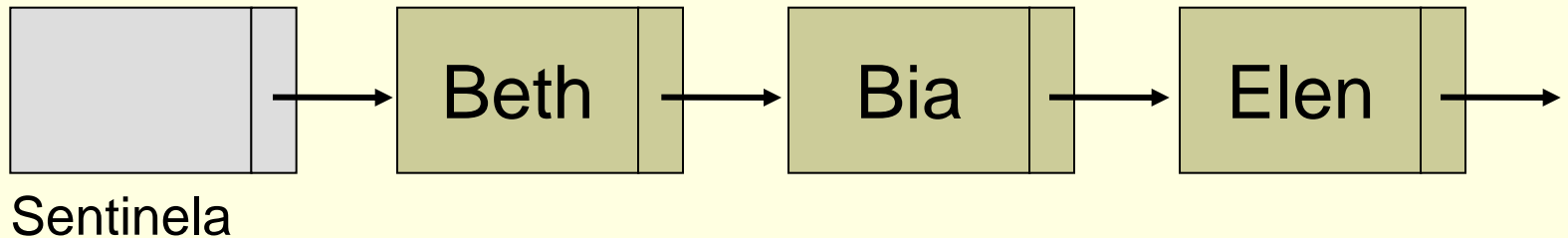


Listas: nós de cabeçalho,  
listas não homogêneas,  
listas generalizadas

SCC-502 – Algoritmos e Estruturas de  
Dados I

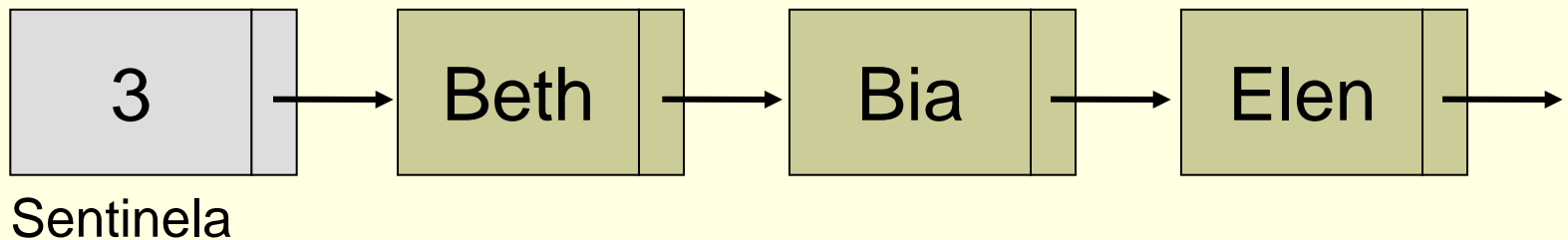
# Lista com nó de cabeçalho

- Nó de cabeçalho
  - *Header*, sentinela, etc.
  
- Para que?



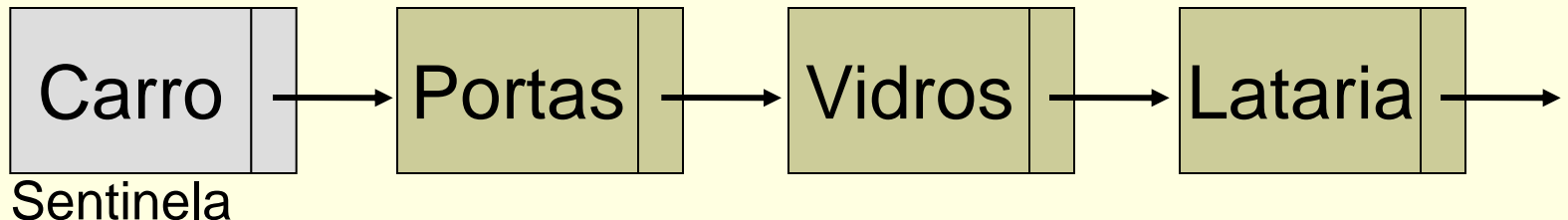
# Lista com nó de cabeçalho

- Possibilidades de uso
  - **Informação global** sobre a lista que possa ser necessária na aplicação
    - Armazenar número de elementos da lista, para que não seja necessário atravessá-la contando seus elementos



# Lista com nó de cabeçalho

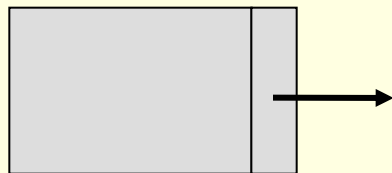
- Possibilidades de uso
  - **Informação global** sobre a lista que possa ser necessária na aplicação
    - Em uma fábrica, guarda-se as peças que compõem cada equipamento produzido, sendo este indicado pelo nó sentinela
    - Informações do voo correspondente a uma fila de passageiros



# Lista com nó de cabeçalho

---

- Possibilidades de uso
  - **Informação global** sobre a lista que possa ser necessária na aplicação
    - Lista vazia contém somente o nó sentinela



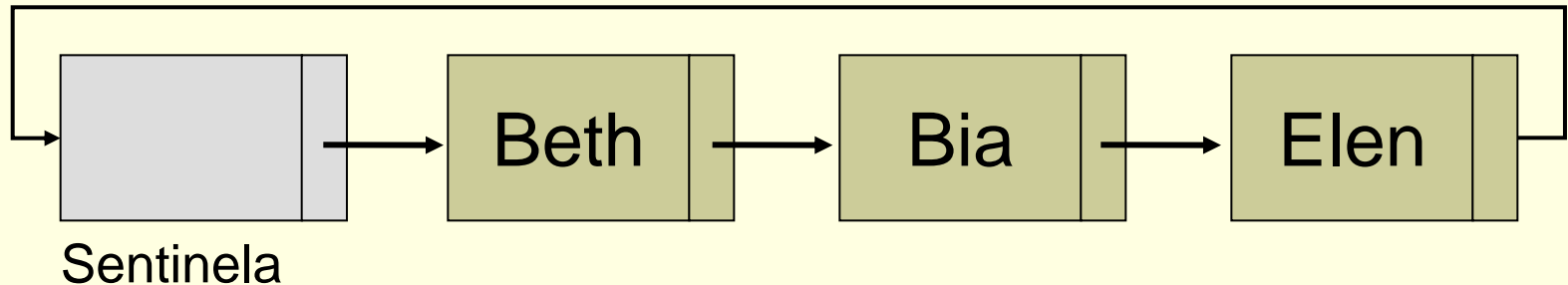
Sentinela

# Lista com nó de cabeçalho

- Possibilidades de uso

- **Lista circular**

- Não existe mais NULL no fim da lista, eliminando-se o risco de acessar uma posição inválida de memória



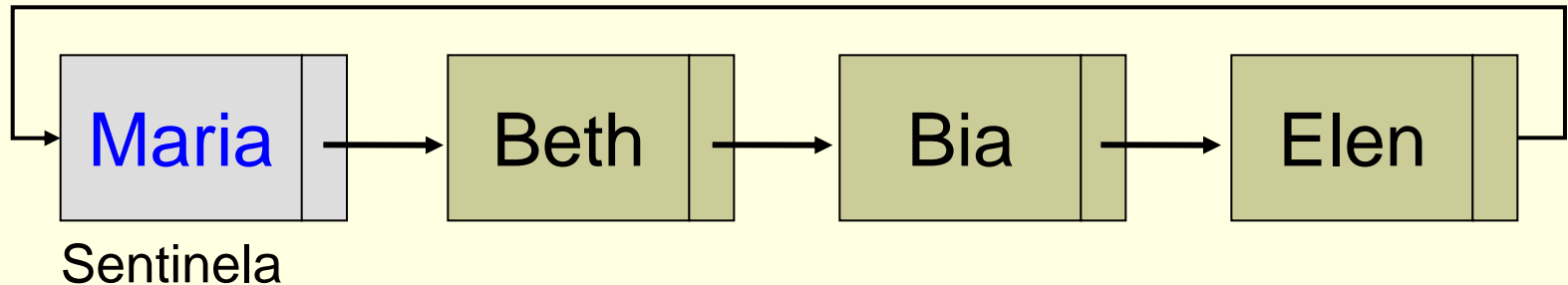
# Lista com nó de cabeçalho

- Possibilidades de uso

- **Lista circular**

- Uso da sentinela para simplificar Busca

- Sempre vai encontrar a chave: se sentinela, então chave não estava na lista.

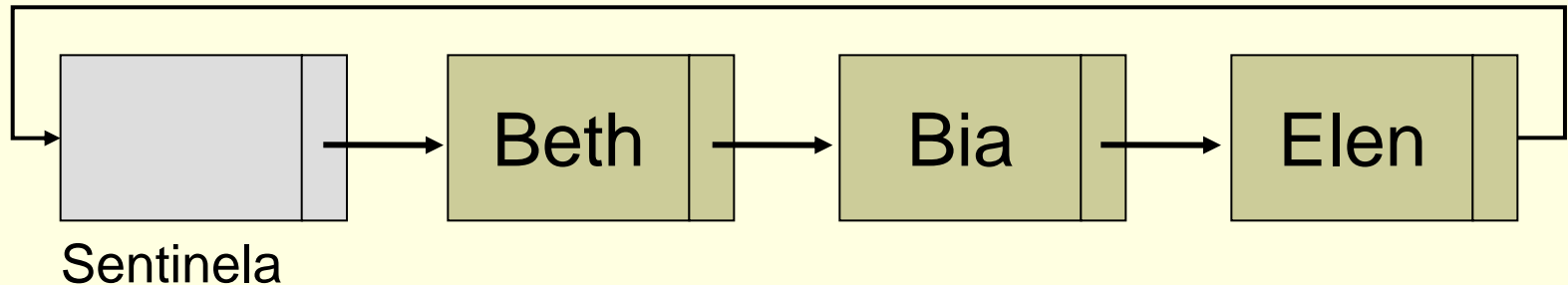


# Lista com nó de cabeçalho

- Possibilidades de uso

- Lista circular

- Como saber qual é o último elemento da lista?



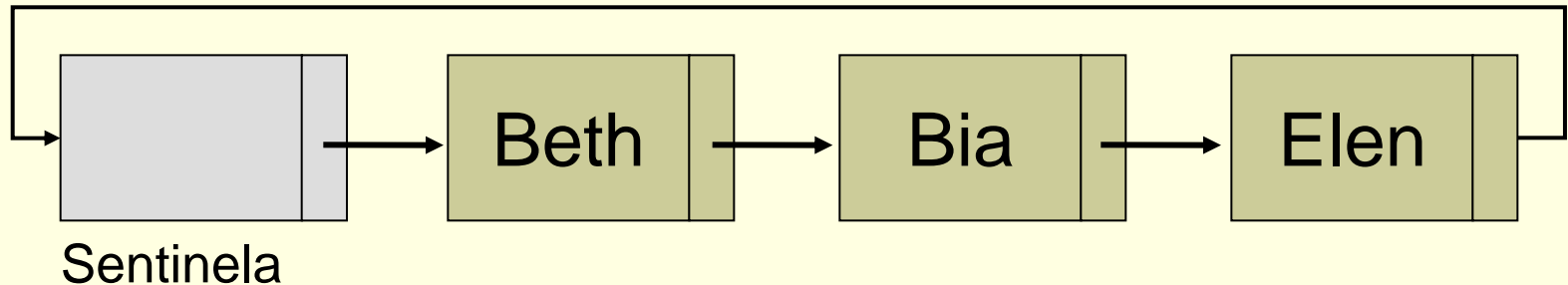


# Lista com nó de cabeçalho

- Possibilidades de uso

- **Lista circular**

- Como saber qual é o último elemento da lista?
      - Ele aponta para o nó sentinela



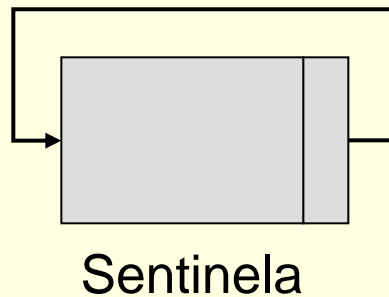
# Lista com nó de cabeçalho

---

- Possibilidades de uso
  - Lista circular
    - Como representar a lista vazia?

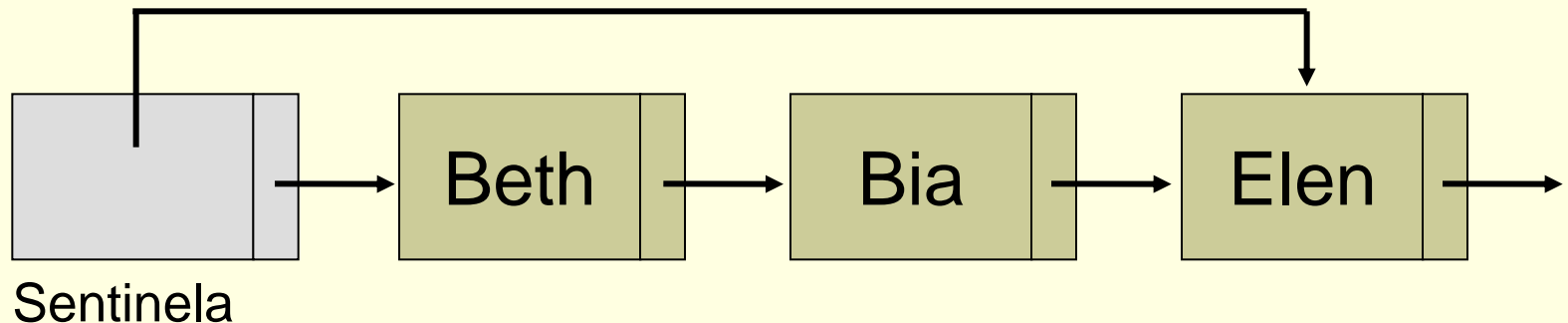
# Lista com nó de cabeçalho

- Possibilidades de uso
  - **Lista circular**
    - Como representar a lista vazia?



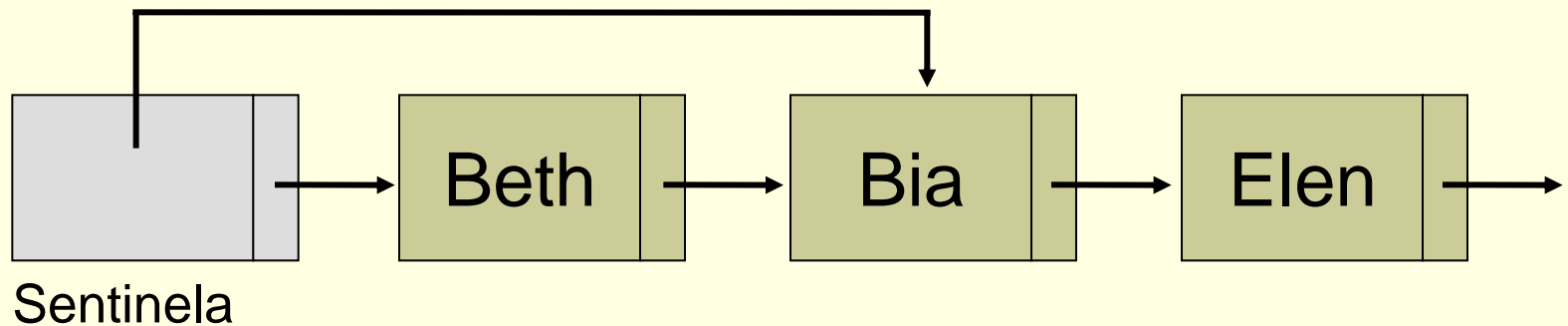
# Lista com nó de cabeçalho

- Possibilidades de uso
  - Informações para uso da **lista como pilha, fila, etc.**
    - Exemplo: em vez de um ponteiro de fim da fila, o nó sentinela pode apontar o fim
      - O campo info do nó sentinela passa a ser um ponteiro
      - Acaba por indicar o início da fila também



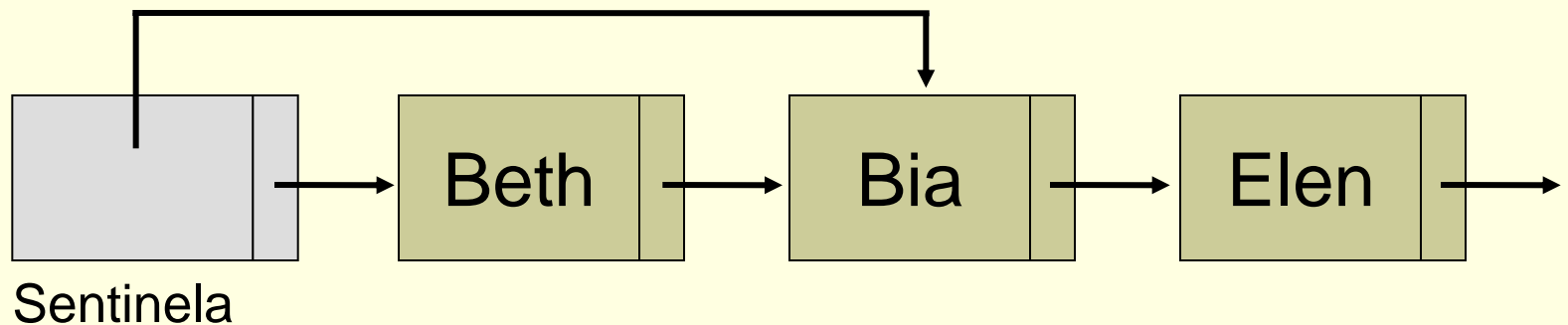
# Lista com nó de cabeçalho

- Possibilidades de uso
  - Indica um **nó específico da lista**
    - Por exemplo, em buscas que são constantemente interrompidas
      - Verificação de pessoas em ordem alfabética: poupa o esforço de se recomeçar ou a necessidade de ter uma variável auxiliar



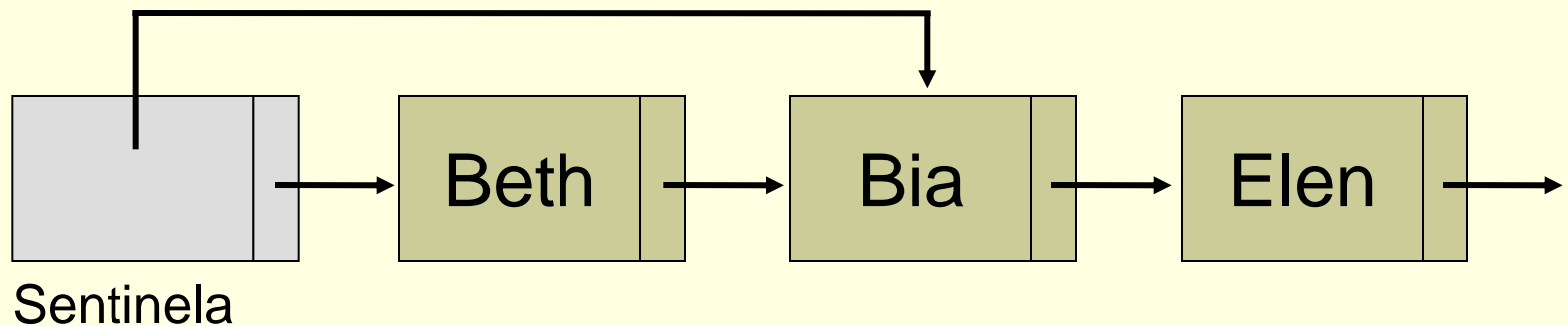
# Lista com nó de cabeçalho

- Possibilidades de uso
  - Nó sentinela com **ponteiro em seu campo info**
    - Vantagem: acesso possivelmente mais direto e imediato
    - Desvantagens? Quais?



# Lista com nó de cabeçalho

- Possibilidades de uso
  - Nó sentinela com **ponteiro em seu campo info**
    - Vantagem: acesso possivelmente mais direto e imediato
    - Desvantagens? Quais?
      - Registro sentinela tem tipo distinto dos demais



# Lista não homogênea

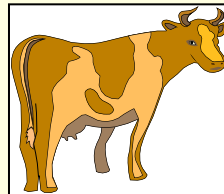
---

- Lista “genérica”
- Possibilidade de usar uma mesma estrutura para armazenar informações diferentes
  - Inteiro, caracter, estrutura, etc.
- Não é necessário definir blocos de memória diferentes

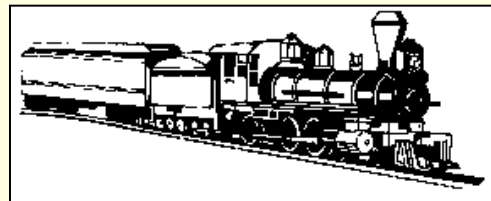
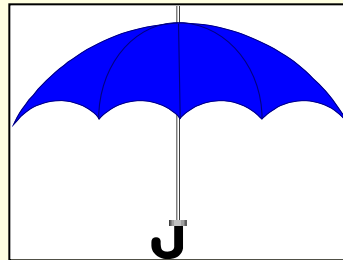


# Lista não homogênea

- Como inserir uma vaca, um guarda-chuva e um trem em uma mesma pilha?



←  
Topo



# Lista não homogênea

---

## ■ Solução 1

- Definem-se vários campos de informação
- Usam-se somente os necessários

```
struct no {  
    char info1;  
    int info2;  
    struct no *prox;  
}
```

- Desvantagem: memória alocada desnecessariamente

# Lista não homogênea

---

- Solução 2

- Definem-se vários ponteiros
- Aloca-se memória conforme necessidade

```
struct no {  
    char *info1;  
    int *info2;  
    struct no *prox;  
}
```

- Desvantagem: memória (dos ponteiros não usados) alocada desnecessariamente

# Lista não homogênea

## ■ Solução 3

- Usa-se um registro/estrutura variante

```
struct no {  
    union {  
        int ival;  
        float fval;  
        char cval;  
    } elemento;  
    int tipo;  
    struct no *prox;  
}
```

```
struct no *p;  
...  
p->tipo = 1; /*inteiro*/  
p->elemento.ival = 256;  
...  
p->tipo = 3; /*char*/  
p->elemento.cval = 'n';  
...
```

# Lista generalizada

---

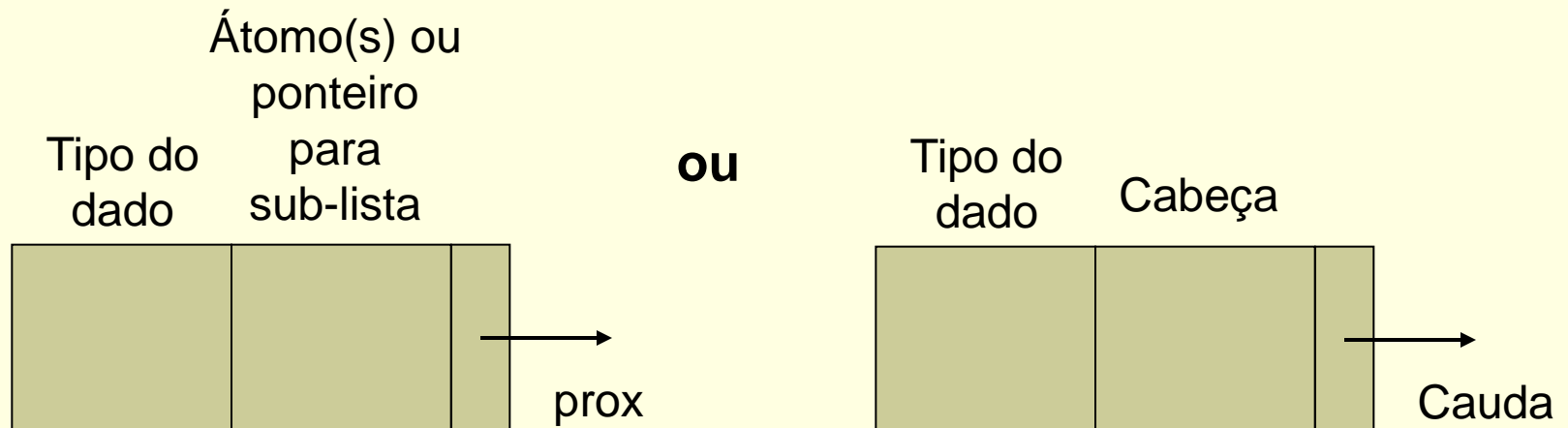
- Uma **lista generalizada** é aquela que pode ter como elemento ou um átomo ou uma outra lista (sub-lista)
  - Átomo: integer, real, char, string, etc.
- **Cabeça e cauda**
  - Cabeça: primeiro elemento da lista (átomo ou lista)
  - Cauda: o resto (uma outra lista, mesmo que vazia)

# Lista generalizada

## ■ Definição formal

- Uma lista generalizada  $A$  é uma seqüência finita de  $n \geq 0$  elementos  $\alpha_1, \alpha_1, \dots, \alpha_n$ , em que  $\alpha_i$  são átomos ou listas. Os elementos  $\alpha_i$ , com  $0 \leq i \leq n$ , que não são átomos são chamados sub-listas de  $A$ .

## ■ Estrutura básica do bloco de memória



# Lista generalizada

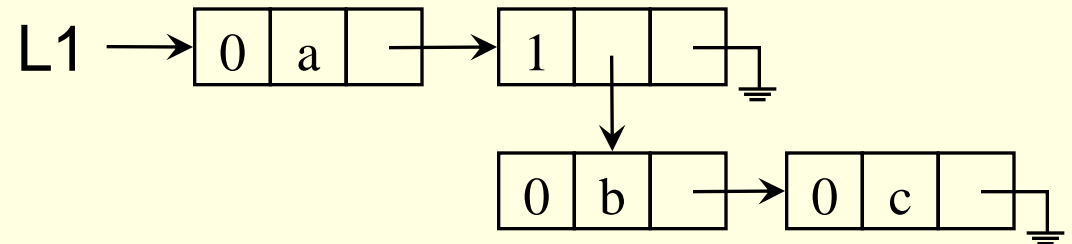
---

- Suponha que uma lista seja representada por elementos entre parênteses (no estilo da linguagem de programação LISP) ou entre colchetes (no estilo de PROLOG)
  - (a,b,c) ou [a,b,c]
  - (a,(b,c)) ou [a,[b,c]]
  - (a,(b),(c)) ou [a,[b],[c]]
  - (a,b,()) ou [a,b,[]]
  
- Tipo=0 indica átomo e tipo=1 indica sub-lista

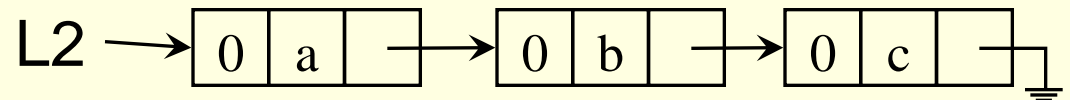
# Lista generalizada

## ■ Exemplos de representação

**L1 = (a,(b,c))**



**L2 = (a,b,c)**

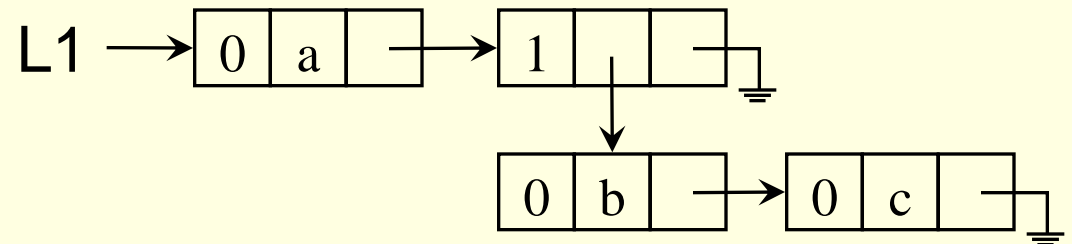




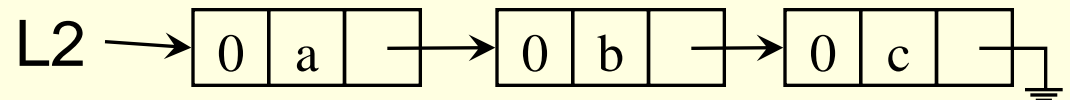
# Lista generalizada

## ■ Exemplos de representação

**L1 = (a,(b,c))**



**L2 = (a,b,c)**



Cabeça(L2)? Cauda(L2)? Cabeça(Cauda(L2))?

Cabeça(L1)? Cauda(L1)? Cabeça(Cauda(L1))?

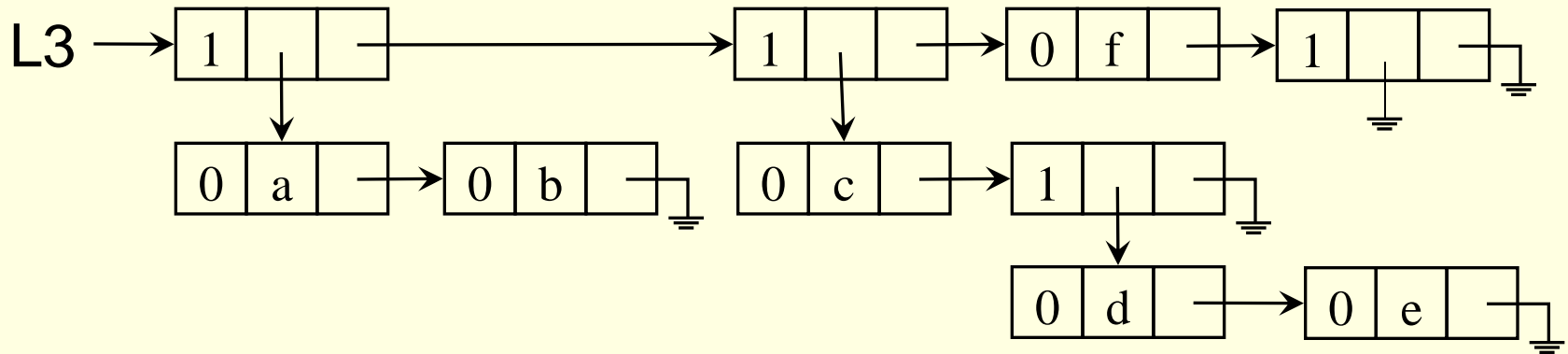
# Lista generalizada

---

- Exercício: faça a representação da lista L3  
**((a,b),(c,(d,e)),f,())**

# Lista generalizada

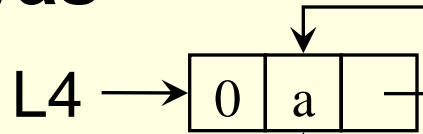
- Exercício: faça a representação da lista L3  
**((a,b),(c,(d,e)),f,())**



# Lista generalizada

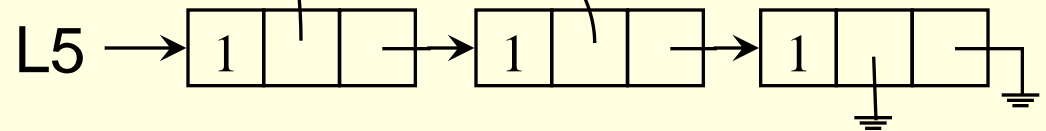
## Listas Recursivas

$L4 = (a, L4)$



## Listas Compartilhadas

$L5 = (L4, L4, ())$



# Lista generalizada

---

- Declaração em C
  - Union

# Lista generalizada

---

- Declaração em C
  - Union

```
typedef struct bloco {  
    union {  
        char atomo;  
        struct bloco *sublista;  
    } info;  
    int tipo;  
    struct bloco *prox;  
} no;
```

# Lista generalizada

---

## ■ Exercícios

- Implementar uma função recursiva para buscar um átomo x numa lista generalizada
  - (1) considere apenas a lista principal;
  - (2) considere que x pode estar em qualquer sublista.
- Implementar uma sub-rotina para verificar se duas listas generalizadas são iguais
  - Tente fazer a sub-rotina recursiva

# Algoritmos

- Uma função booleana recursiva para buscar um átomo  $x$  numa lista generalizada,  $L$ . Retorna também o endereço, se estiver lá.
  - (1) considere apenas a lista principal;

**Função** Busca ( $x, L$ ):

**Se**  $L$  é vazia **então retorna** FALSE

**Senão**  $L = (l_1, l_2, \dots, l_n)$  e

**se**  $l_1$  é átomo **então**

**se**  $l_1 = x$  **então retorna** TRUE e  $x$

**retorna** Busca ( $x, (l_2, l_3, \dots, l_n)$ )



# Função não-recursiva

---

```
int esta_na_lista_principal(no *p, elem x) {
    int achou=0;
    while ((p!=NULL) && (!achou)) {
        if ((p->tipo==0) && (p->info.atomo==x))
            achou=1;
        else p=p->prox;
    }
    return(achou);
}
```

# Algoritmos

- Uma função booleana recursiva para buscar um átomo  $x$  numa lista generalizada,  $L$ . Retorna também o endereço, se estiver lá.
  - (2) considere que  $x$  pode estar em qualquer sublista.

**Função** Busca ( $x, L$ ):

**Se**  $L$  é vazia **então retorna** FALSE

**Senão**  $L = (l_1, l_2, \dots, l_n)$  e

**se**  $l_1$  é átomo **então**

**se**  $l_1 = x$  **então retorna** TRUE e  $x$

**senão retorna** Busca ( $x, (l_2, l_3, \dots, l_n)$ )

**senão se** Busca ( $x, l_1$ ) **retorna** TRUE e  $x$

**senão retorna** Busca ( $x, (l_2, l_3, \dots, l_n)$ )

```
int esta_em_qualquer_parte_da_lista(no *p, elem x) {
    if (p==NULL)
        return 0;
    else if (p->tipo==0) {
        if (p->info.atomo==x)
            return 1;
        else return(esta_em_qualquer_parte_da_lista(p->prox,x));
    }
    else if (p->tipo==1) {
        if (esta_em_qualquer_parte_da_lista(p->info.sublista,x))
            return 1;
        else return(esta_em_qualquer_parte_da_lista(p->prox,x));
    }
}
```

# Algoritmos

- Verificar se duas listas generalizadas, L1 e L2, são iguais
  - Tente fazer função booleana recursiva

**Função Igual (K, L):**

**Se** K e L são vazias **então retorna** TRUE;

**Se** K ou L é vazia **então retorna** FALSE;

/\*ambas são não vazias: (k1,...kn) (l1,...lm)\*/

**Se** k1 e l1 são átomos e são iguais

**Então retorna** Igual((k2,...kn), (l2,...lm))

**Senão** se k1 e l1 são sublistas

**então se** Igual(k1, l1)

**então retorna** Igual((k2,...kn), (l2,...lm))

**senão retorna** FALSE

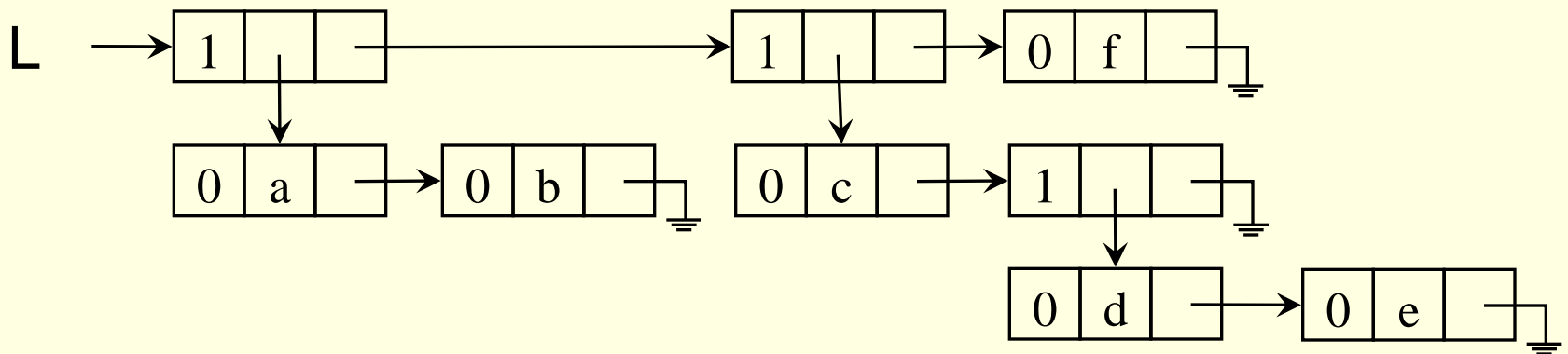
```
int iguais(no *L1, no *L2) {
    if ((L1==NULL) && (L2==NULL))
        return 1;
    else if ((L1==NULL) || (L2==NULL))
        return 0;
    else if ((L1->tipo==L2->tipo==0) && (L1->info.atomo==L2->info.atomo))
        return(iguais(L1->prox,L2->prox));
    else if ((L1->tipo==L2->tipo==1) && (iguais(L1->info.sublista,L2->info.sublista)))
        return(iguais(L1->prox,L2->prox));
    else return 0;
}
```

# Listas e recursão

## ■ Exercício extra

- Implementar uma sub-rotina que determina a profundidade máxima de uma lista generalizada
  - Tente usar recursividade

- Por exemplo, para o caso abaixo, a sub-rotina deveria retornar profundidade 3



# Profundidade máxima de uma lista generalizada S

```
Função Profundidade ( S ) :  
Se S é atomo ou S = lista vazia então retorna 0  
senão{prof_atual = 0;  
    para cada elemento elem de S:  
        {prof := Profundidade(elem);  
        se prof > prof_atual  
            então prof_atual := prof };  
    retorna prof_atual + 1;  
}
```

Ex.  $S = (a, (b)) \Rightarrow \text{Prof}(S) = 2$

$A = (a, b, c) \Rightarrow \text{Prof}(A) = 1$

$B = () \Rightarrow \text{Prof}(B) = 0;$

```
int profundidade(no *p) {
    int prof, aux;
    if (p==NULL)
        prof=0;
    else if (p->tipo==0) {
        prof=1;
        aux=profundidade(p->prox);
        if (aux>prof)
            prof=aux; }
    else if (p->tipo==1) {
        prof=1+profundidade(p->info.sublista);
        aux=profundidade(p->prox);
        if (aux>prof)
            prof=aux; }
    return(prof);
}
```



# Lista generalizada

---

- Exercício

- Implementar uma sub-rotina para verificar se duas listas generalizadas são estruturalmente iguais
  - O conteúdo em si não importa

```
int iguais_estruturalmente(no *L1, no *L2) {
    if ((L1==NULL) && (L2==NULL))
        return 1;
    else if ((L1==NULL) || (L2==NULL))
        return 0;
    else if (L1->tipo==L2->tipo==0)
        return(iguais_estruturalmente(L1->prox,L2->prox));
    else if ((L1->tipo==L2->tipo==1) && (iguais_estruturalmente(L1->
info.sublista,L2->info.sublista)))
        return(iguais_estruturalmente(L1->prox,L2->prox));
    else return 0;
}
```

# Lista generalizada e polinômios

---

- Considere os polinômios:

$$P1 = 4x^2y^3z + 3xy + 5$$

$$P2 = x^{10}y^3z^2 + 2x^8y^2z^2 + x^4y^4z + 6x^3y^4z + 2yz$$

$$P3 = 3x^2y$$

(a) n° de termos: variável

- P1=3, P2=5, P3=1

(b) n° de variáveis: variável

- P1=P2=3, P3=2

(c) nem todo termo é expresso com todas as variáveis

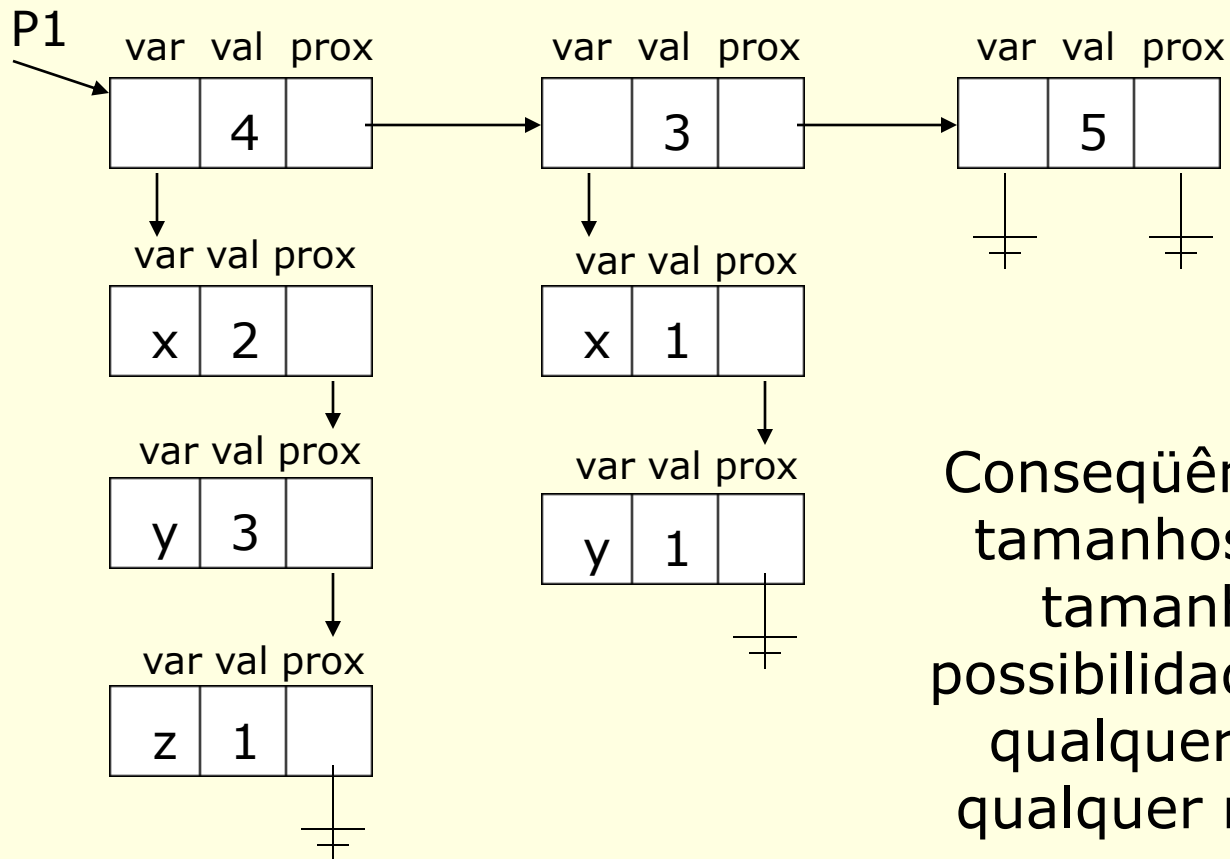
# Lista generalizada e polinômios

---

- Objetivos
  - representar de forma a otimizar o uso de memória
  - representação única para todo polinômio
- Solução: lista generalizada

# Lista generalizada e polinômios

Ex:  $P1 = 4x^2y^3z + 3xy + 5$



Conseqüência: registros de tamanhos fixos; listas de tamanhos variáveis; possibilidade de representar qualquer polinômio com qualquer n° de variáveis e qualquer grau

# Exercício

---

- Implementar uma sub-rotina que:
  - (a) receba um polinômio representado via lista generalizada e os valores das variáveis
  - (b) percorra a lista generalizada e compute o resultado do polinômio
  - (c) retorne o resultado para quem chamou a sub-rotina