \times m_{i+1}$, $i = (n-1), .., 1$
  - Origem virtual: $VO = \alpha - \Sigma_{i=1}^{n}(LB_i \times m_i)$
  - Endereço de $A[s_1,s_2, ..., s_n]$: $VO + \Sigma_{i=1}^{n}(s_i \times m_i)$

Vetores e Arranjos
Multidimensionais: implementação

- Fatia (slice) ou subarranjo
  - PL/1

    A(*,2) - segunda coluna, B(3,*) - terceira linha, C(3,*,*) - terceiro plano de um arranjo
  - Fortran 90

    A(1:4,2), B(3,1:3), C(3,1:3,1:4) eqüivalente ao exemplo em PL/1, para A(4,3), B(4,3) e C(4,3,4).
  - Implementação imediata, usando o descritor do arranjo

VO	LB_1	UB_1	m_1	LB_2	UB_2	m_2	$\Leftrightarrow$ A(4,3)
α-4	1	4	3	1	3	1	α-4+i*3+j*1

VO	LB_1	UB_1	m_1	$\Leftrightarrow$ A(*,2)	$VO=\alpha-LB_1*S-LB_2*E$
α-2	1	4	3		

Registros

Especificação e sintaxe

- Registro é uma estrutura de dados com um número fixo de componentes de diferentes tipos:
  - estrutura de dados com representação contígua (linear)
    - os componentes podem ser heterogêneos.
    - os componentes possuem identificadores.
  - Exemplo em C

```
struct Tempregado {
   int mat;
   int idade;
   float salário;
   char depto[3] } empregado1;
struct Tempregado empregado2, empregado3;
```

Registros

Especificação e sintaxe

- Seleção de componentes
  - Corresponde a subscrição em arranjos. Utiliza literal e não um número como “subscrito”
    - empregado1.mat
    - empregado1.idade
    - empregado1.salário
    - empregado1.depto
- Atributos
  1. número de componentes.
  2. tipo de cada componente.
  3. identificador (seletor) usado para nomear componente.

Registros

Especificação e sintaxe

- Operações
  - básica: seleção de componente
  - Operação sobre o registro inteiro
    - atribuição de registros com estruturas idênticas

      RegA := RegB;
    - Atribuição dos nomes comuns entre registros (Cobol e PL/1)

      move corresponding RegX **to** RegY.

      RegX = RegY, **by name**;

      Os nomes em RegX comuns a RegY precisam ter os mesmos tipos, mas não precisam estar na mesma ordem.

Registros

Implementação

- RA: único bloco seqüencial de memória
  - componentes são armazenados na seqüência declarada.
  - o registro não requer descritor em tempo de execução.
    - componentes individuais podem precisar de descritor
- Seleção de componentes
  - o nome de cada componente é conhecido estaticamente.

    $\Rightarrow$ sabe-se tamanho e posição de cada componente no bloco.
  - Seleção do I-ésimo componente de R
    - l-value(R.I) = $\alpha + \Sigma^{I-1}_{j=1}$(size(R.j)), onde

      α = l-value(R) e R.j é o j-ésimo componente de R.
    - $K_I = \Sigma^{I-1}_{j=1}$(size(R.j)) pode ser calculado estaticamente

      $\Rightarrow$ l-value(R.I) = $\alpha + K_I$

Registros

Implementação

- Problemas
  - componentes que precisam começar em certos limites de endereços: inteiros começam no início de uma palavra.
    - Em máquinas endereçáveis a nível de byte (e palavra com 32 bits), o endereço deve ser divisível por 4.
  - filler
    - inserir campos de preenchimento no registro

      struct Tempregado {
      char depto;
      int mat;
      } E1; eqüivale a:
      struct Tempregado {**char** depto; **char** não-Usado[3]; **int** mat;} E1;

Registros

Implementação

- Operações
  - Atribuição RegA := RegB
    - Copia o conteúdo do bloco de RegB para o bloco de RegA (mais eficiente do que fazer múltiplas atribuições!)
  - **move corresponding** (Cobol) e **by name** (PL/1)
    - seqüência de atribuições dos componentes individuais de um registro para outro (os registros podem não serem idênticos!)

Registros

Registros e arranjos c/ componentes estruturados

- Registros podem ter componentes que são arranjos

```
struct Tempregado {
    int mat, idade;
    float salário [12];
    char depto;
} empregado;
```

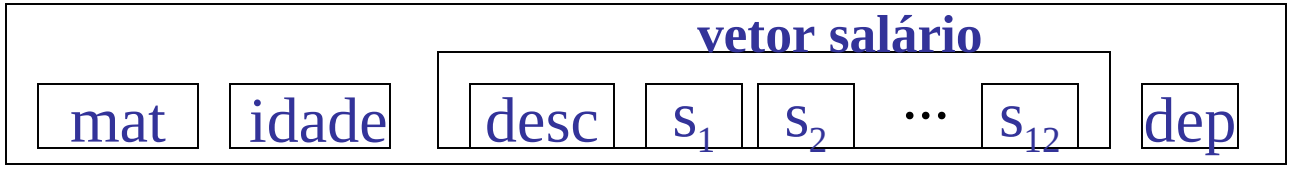
- Arranjos podem ter componentes que são registros

```
struct Tempregado {
    int mat, idade;
    float salário[12];
    char depto;
} empregados [500];
```

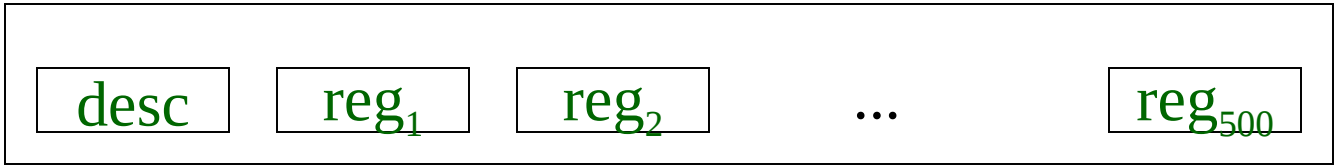
Registros

Registros e arranjos c/ componentes estruturados

Registro empregado



Vetor de registros



Registros

Variantes em PASCAL

```
type Tbens = (casa, apto), Tpropriedade = record
    dono: string[32];
    idade: integer;
    endereço: string[88];
    áreaConstruída: real;
    dependências: integer;
    case tag:Tbens of
          apto: (salãoFesta, salãoJogos: boolean;)
          casa: (áreaLote, áreaGramada: real;
                cobertura: (eternit, telha, zinco, palha));
    end ; (* do case *)
end ; (* do record  *)
var prop1: Tpropriedade;
```

Registros
Variantes

Registro Proprietário

Registros
Variantes

- RA
  - área
    - uma comum a todos os registros.
    - área específica para cada valor do tag (ou discriminante).
    - tamanho fixo: área comum + área do maior variante.
  - layout da área variante depende do tag.
    - variações podem ser vistas como subregistros, dos quais apenas UM estará presente num dado INSTANTE.
    - registro atual: subregistro comum + uma das partes discriminadas.
      - tag armazenado ⇒ tipo união DISCRIMINADA.
      - tag não armazenado (C) ⇒ tipo união LIVRE.

Registros
Variantes em C

```
enum Tbens {casa, apto}; struct Tpropriedade {
    char dono [32]; int idade; char endereço [88];
    float área-Construída; int  dependências;
    enum Tbens tag; union Tvariante {
        struct Tcasa {
            float área-Lote, área-Gramada;
            enum cobertura = {eternit, telha, zinco, palha};
        } casa;
        struct Tapto {
            int salão-Festa, salão-Jogos;
        } apto;
    } ; /* fim da união */
} propriedade;
```

Registros
Variantes

- Seleção: base + offset
  - utiliza seletor, como no caso de registro comum.
    - tempo de vida: diferente do tempo de vida do registro ⇒ ERRO: tentativa de acesso a variante inexistente.
  - checagem dinâmica (∃ componente compatível com tag?)
    - sim: acesso permitido.
    - não: erro similar ao subscript range em arrays.
  - Sem checagem dinâmica (seleção presumida válida)
    - Cobol, PL/1, Pascal: permitem formas de variantes sem tag explícito. Uniões em C não permitem tag.
      - programador garante que o campo selecionado é o certo.
    - selecionar o subregistro errado leva a erros imprevisíveis.

Listas

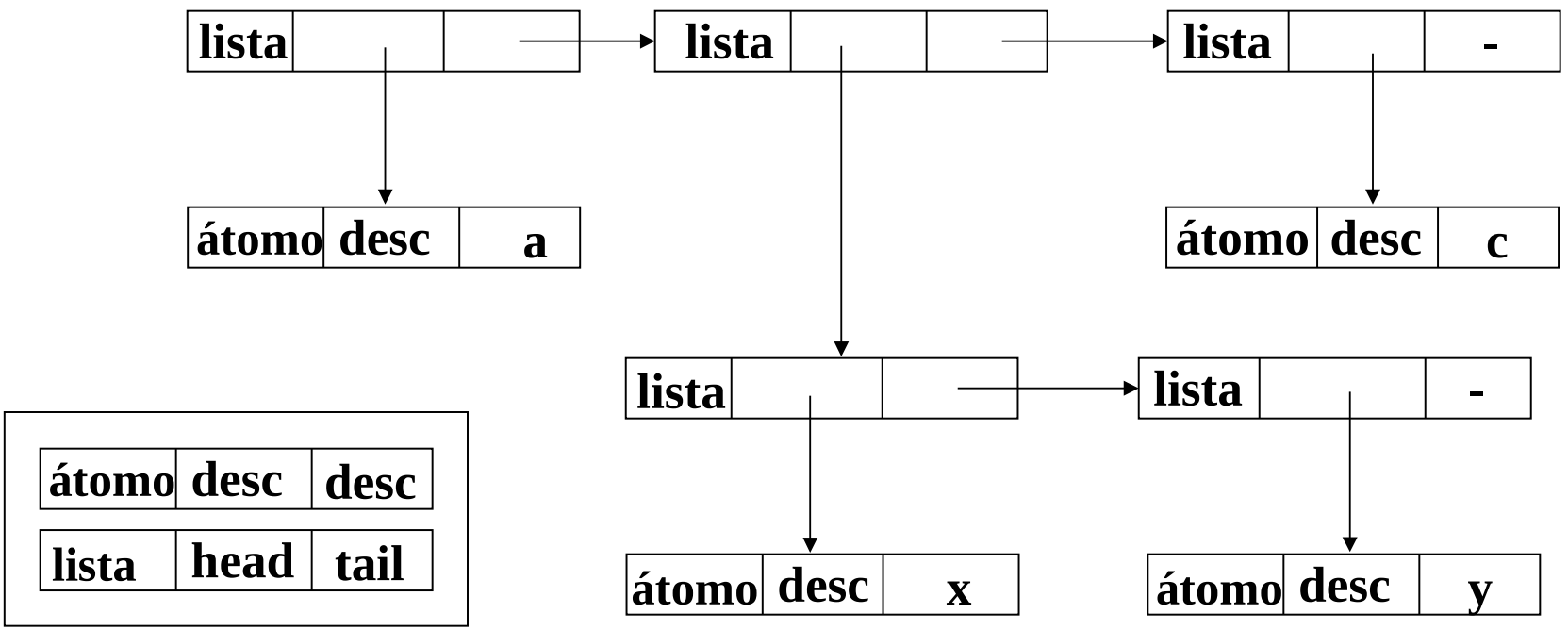
Especificação e Sintaxe

- ODE composto de uma seqüência ordenada de ODE.
- Na lista $(x_1, x_2, x_3, x_4,..., x_i,..., x_n)$
  - x_1 é o primeiro elemento (cabeça da lista).
  - $(x_2, x_3, x_4,..., x_i,.., x_n)$ é a cauda.
  - x_{i-1} precede x_i e x_{i+1} sucede x_i (lista é ordenada).
  - em geral as listas:
    - possuem comprimento variável e não são homogêneas:
      LP que declaram listas, não declaram o tipo dos componentes.
      LP com listas homogêneas, inferem tipos (Haskell, ML, Hugs).
    - RA em vetores não é usual, devido a heterogeneidade $\Rightarrow$ RA organizada na forma de lista encadeada.
      – seus itens são elementos primitivos do tipo atômico (tamanho fixo) ou lista.

Listas

Especificação e sintaxe: estrutura

RA em Lisp: item da lista contém um campo de tipo e dois ponteiros de lista.



Listas

Especificação e sintaxe: Lisp

```
> (+ 1 2 3 4 5 6)
21
> (cons ‘(a b c) ‘(d e f))                ; aloca novo item de lista.
((a b c) d e f)
> (defun quadrado(x) (* x x))            ; declaração de função.
QUADRADO
> (mapcar ‘quadrado ‘(1 2 3 4 5))        ; mapeamento de função.
(1 4 9 16 25)
```

– Avaliação gulosa em LISP ; avalia na ordem em que aparece.

```
> (+ (* 3 4) (/ 15 3) 3)
20
> (apply '+ 1 2 3 '(4 5 6))              ; monta lista e a avalia.
21
> (apply '+ '(1 2 3 4 5 6))
21
```

Listas

Especificação e Sintaxe - Lisp

```
>  (defun vezes(x y) (* x y))            ; mais de um argumento.
VEZES
> (mapcar ‘vezes ‘(1 2 3 4 5 6) ‘(0 1 10 100 1000 10000))
(0 2 30 400 5000 60000)

                                        ; argumento é uma lista.
> (defun times(lista) (* (car lista) (car (cdr lista))))
TIMES
> (mapcar ‘TIMES ‘((1 0) (2 1) (3 10) (4 100) (5 1000)))
(0 2 30 400 5000)
```

Listas

Especificação e Sintaxe: Haskell e Hugs

- **Resolução baseada em casamento de padrões:**

```
Prelude> let q2 x = x*x in map q2 [1,2,3,4,5]
[1,4,9,16,25]
Prelude> map (10 * ) [1,2,3,4,5]
[10,20,30,40,50]
Prelude>  inverso [1,2,3,4,5,6] where {
   inverso xs = inv xs [ ];
   inv [ ] ys = ys;
   inv (x:xs) ys = inv xs (x:ys) }
[6,5,4,3,2,1]
```

Listas

Especificação e sintaxe: Prolog

Montagem da base de regras (cláusulas de Horn):

```
inverso(Xs,Rs) :- inv(Xs,[],Rs).        % maiúscula é variável.
inv([],Ys,Ys).                          % fato.
inv([X|Xs],Ys,Rs) :- inv(Xs,[X|Ys],Rs).
tamanho([],0).
tamanho([X|Xs],T) :- tamanho(Xs,Tc), T is 1 + Tc.
```

Ambiente do interpretador: prompt ?-

```
?- inverso([1,2,3,4,5,6,7,8], Rs).
Rs = [8,7,6,5,4,3,2,1]
?- tamanho([a,b,1,joão,maria,Variável, (R is 2*4+3)], T).
   Variável = _ ,
   R = 11 ,
   T = 7
```

Listas

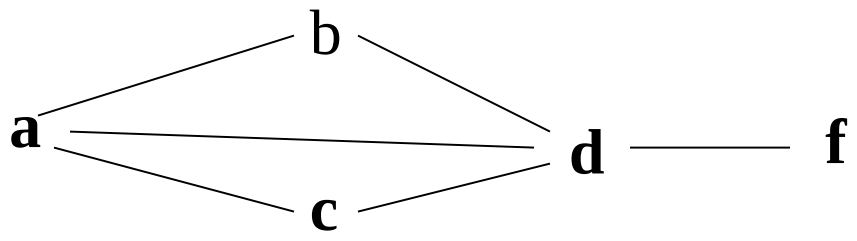
Variações sobre listas

- Pilha (RA seqüencial ou encadeada)
  - lista com seleção, inclusão e exclusão de componentes feita em apenas uma das extremidades.
- Fila (RA seqüencial ou encadeada)
  - lista com inclusão de componentes feita em uma extremidade e seleção e exclusão pela outra.
- Grafo orientado
  - estrutura de dados G = (V, A), onde V é um conjunto de nós e A é um conjunto de arcos (orientado ou não) ligando elementos de V.
- Árvore
  - grafo sem ciclos, com arcos orientados, onde há um nó r (raiz) sem pai e, dado um elemento x de V, há um único caminho orientado de r para x. Um elemento pode ser uma lista ou atômico.

Listas

RA de grafos em listas

- Pares de nós adjacentes
  ((a b) (a c) (a d) (b d) (c d) (d f))



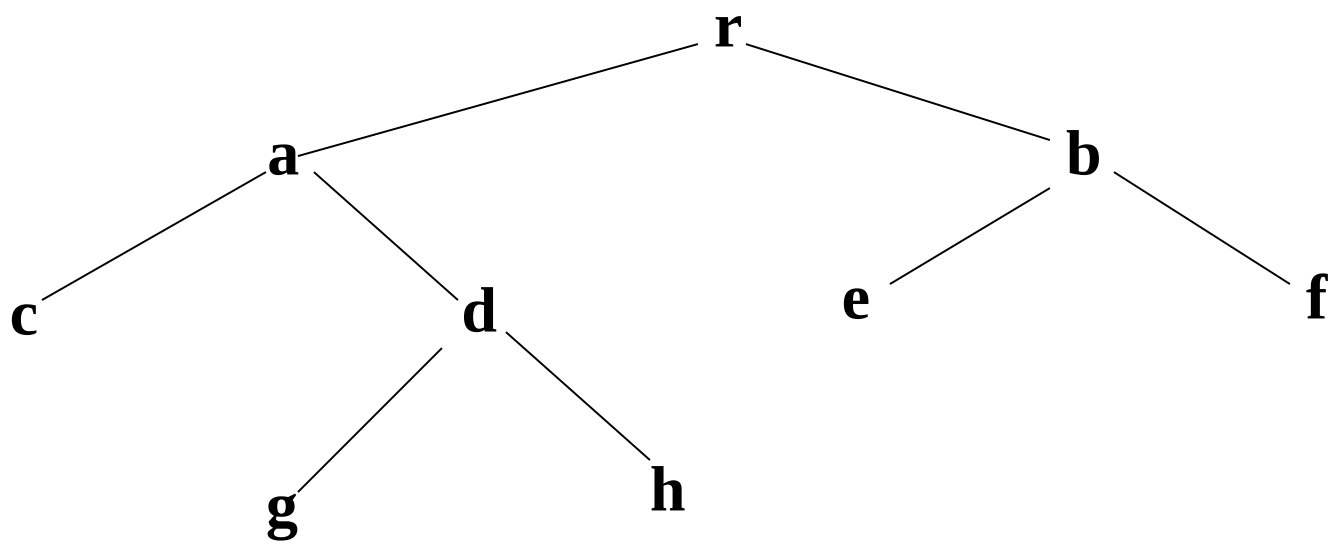
- Nó e seus adjacentes
  ((a b c d) (b a d) (c a d) (d a b c f) (f d))

Listas

RA de árvores em listas

- Pares de nós adjacentes

  ((r a) (r b) (a c) (a d) (b e) (b f) (d g) (d h))



- Nós e seus filhos

  ((r a b) (a c d) (b e f) (c nil) (d g h) (e nil) (f nil) (g nil) (h nil))

Listas

Lista de propriedades

- registro em que o número de componentes (campos) pode variar sem restrição. O nome do campo é chamado *nome da propriedade* e o seu valor é o *valor da propriedade*.

- RA
  - requer armazenar nome e valor da propriedade. Porquê?
  - lista encadeada onde cada nó é um registro com um único campo de informação (alterna *nome* e *valor*), mais o elo de próximo nó.
  - lista de (<nome da propriedade> <valor da propriedade>)

    ((nome “João da Silva”) (idade 35) (esposa “Maria Bela”)
    (filhos (“João da Silva Júnior” “Mariana Bela da Silva”)))