Lista: conceito, representação e algoritmos

SCC-202 – Algoritmos e Estruturas de Dados I

Lucas Antiqueira

Problema

- Imaginem a situação da automação de uma biblioteca
  - Todos os livros devem ser cadastrados
  - O sistema deve informar se um determinado livro está ou não disponível nas estantes
  - Caso o livro não esteja disponível, o usuário poderá aguardar pela liberação do livro se cadastrando em uma fila de espera
  - Quando o livro for devolvido e liberado, o primeiro da fila deve ser contatado para vir buscá-lo

Problema

- 120.000 livros
- 1 fila de espera para cada livro
- No máximo 1000 pessoas ficam esperando por livros da biblioteca
- No máximo 30 pessoas ficam esperando pela retirada de um mesmo livro

Problema

- Como representar/estruturar o problema?

Soluções

- Alternativa 1
  - Reservar espaço para 120.000 filas (uma para cada livro), com capacidade para 30 pessoas
  - 120.000 vetores 30 elementos
  - Espaço reservado para 3.600.000 pessoas

  - Problema?

Soluções

- Alternativa 1
  - Reservar espaço para 120.000 filas (uma para cada livro), com capacidade para 30 pessoas
  - 120.000 vetores 30 elementos
  - Espaço reservado para 3.600.000 pessoas

  - Problema?
    - Muito espaço reservado não é utilizado

Soluções

- Alternativa 2
  - Alocar espaço para 1000 elementos
  - Todas as 120.000 filas compartilham o mesmo espaço
  - Problema?

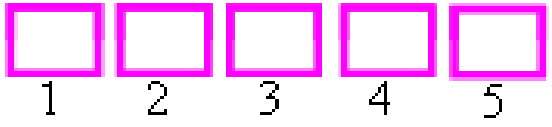
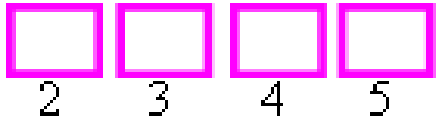
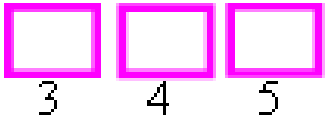
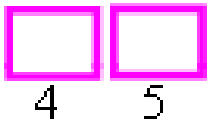
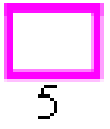
Soluções

- Alternativa 2
  - Alocar espaço para 1000 elementos
  - Todas as 120.000 filas compartilham o mesmo espaço
  - Problema?
    - Como 120.000 filas podem compartilhar a memória reservada a elas?

Como várias estruturas podem compartilhar um espaço de memória?

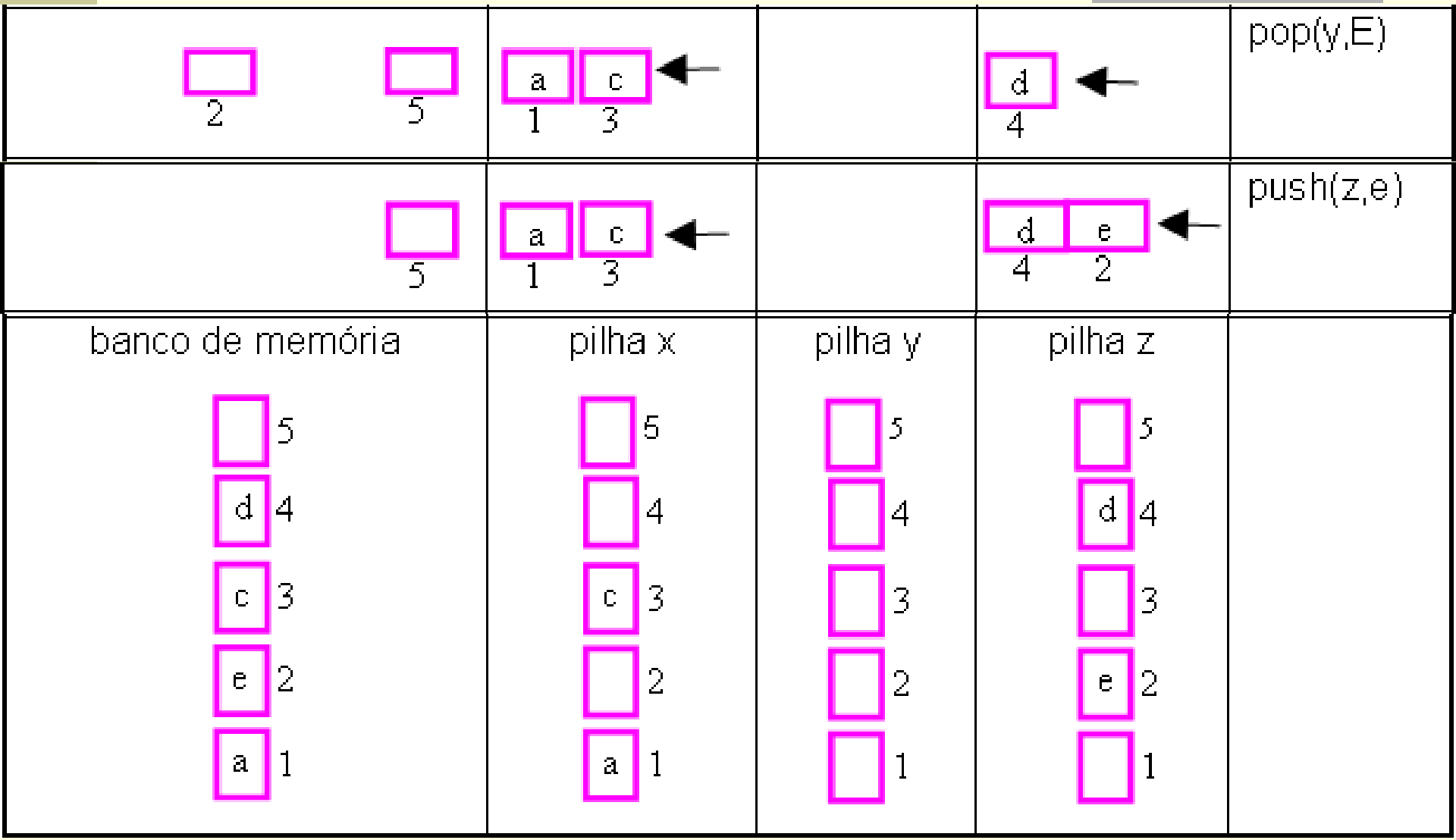
Compartilhamento de memória

(Exemplo com pilhas)

banco de memória	pilha x	pilha y	pilha z	operação
				
				push(x,a)
				push(y,b)
				push(x,c)
				push(z,d)

Compartilhamento de memória

(Exemplo com pilhas)



Compartilhamento de memória
(Exemplo com pilhas)

- Perguntas
  - Como saber qual é o topo de uma pilha dessas (x, y, z)?
  - Como saber qual elemento vem logo abaixo do topo (o próximo na seqüência)?

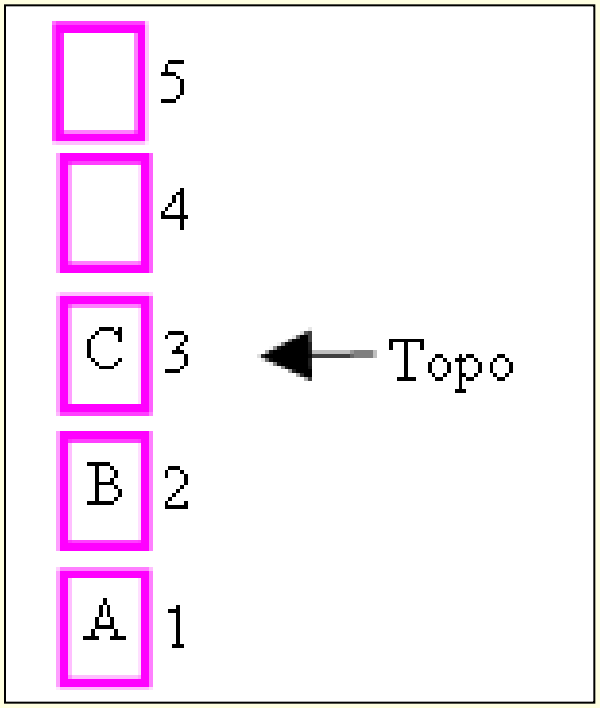
Compartilhamento de memória
(Exemplo com pilhas)

- Perguntas
  - Como saber qual é o topo de uma pilha dessas (x, y, z)?
  - Como saber qual elemento vem logo abaixo do topo (o próximo na seqüência)?

Alocação encadeada!

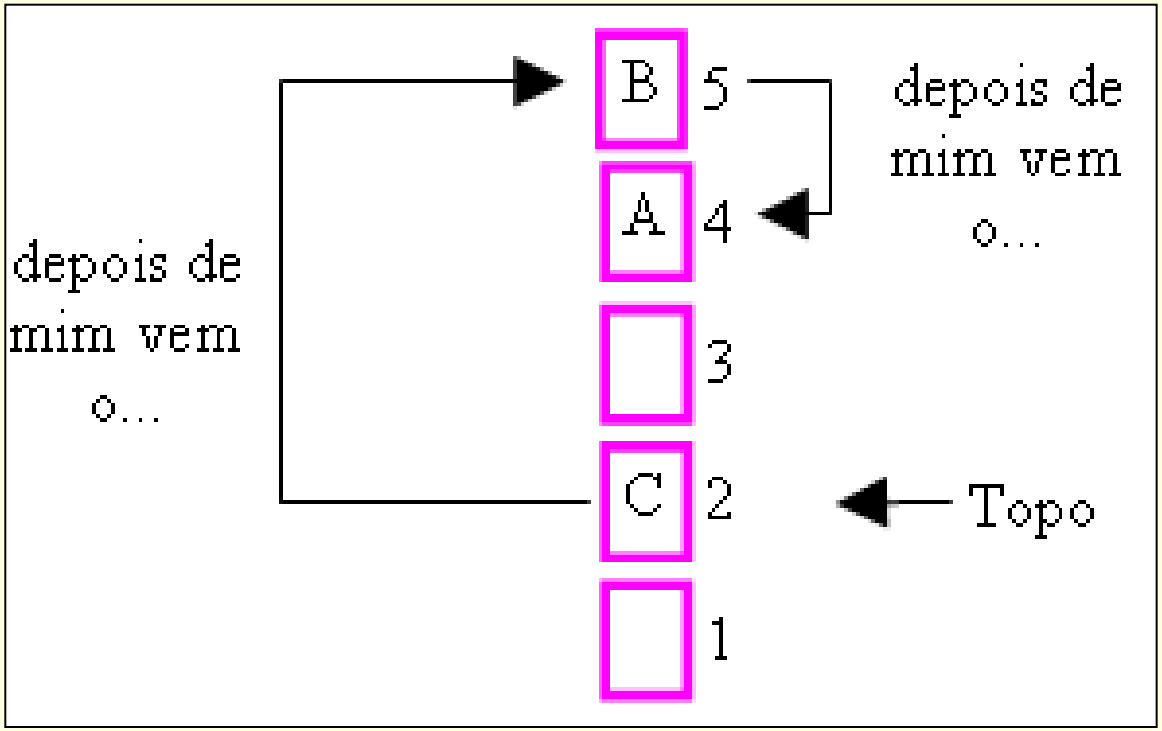
Alocação seqüencial vs. encadeada

- Alocação seqüencial: elementos são alocados em seqüência (seqüência “física”)



Alocação seqüencial vs. encadeada

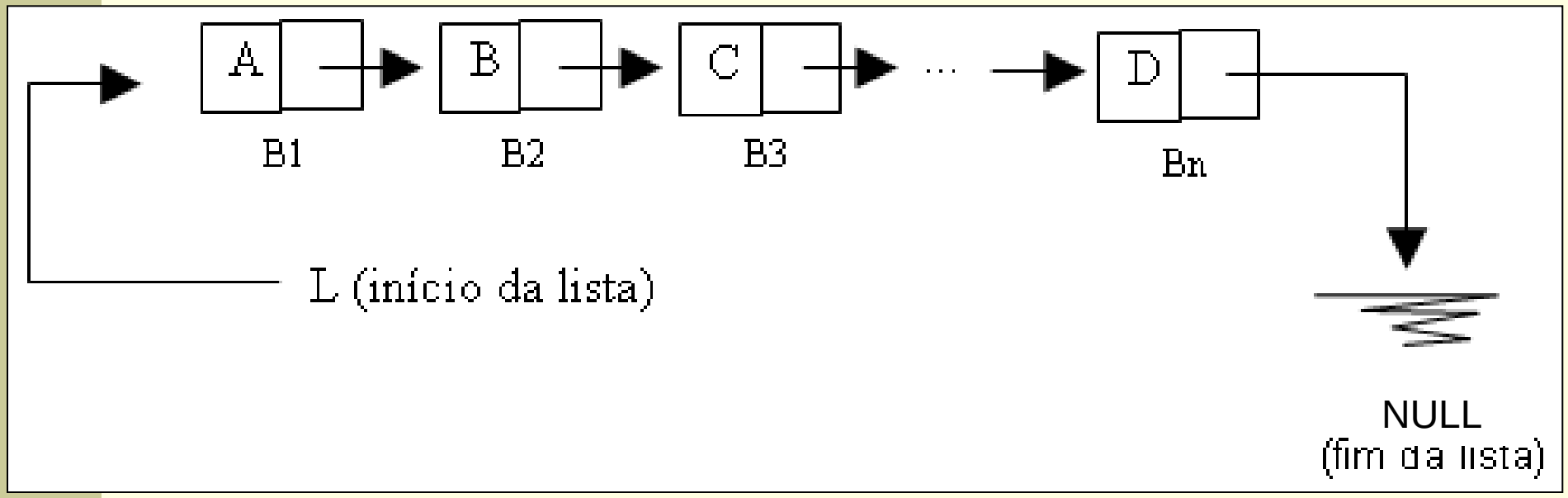
- Alocação encadeada: elementos não estão necessariamente em posições adjacentes de memória; seqüência “lógica” ou “virtual”

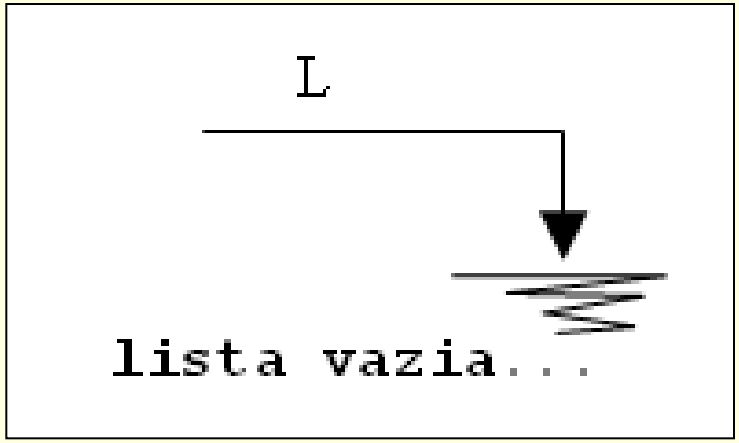


Listas encadeadas

- Definição: uma lista encadeada L, com n blocos de memória $B_1, B_2, ..., B_n$ é definida pelas seguintes características:
  - Cada bloco de memória B_1, ou cada "nó" da lista, tem pelo menos dois campos:
    - Informação a ser armazenada
    - Indicação do próximo elemento da lista
  - Os blocos de memória não estão necessariamente em seqüência física na memória
  - O acesso aos elementos da lista ocorre através de um indicador do início da lista (o primeiro elemento); o acesso aos demais elementos ocorre através da indicação de quem é o próximo na seqüência
  - O último nó da lista aponta para NULL

Representação





Lista

- Listas: lineares ou não-lineares
  - Exemplos?

- Definição de lista linear
  - *Estrutura de dados que armazena elementos de forma alinhada, ou seja, um após o outro*

- Implementação:
  - Estática ou dinâmica
  - Seqüencial ou encadeada

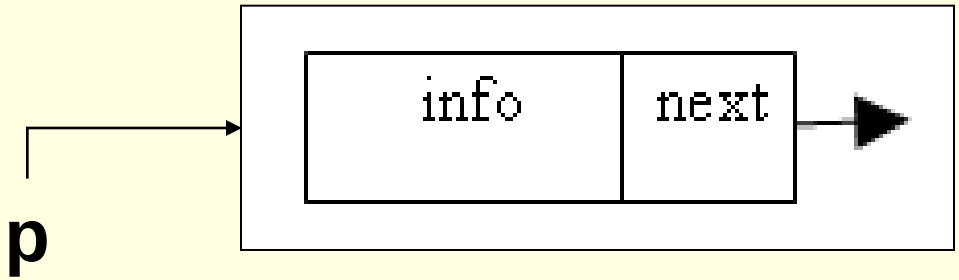
Lista

- Lista encadeada e dinâmica
  - Uma das representações mais interessantes e flexíveis que há
  - Aplicável a diversos problemas

Possível declaração

```
struct no {
   char info;
   struct no *next;
}

struct no *p;
p = (struct no*) malloc(sizeof(struct no));
```



```c
#include <stdio.h>

struct no {
       char info;
       struct no *next;
};

struct no *ini, *fim, *p;

int main(void) {
    ini=NULL;
    fim=NULL;

    p=(struct no*)
       malloc(sizeof(struct no));
    p->info='a';
    p->next=NULL;
    ini=p;
    fim=p;

    p=(struct no*)
       malloc(sizeof(struct no));
    p->info='b';
    p->next=NULL;
    fim->next=p;
    fim=p;

    p=ini;
    while (p!=NULL) {
          printf("%c ",p->info);
          p=p->next;
    }

    p=ini;
    while (p!=NULL) {
          ini=ini->next;
          free(p);
          p=ini;
    }

    return 0;
}
```

Exemplo

Qual o resultado da execução desse programa?

Operações genéricas sobre lista

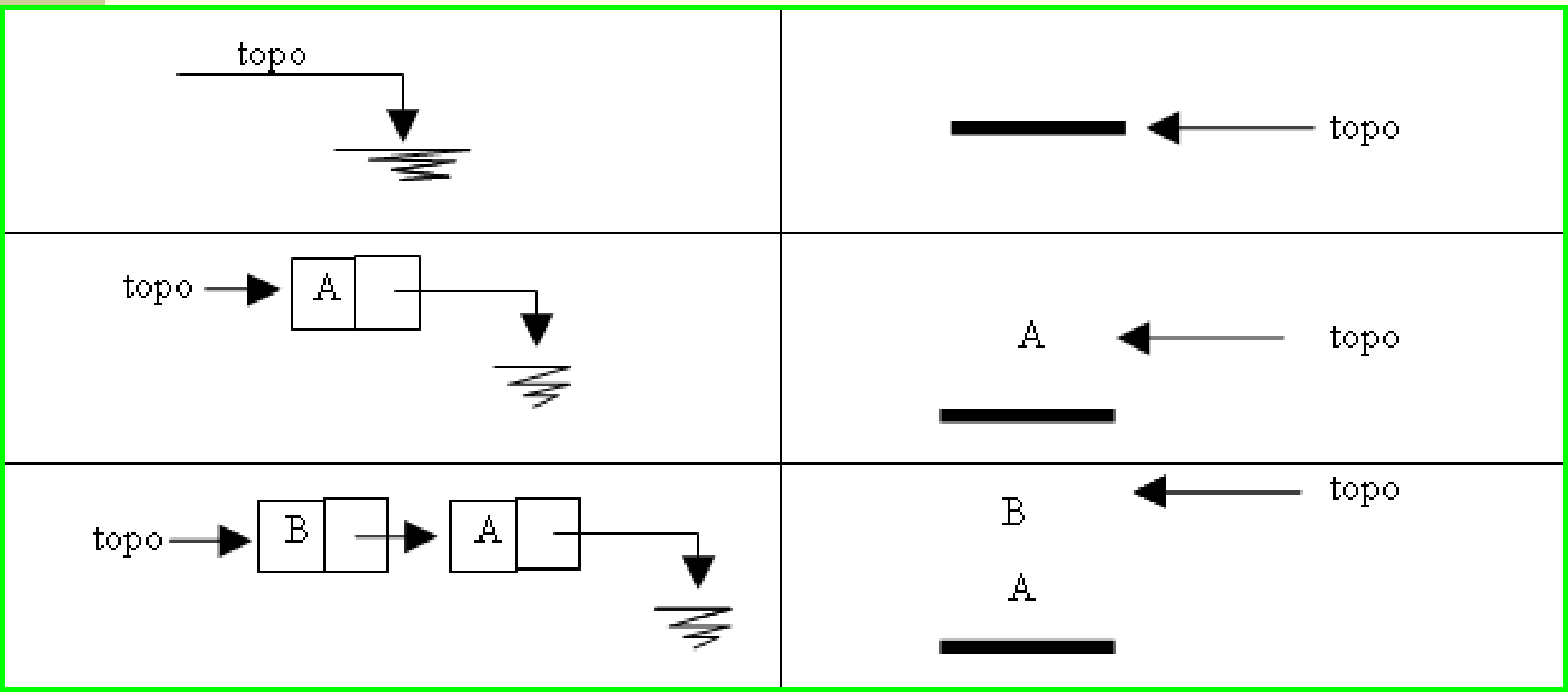
- Considerando a representação de lista anterior, implemente o TAD lista com as seguintes operações
  - `cria(lista)`
  - `finaliza(lista)`
  - `inserir(x)`
  - `eliminar(x)`
    - recursiva e não recursiva
  - `tamanho(lista)`
    - recursiva e não recursiva
  - `esta_na_lista(x)`
    - recursiva e não recursiva
  - `imprimir(lista)`

Pilha

- Lista linear: pilha

- Represente graficamente o funcionamento da pilha, representando a pilha vazia, a entrada e a saída de elementos
  - Quais e quantos ponteiros são necessários?

Pilha



Exercício

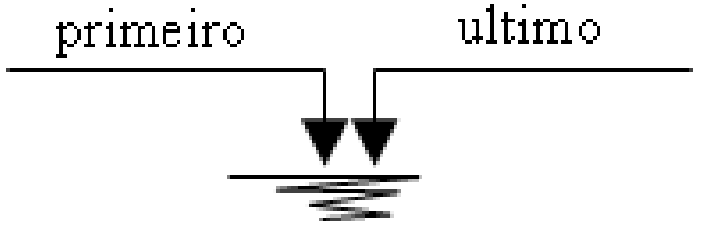
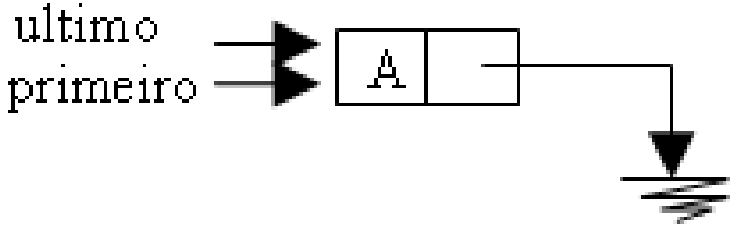
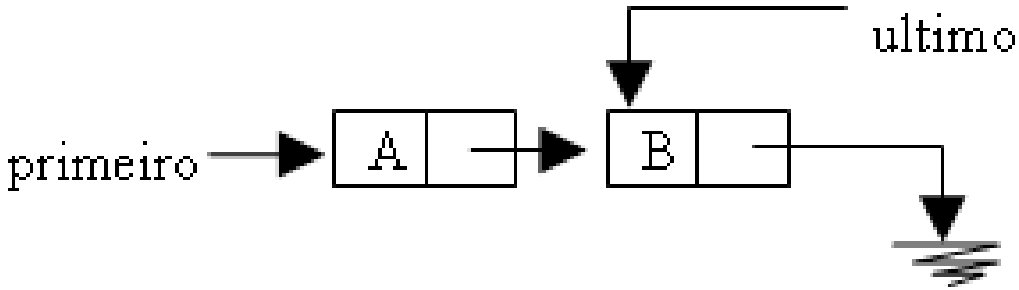
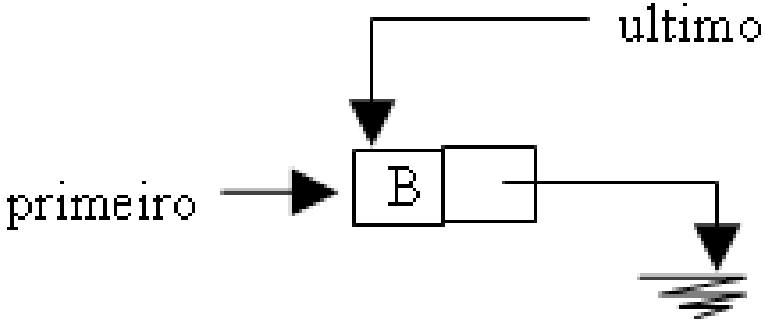
- Implementar as rotinas da pilha utilizando a lista encadeada e dinâmica

`Create`, `Push`, `Pop`, `IsEmpty`

Fila

- Lista linear: fila

- Represente graficamente o funcionamento da fila, representando a fila vazia, a entrada e a saída de elementos
  - Quais e quantos ponteiros são necessários?

fila vazia	
entra A	
entra B	
sai o primeiro	

Exercício

- Implementar as rotinas da fila utilizando a lista encadeada e dinâmica
  - `Cria`, `Entra`, `Sai`, `IsEmpty`

Lista

- Pode ser utilizada para representar qualquer estrutura
  - Não apenas pilhas e filas
- Operações diversas
- Formas diversas (linear, não-linear)

Listas

- Estão entre as estruturas mais flexíveis e versáteis
  - Quaisquer formatos de informação
  - 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

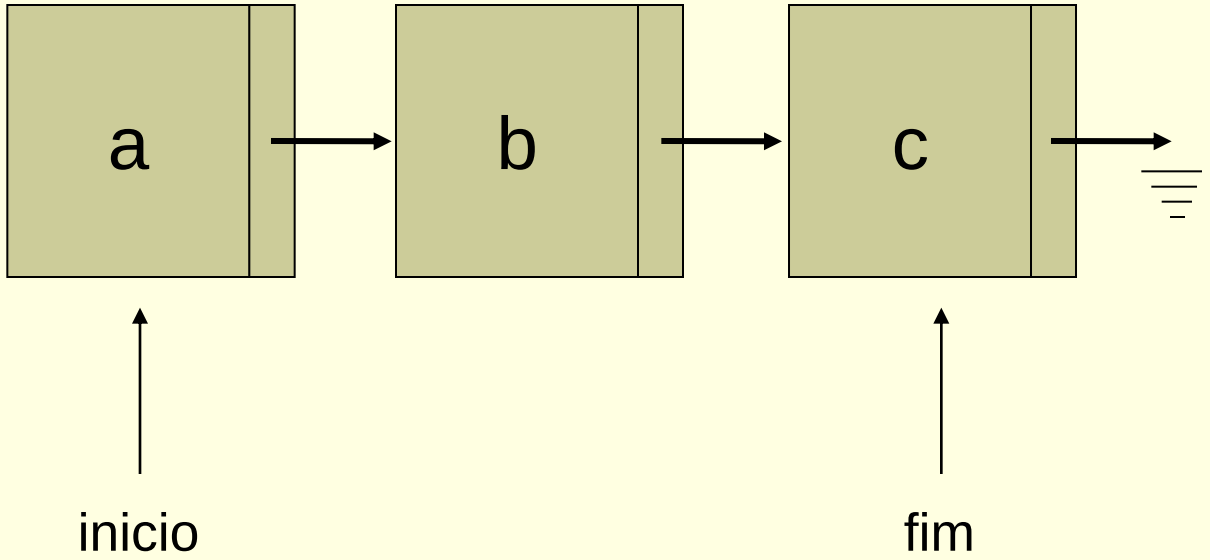
Velocidade de acesso à informação

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

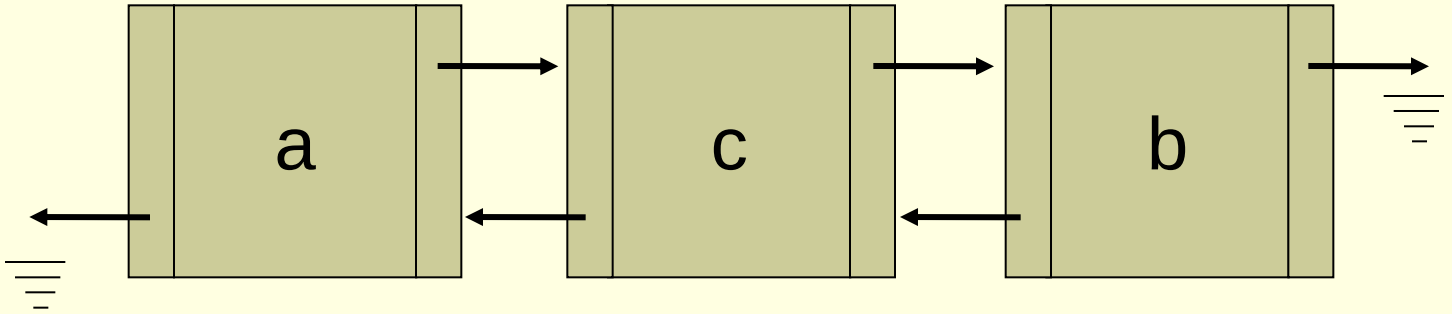
Variações da lista

- Lista simples, ordenada



Variações da lista

- Lista duplamente encadeada
  - Facilita navegação



Variações da lista

- Lista duplamente encadeada
  - Facilita navegação

```
typedef struct bloco {
    char nome[50];
    struct bloco *ant, *prox;
} no;
```

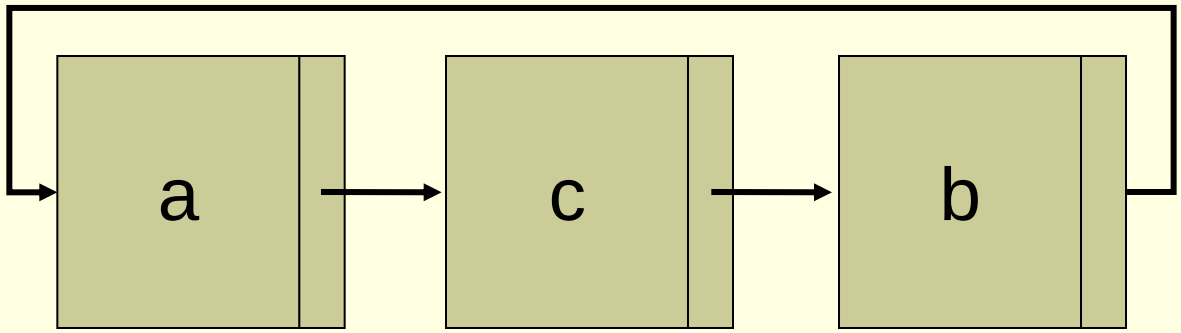
Variações da lista

- Lista duplamente encadeada
  - Facilita navegação

- Exercício: 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>número de elementos da lista, inserir no fim

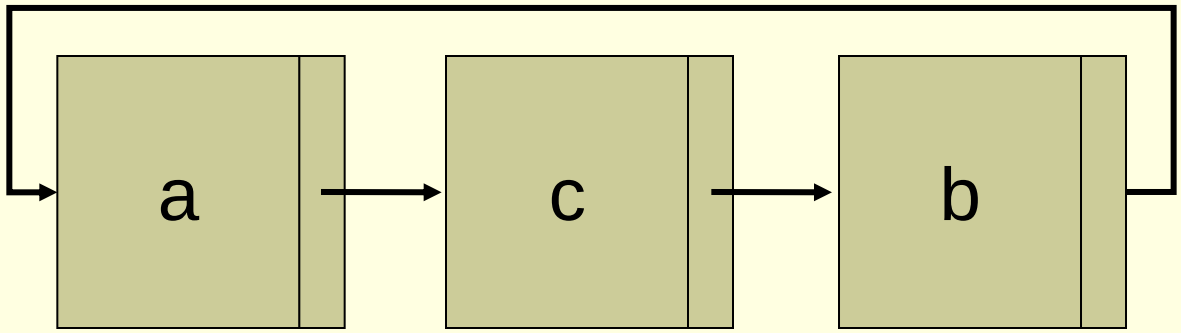
Variações da lista

- Lista circular



Variações da lista

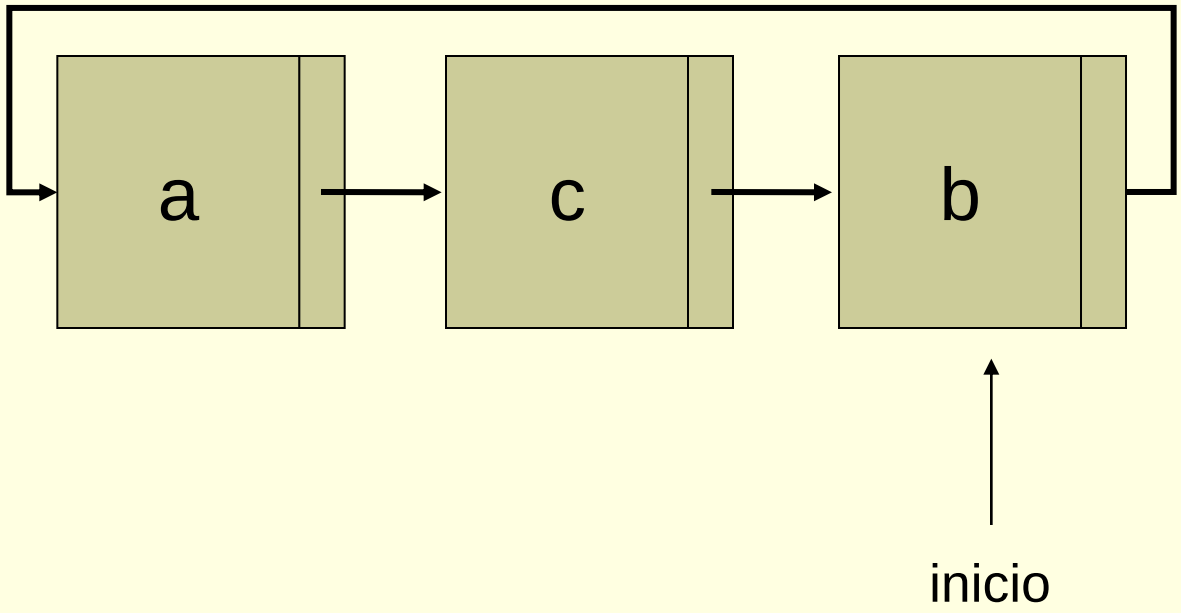
- Lista circular



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

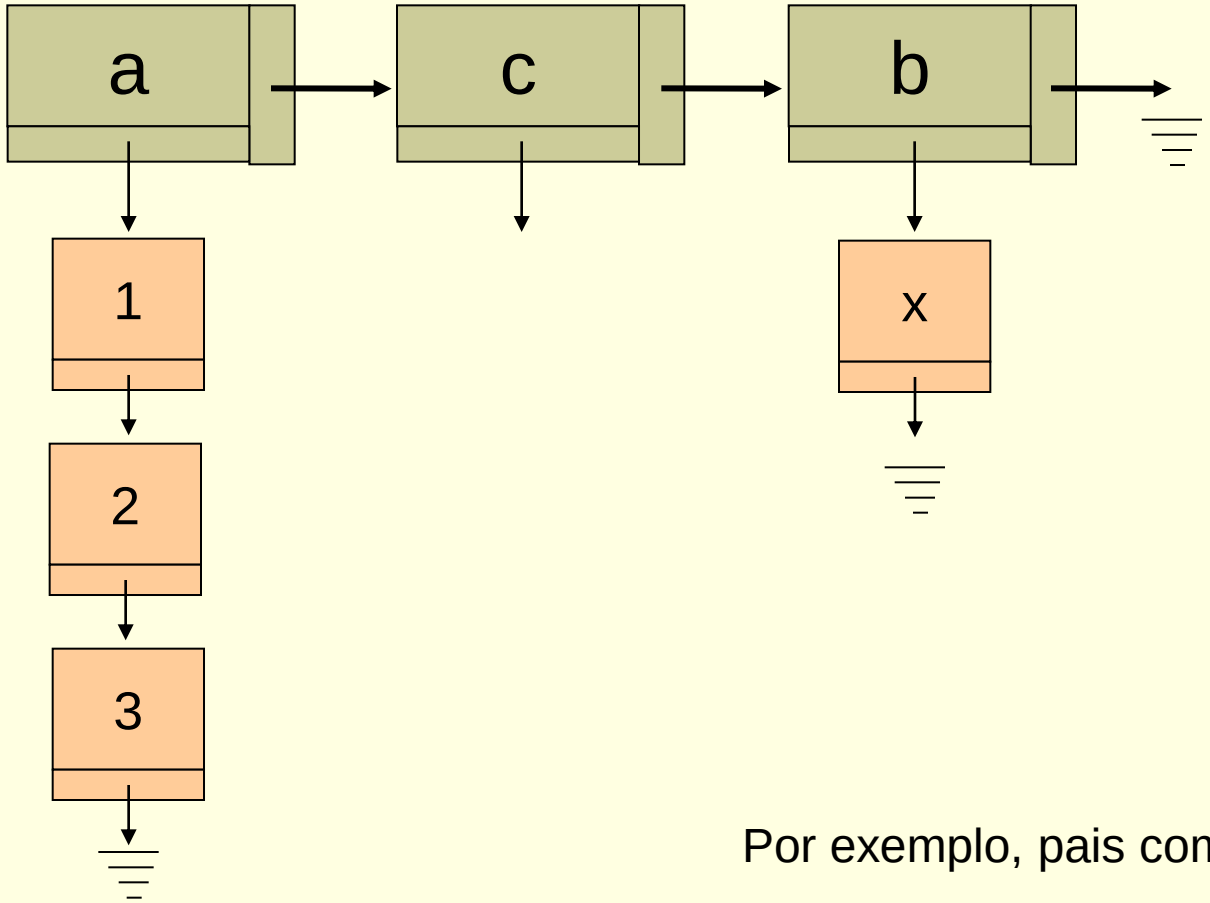
Variações da lista

- Lista circular



Variações da lista

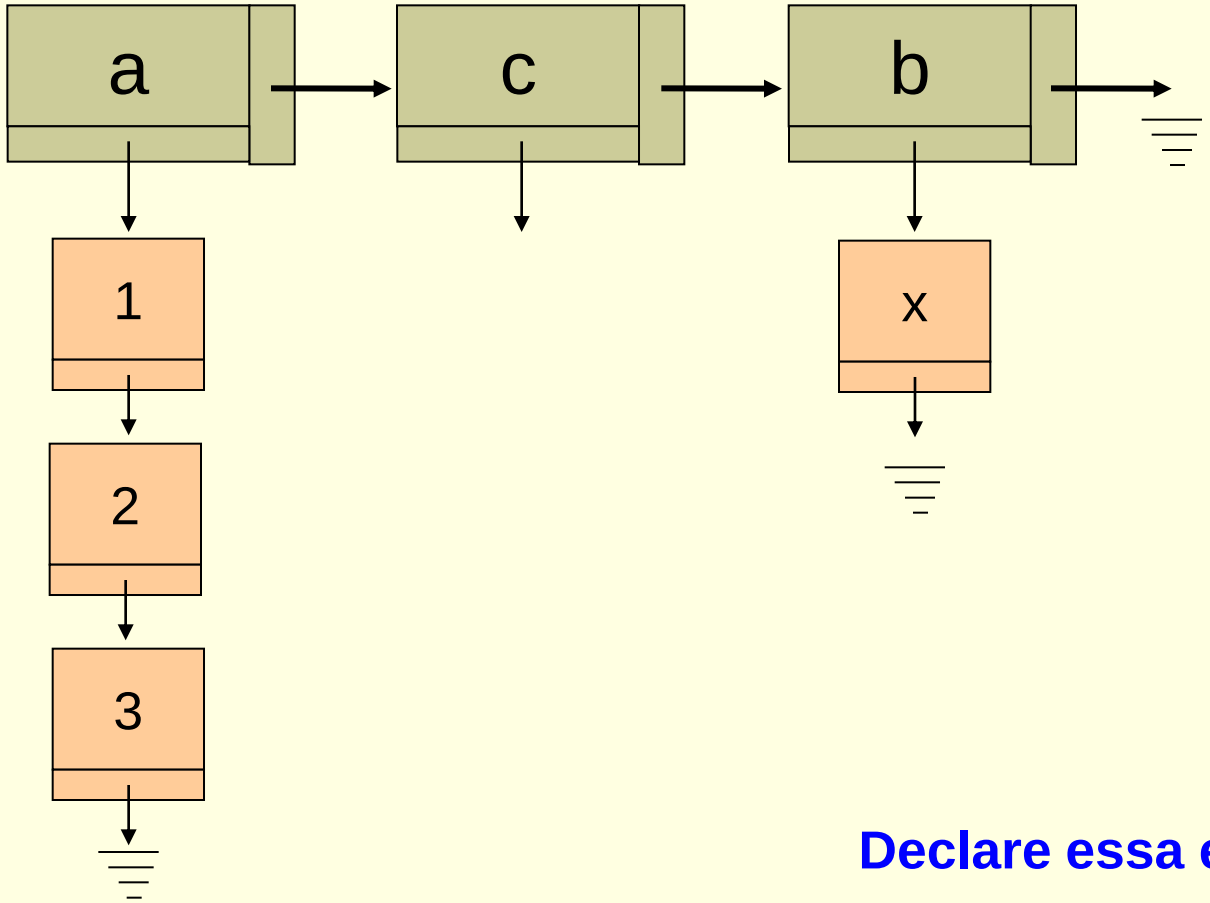
- Exemplo de lista não linear
  - Informações de diferentes níveis



Por exemplo, pais com seus filhos

Variações da lista

- Exemplo de lista não linear
  - Informações de diferentes níveis



Declare essa estrutura!

Variações da lista

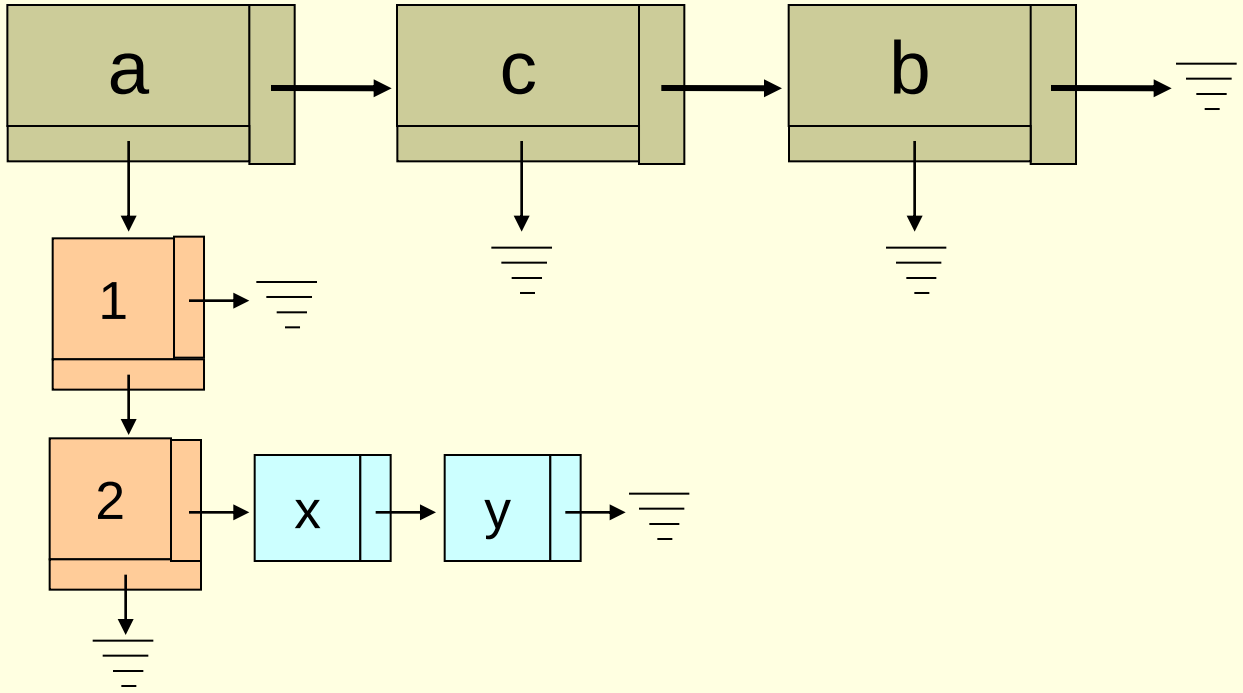
- Lista com blocos de N ponteiros
  - Informações de diferentes níveis

```
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
```

Variações da lista

- Outro exemplo de lista não linear
  - Informações de diferentes níveis
  - Tão complexo e elaborado quanto preferir



Exercício

- Faça os diagramas necessários e declare uma lista de mães que têm filhos estudando no ICMC, sendo que:
  - Cada mãe 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 sua própria mãe

Facilidade de manipulação da lista

- Operações a serem efetuadas sobre os dados
  - Organização dos dados
  - Ponteiros

- Perguntas
  - Quais as operações mais freqüentes?
  - Qual o custo computacional de cada operação?

Créditos

- *Material gentilmente cedido pelo Prof. Thiago A. S. Pardo*