

Listas: conceito, representação e algoritmos

SCC0202 Algoritmos e Estruturas de
Dados I

Lista

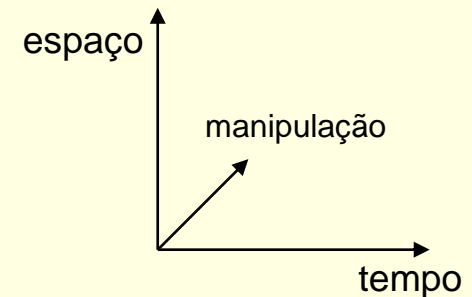
- Pode ser utilizada para representar **qualquer conceito ou necessidade**
 - Não apenas pilhas e filas
- **Quaisquer operações e estruturas**

Listas

- Estão entre as estruturas mais flexíveis e versáteis
 - Quaisquer **estruturas** de informação
 - Quaisquer **relações** entre informações
 - Quaisquer formas de **organização e manipulação**
 - Criação, inserção, remoção, atualização, etc.
 - Dependente somente da aplicação e de suas necessidades

Listas e aplicações

- Diversas **necessidades** podem ser requeridas por aplicações
 - Determinam como serão a lista e as operações
- **Requisitos**
 - Capacidade de armazenamento
 - Velocidade de acesso
 - Facilidade de manipulação



Capacidade de armazenamento

- Informação a ser armazenada
 - Exemplo: **inteiros vs. strings**
 - Melhor solução: uma tabela de inteiros e strings, usando-se os inteiros como informação
 - Por exemplo, em uma universidade

Inteiro	String
1	Professor
2	Estudante
3	Técnico
...	...

Capacidade de armazenamento

- Informação a ser armazenada
 - Exemplo: **inteiros vs. strings**
 - Melhor solução: uma tabela de inteiros e strings, usando-se os inteiros como informação
 - Por exemplo, em uma universidade

Inteiro	String
1	Professor
2	Estudante
3	Técnico
...	...

4 bytes

10 bytes

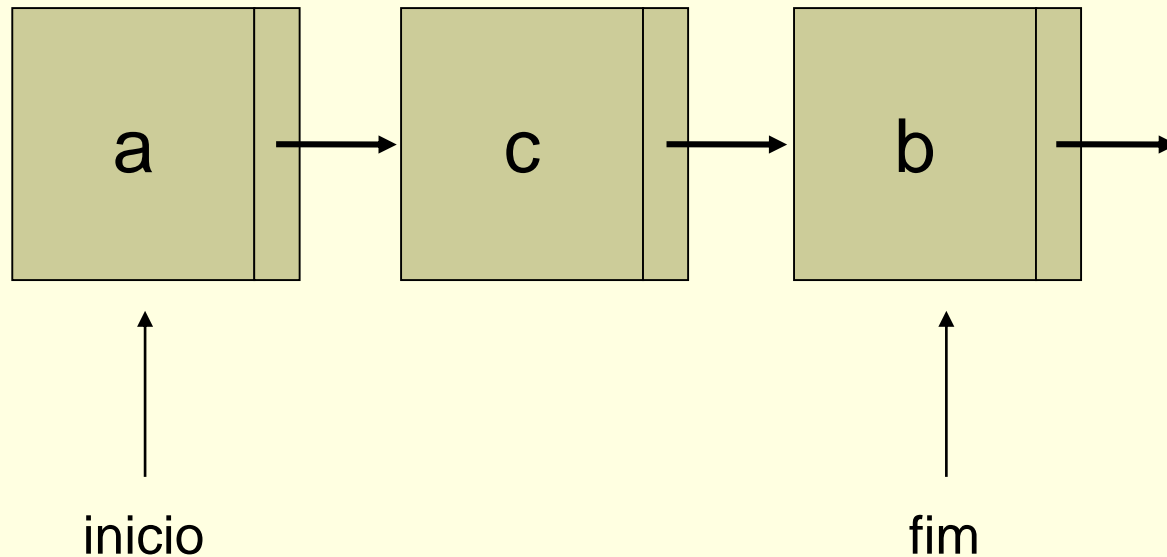
Velocidade de acesso à informação

- **Perguntas** a serem respondidas
 - Que informação se quer acessar?
 - Com que frequência se quer acessar a informação?

- Pode determinar
 - Estrutura de dados: pilha, fila, árvore ou outra qualquer
 - Organização da informação na lista
 - Insere-se elementos no início, no fim ou no meio da lista? Informação ordenada ou não?
 - Número de ponteiros a ser utilizado

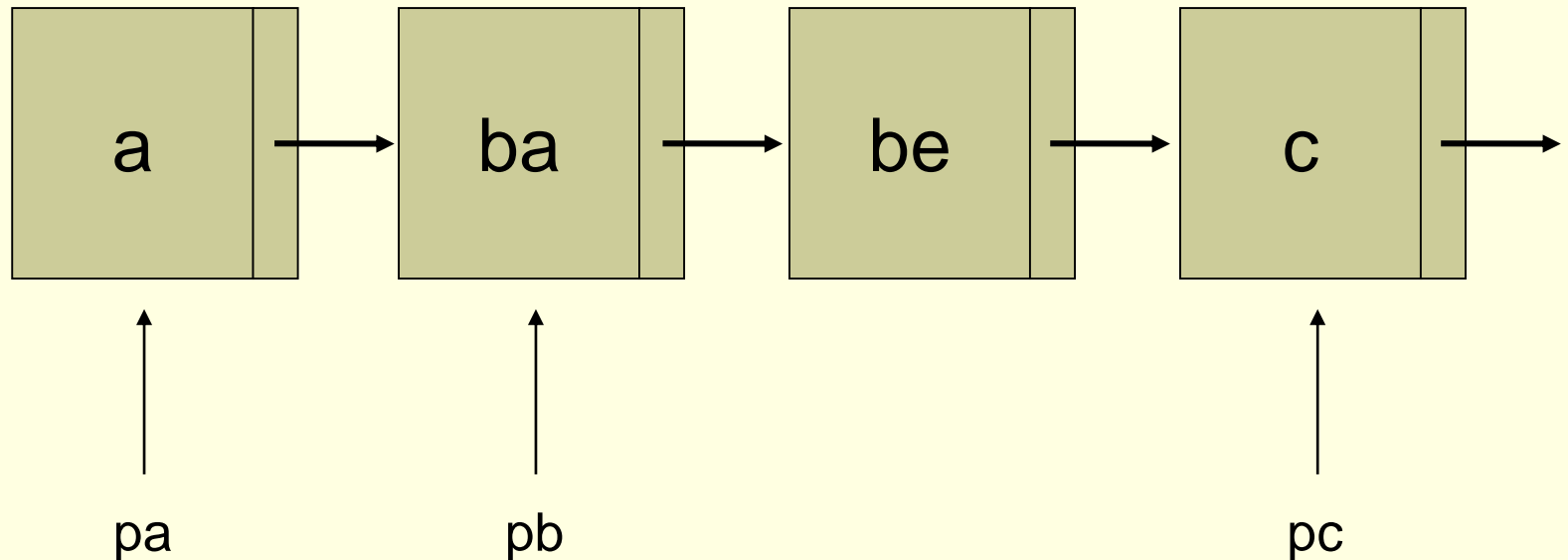
Variações da lista

- Lista simples, desordenada



Variações da lista

- Lista simples, ordenada e com ponteiros extras



Variações da lista

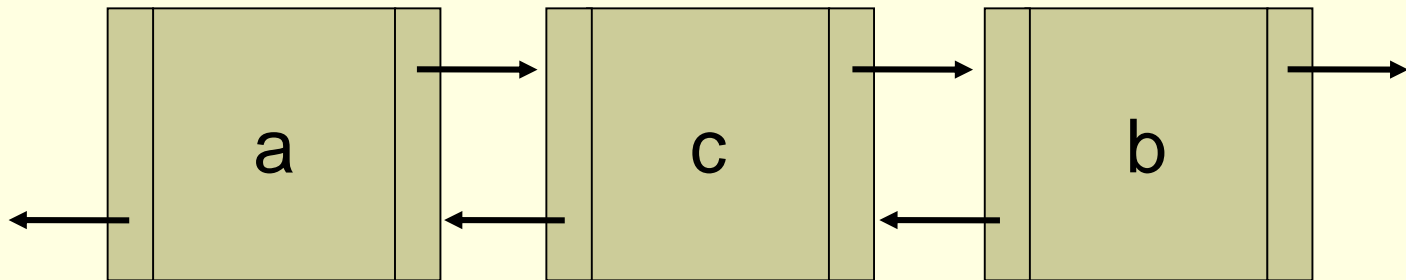
- Lista simples, ordenada e com ponteiros extras

```
typedef struct bloco {  
    char nome[20];  
    struct bloco *prox;  
} pessoa
```

```
pessoa *pa, *pb, *pc, *pd, ... , *pz;
```

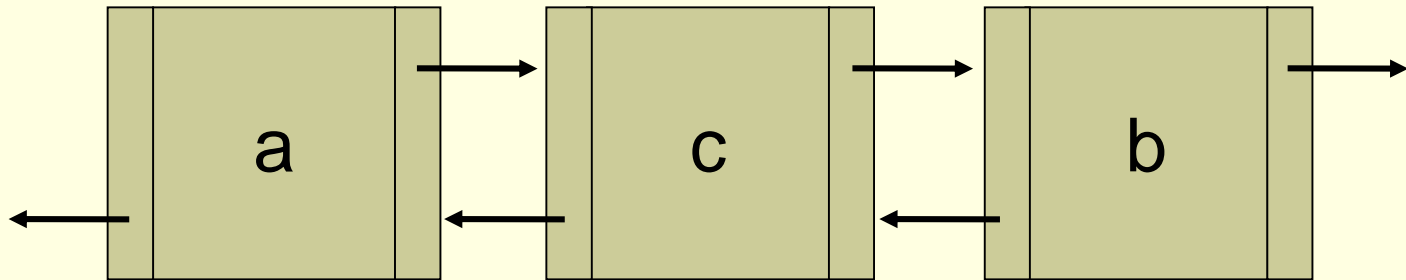
Variações da lista

- Lista duplamente encadeada
 - Utilidade?



Variações da lista

- Lista duplamente encadeada
 - Facilita navegação



Variações da lista

- Lista duplamente encadeada
 - Facilita navegação

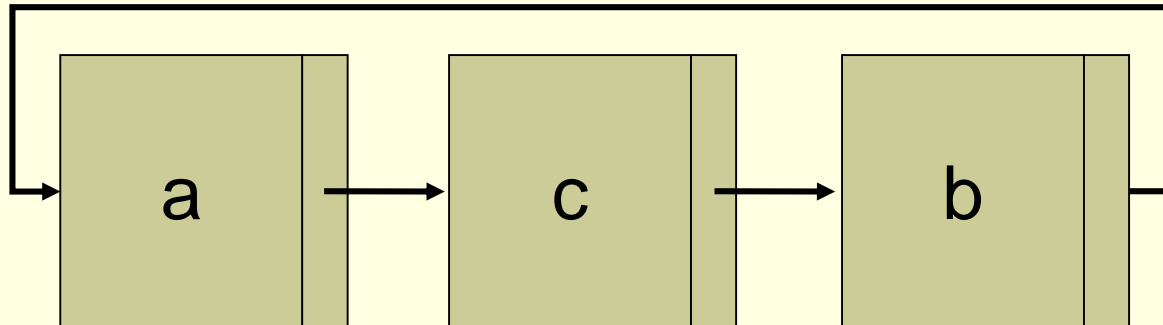
```
typedef struct bloco {  
    char nome[20];  
    struct bloco *ant, *prox;  
} no;
```

Variações da lista

- Lista duplamente encadeada
 - Facilita navegação
- Exercício em duplas: implemente uma função para inserir um nome nessa lista, sendo que o usuário decide em que posição quer inserir
 - Se 1, primeiro elemento (se necessário, “empurra demais elementos”)
 - Se 2, segundo elemento (se necessário, “empurra demais elementos”)
 - ...
 - Se $n > \text{número de elementos da lista}$, inserir no fim

Variações da lista

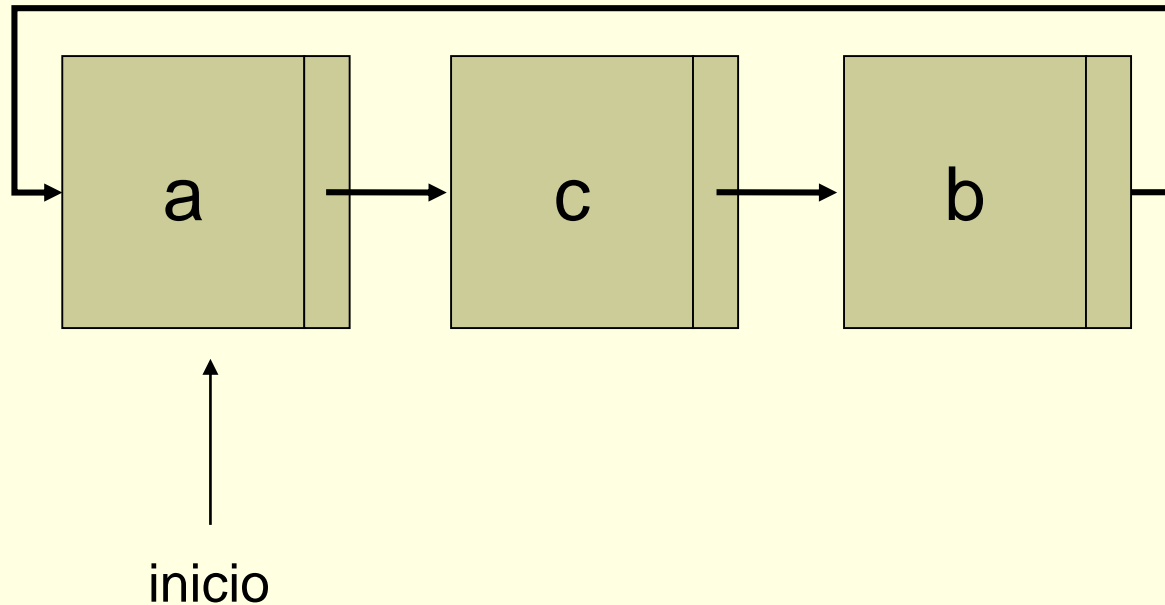
■ Lista circular



Como saber onde é o início (se é que há algum)?

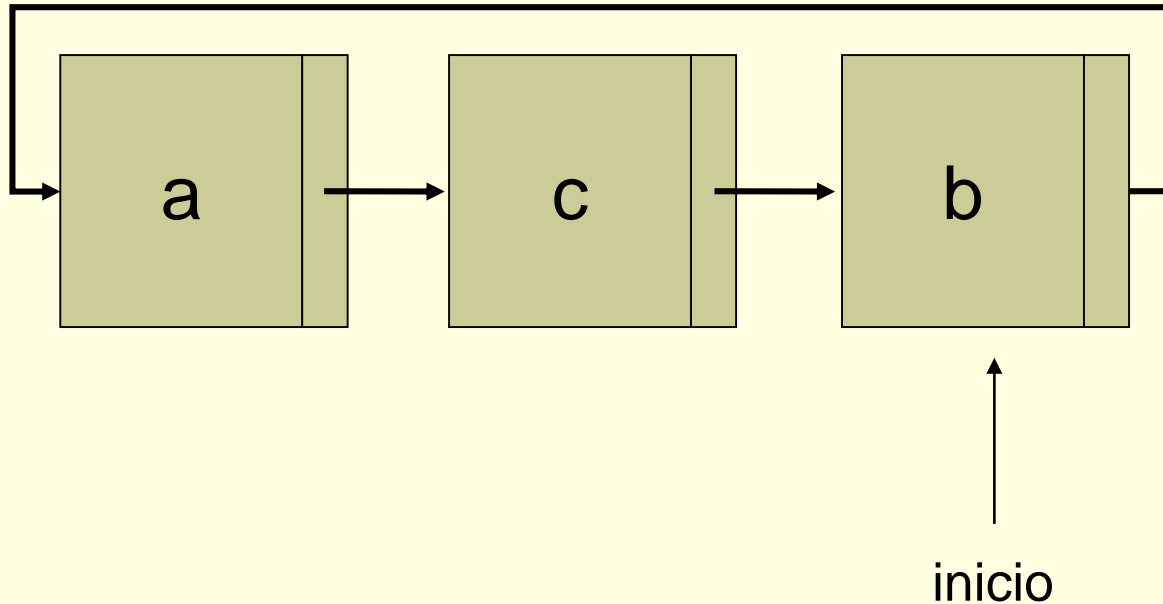
Variações da lista

■ Lista circular



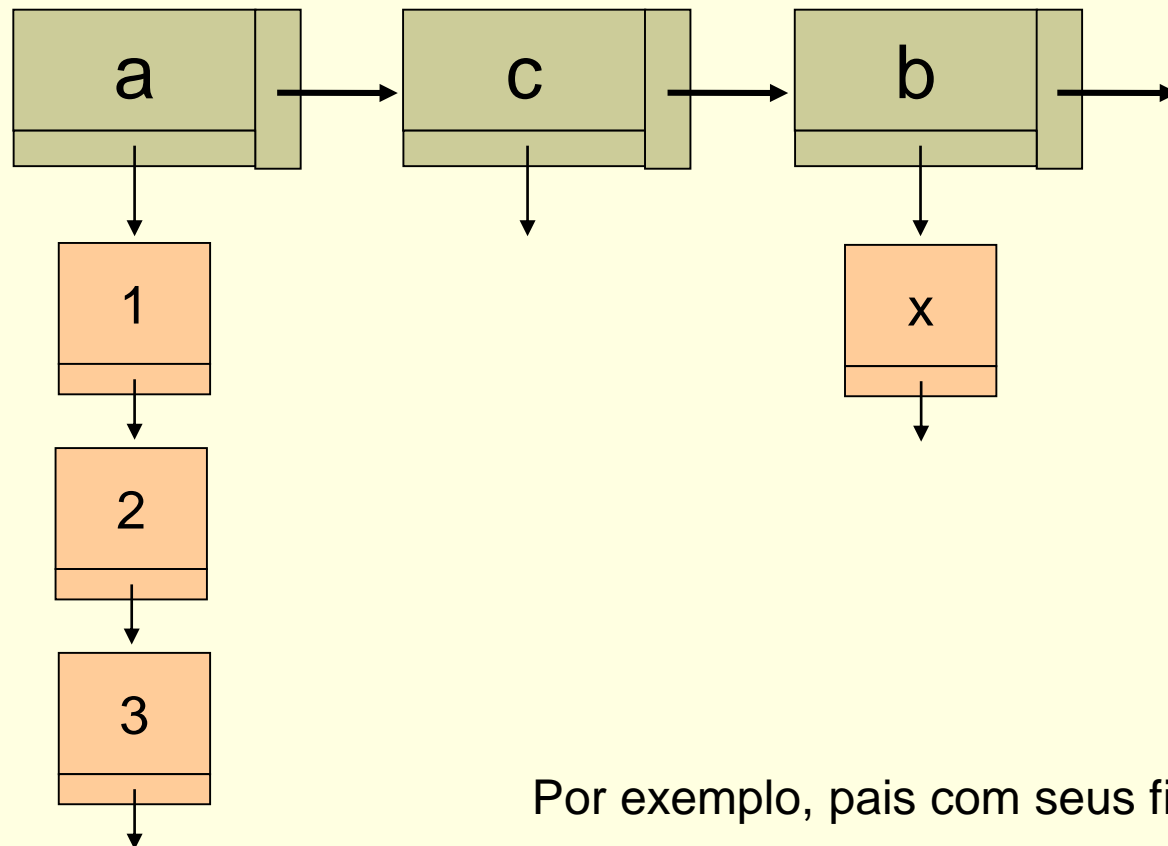
Variações da lista

■ Lista circular



Variações da lista

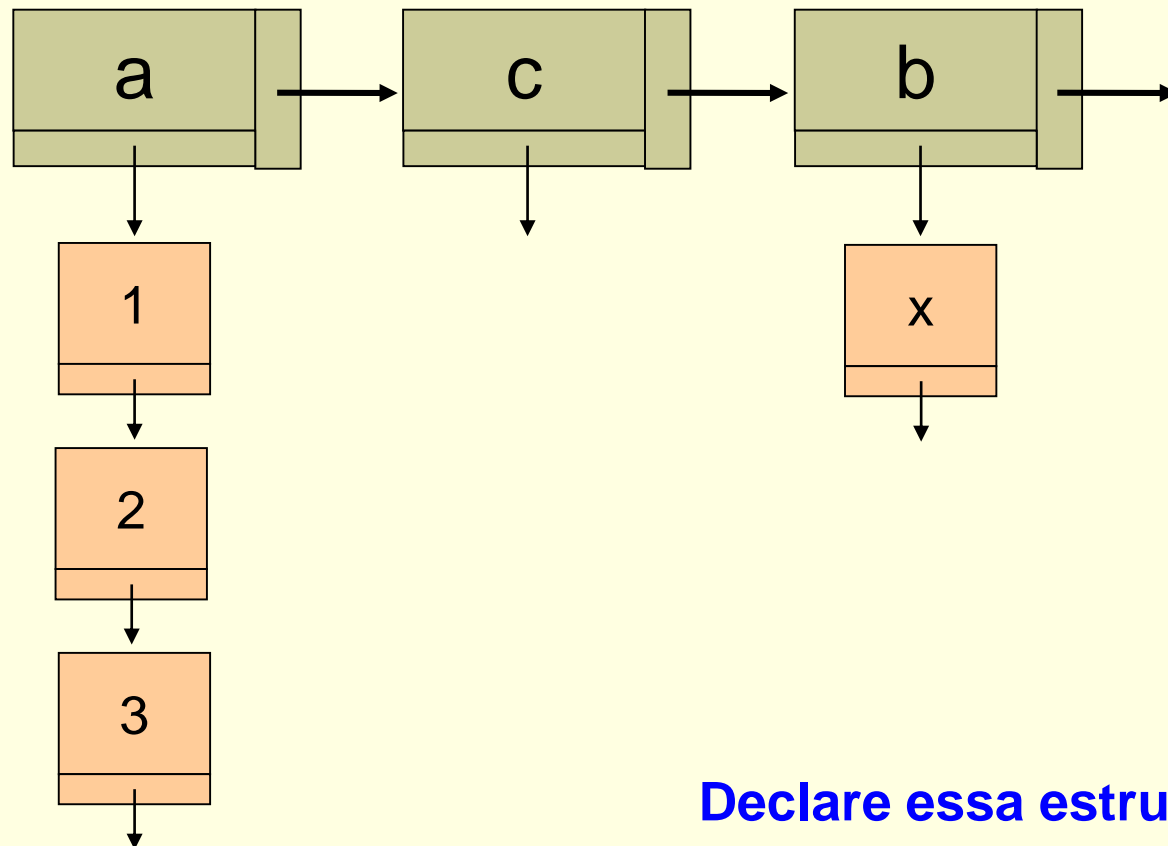
- Exemplo de lista não linear
 - Informações de diferentes níveis



Por exemplo, pais com seus filhos

Variações da lista

- Exemplo de lista não linear
 - Informações de diferentes níveis



Declare essa estrutura!

Variações da lista

■ Possível declaração

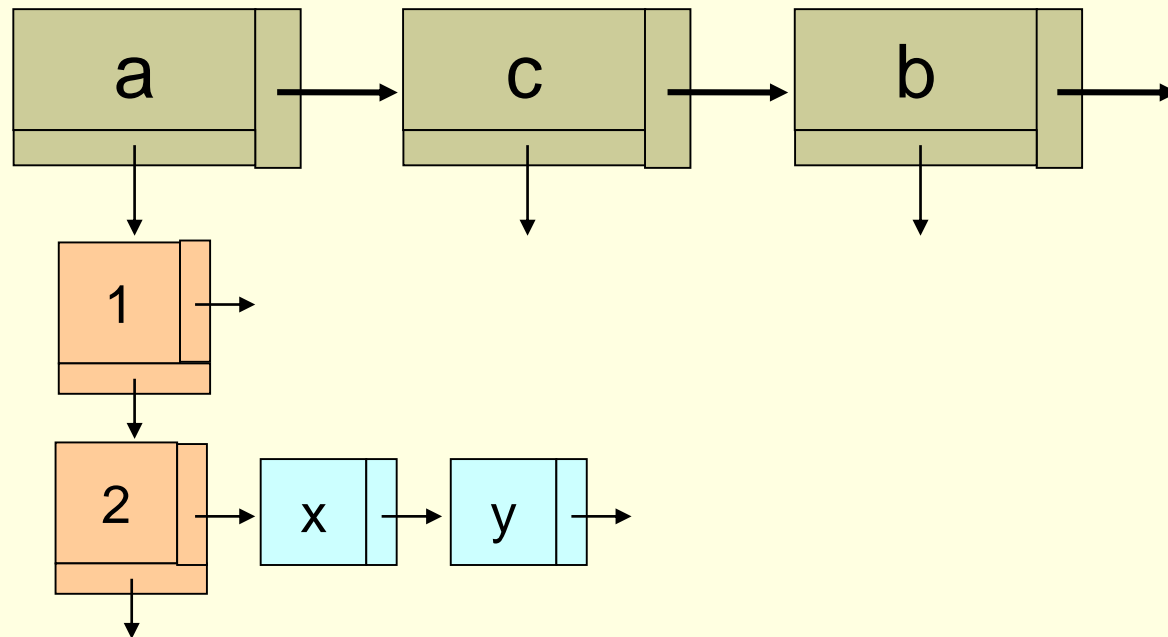
```
typedef struct filho {  
    char nome[20], data_nascimento[10];  
    struct filho *prox;  
} Filho;
```

```
typedef struct pai {  
    char nome[20], nome_conjuge[20];  
    int nro_filhos;  
    struct pai *prox;  
    Filho *filhos;  
} Pai;
```

```
typedef struct {  
    Pai *ini, *fim;  
} Listagem;
```

Variações da lista

- Exemplo de lista não linear
 - Informações de diferentes níveis
 - Tão complexo e elaborado quanto se queira

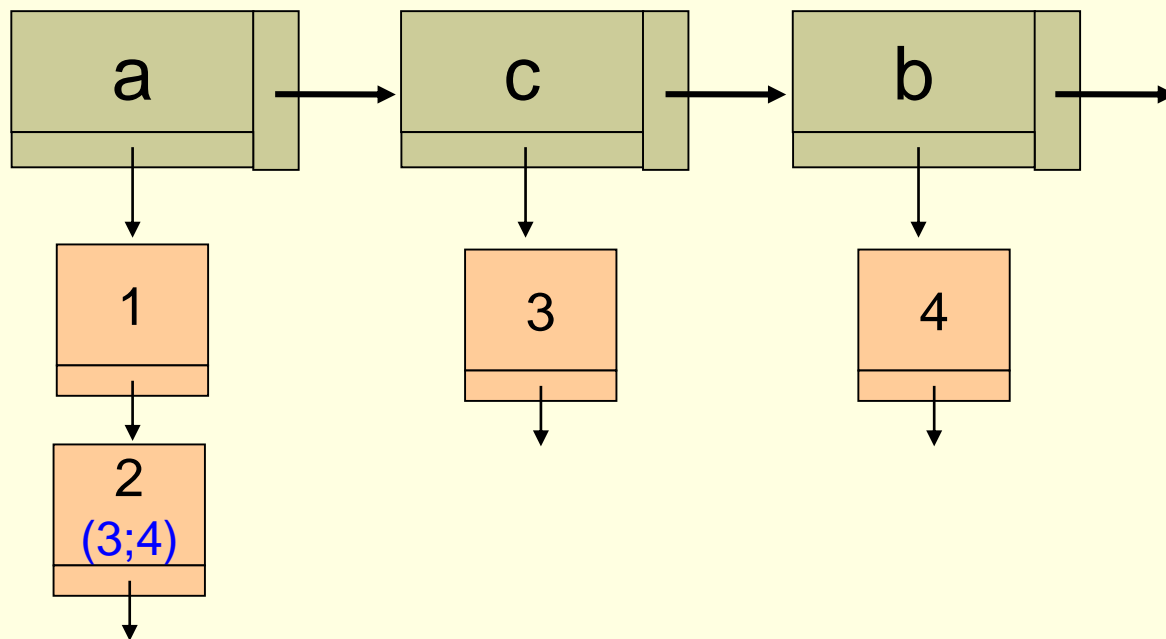


Exercício em duplas

- Faça os diagramas necessários e declare uma lista de pais que têm filhos estudando no ICMC, sendo que
 - Cada pai pode ter uma lista de filhos estudando no ICMC
 - Cada estudante do ICMC pode ter uma lista de amigos que também são estudantes do ICMC
 - Cada amigo de um estudante deve ser relacionado a seu próprio pai

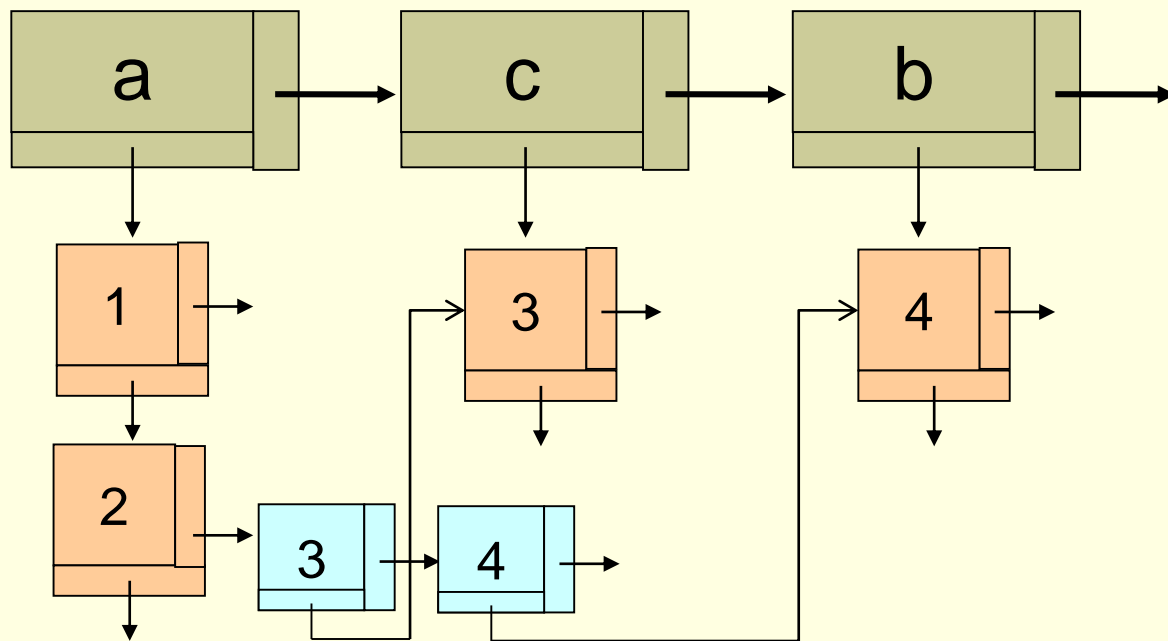
Possível solução

- Cada estudante guarda uma string com a listagem de amigos concatenados
 - Problemas dessa solução?



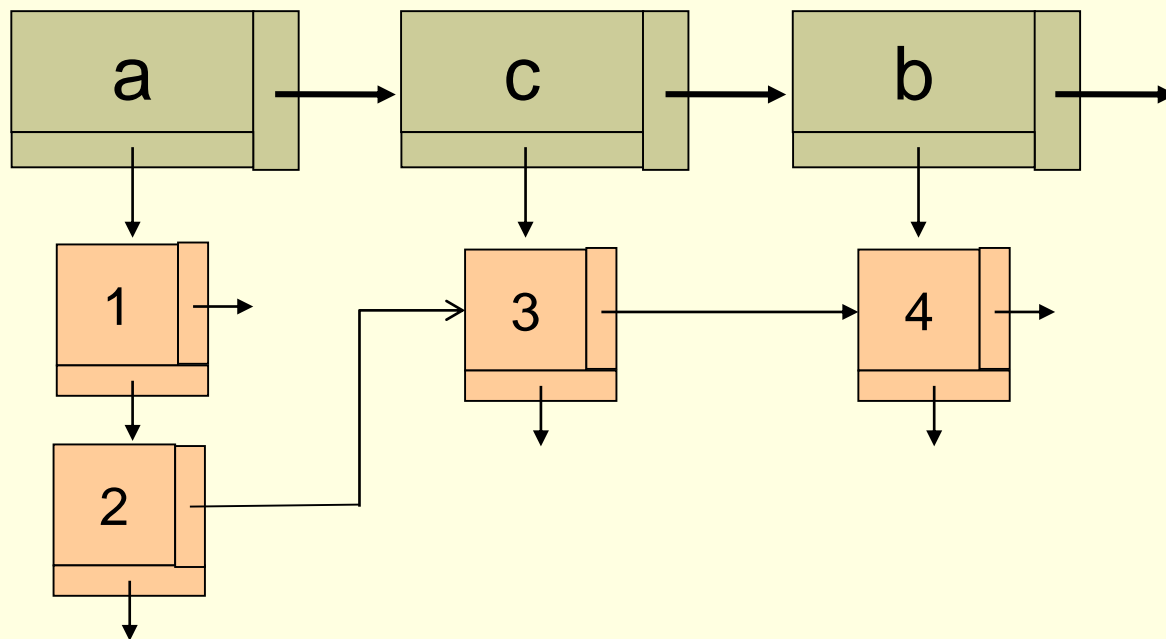
Possível solução

- Cada estudante guarda uma lista de amigos, sendo que cada amigo aponta para seu nó na estrutura
 - Problemas dessa solução?



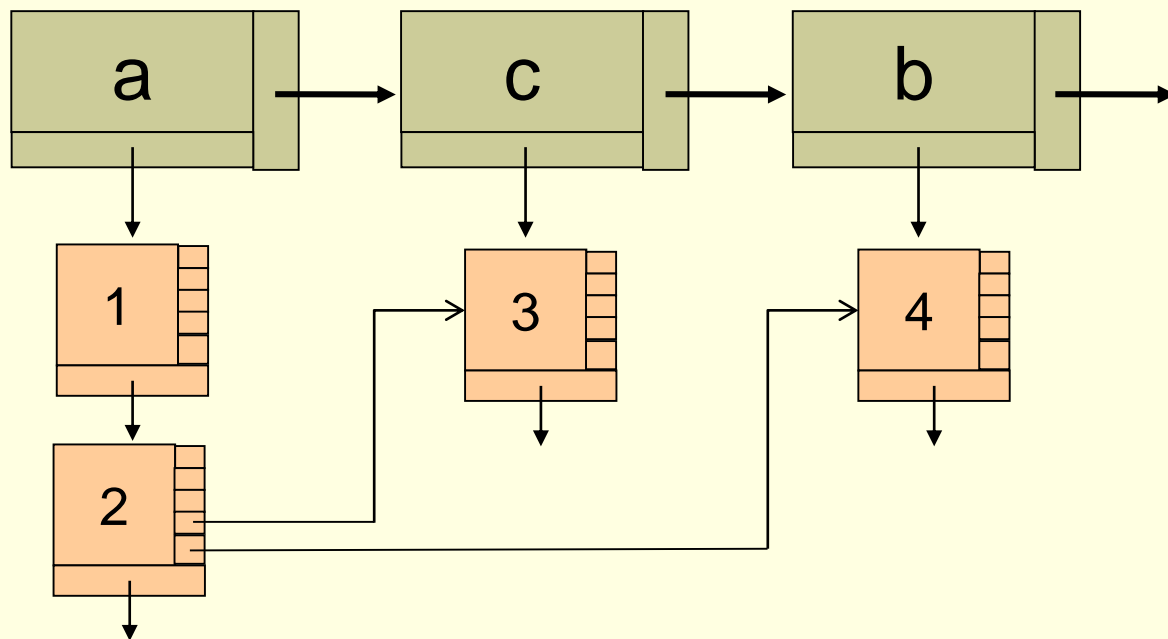
Possível solução

- Cada estudante tem um ponteiro para seus amigos
 - Problemas dessa solução?



Possível solução

- Cada estudante tem uma lista de ponteiros para seus amigos
 - Problemas dessa solução?



Possível solução

```
#define NRO_MAX_AMIGOS 5
```

```
typedef struct filho {  
    char nome[20], data_nascimento[10];  
    struct filho *amigos[NRO_MAX_AMIGOS];  
    struct filho *prox;  
} Filho;
```

```
typedef struct pai {  
    char nome[20], nome_conjuge[20];  
    int nro_filhos;  
    struct pai *prox;  
    Filho *filhos;  
} Pai;
```

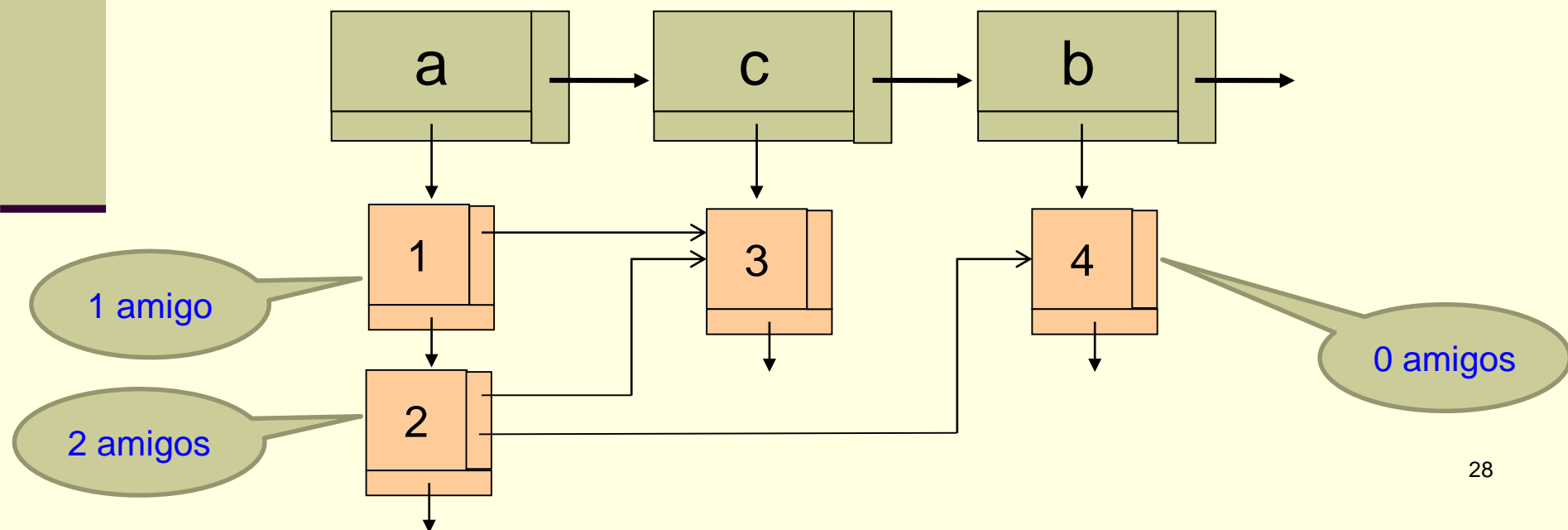
```
typedef struct {  
    Pai *ini, *fim;  
} Listagem;
```



5 amigos
pré-alocados

Possível solução

- Cada estudante tem uma lista de número variável de ponteiros (portanto, número de amigos pode variar de pessoa para pessoa)
 - Como declarar algo assim?



Possível solução

```
typedef struct filho {  
    char nome[20], data_nascimento[10];  
    struct filho **amigos;  
    int nro_amigos;  
    struct filho *prox;  
} Filho;  
  
typedef struct pai {  
    char nome[20], nome_conjuge[20];  
    int nro_filhos;  
    struct pai *prox;  
    Filho *filhos;  
} Pai;  
  
typedef struct {  
    Pai *ini, *fim;  
} Listagem;
```

Número variável de amigos por pessoa

Como fazer essa alocação?

Possível solução

```
typedef struct filho {
    char nome[20], data_nascimento[10];
    struct filho **amigos;
    int nro_amigos;
    struct filho *prox;
} Filho;

typedef struct pai {
    char nome[20], nome_conjuge[20];
    int nro_filhos;
    struct pai *prox;
    Filho *filhos;
} Pai;

typedef struct {
    Pai *ini, *fim;
} Listagem;
```

Número variável de amigos por pessoa

```
amigos=(Filho**) malloc(5*sizeof(Filho*));
nro_amigos=5;
amigos[0]=... //endereço do amigo
amigos[1]=... //endereço do outro amigo
...
```

Facilidade de manipulação da lista

- Operações a serem efetuadas sobre os dados
 - Organização dos dados
 - Ponteiros
- Perguntas
 - Quais as operações mais frequentes?
 - Qual o custo computacional de cada operação?

Exercício para casa

- Implementar uma rotina para manter um cadastro de conveniados de um plano médico e seus dependentes
 - Operações
 - Inserir ou retirar um conveniado
 - Inserir ou retirar dependentes de um conveniado
 - Restrições
 - Conveniados só podem ser cadastrados uma vez e não podem ser dependentes de outros conveniados