ESTRUTURA DE DADOS I

Aula 10

Prof. Sérgio Luis Antonello

FHO - Fundação Hermínio Ometto

05/05/2025

Plano de Ensino

1. Unidade I – Métodos de ordenação em memória principal (objetivos d, e, f)
   1.1. Revisão de tipos de dados básicos em C, variáveis indexadas
   1.2. Recursividade
   1.3. Noções de complexidade computacional
   1.4. Conceitos e métodos de ordenação de dados
   1.5. Bubble sort, Insert sort e Select sort
   1.6. Quick sort e Merge sort
   1.7. Shell sort e Radix sort
2. Unidade II – Métodos de pesquisa em memória principal (objetivos e, f)
   2.1. Pesquisa sequencial
   2.2. Pesquisa binária
   2.3. Hashing
3. Unidade III – Tipo abstrato de dados (TAD) (objetivo a)
   3.1. Revisão de registros, ponteiros e alocação dinâmica de memória
   3.2. Tipo abstrato de dados (TAD): conceitos e aplicações
4. Unidade IV – Estrutura de dados lineares (objetivos a, b, c)
   4.1. Lista Encadeada: conceitos e aplicações
   4.2. Pilha: conceitos e aplicações
   4.3. Fila: conceitos e aplicações

Cronograma do Plano de Ensino

- 28/04 - Devolutiva P1; Tipo Abstrato de Dados (TAD).
- 05/05 - Conceitos de estruturas lineares: Lista ligada; Pilha; Fila.
- 12/05 - Algoritmos para Lista Simplesmente Encadeada.
- 19/05 - Algoritmos para Lista Simplesmente Encadeada.
- 26/05 - Implementação de Pilha e de Fila.
- 02/06 - Semana Científica do Curso.
- 09/06 - Desenvolvimento do trabalho A2.
- **16/06 - Prova 2.**
- **23/06 - Prova SUB.**

Sumário

- Primeiro momento (Revisão)
  - TAD
- Segundo momento (conteúdo)
  - Conceitos sobre estrutura de dados lineares
    - Listas
    - Fila
    - Pilha
    - Listas simplesmente e duplamente encadeadas, circular e ordenada.
- Terceiro momento (síntese)
  - Retome pontos importantes da aula

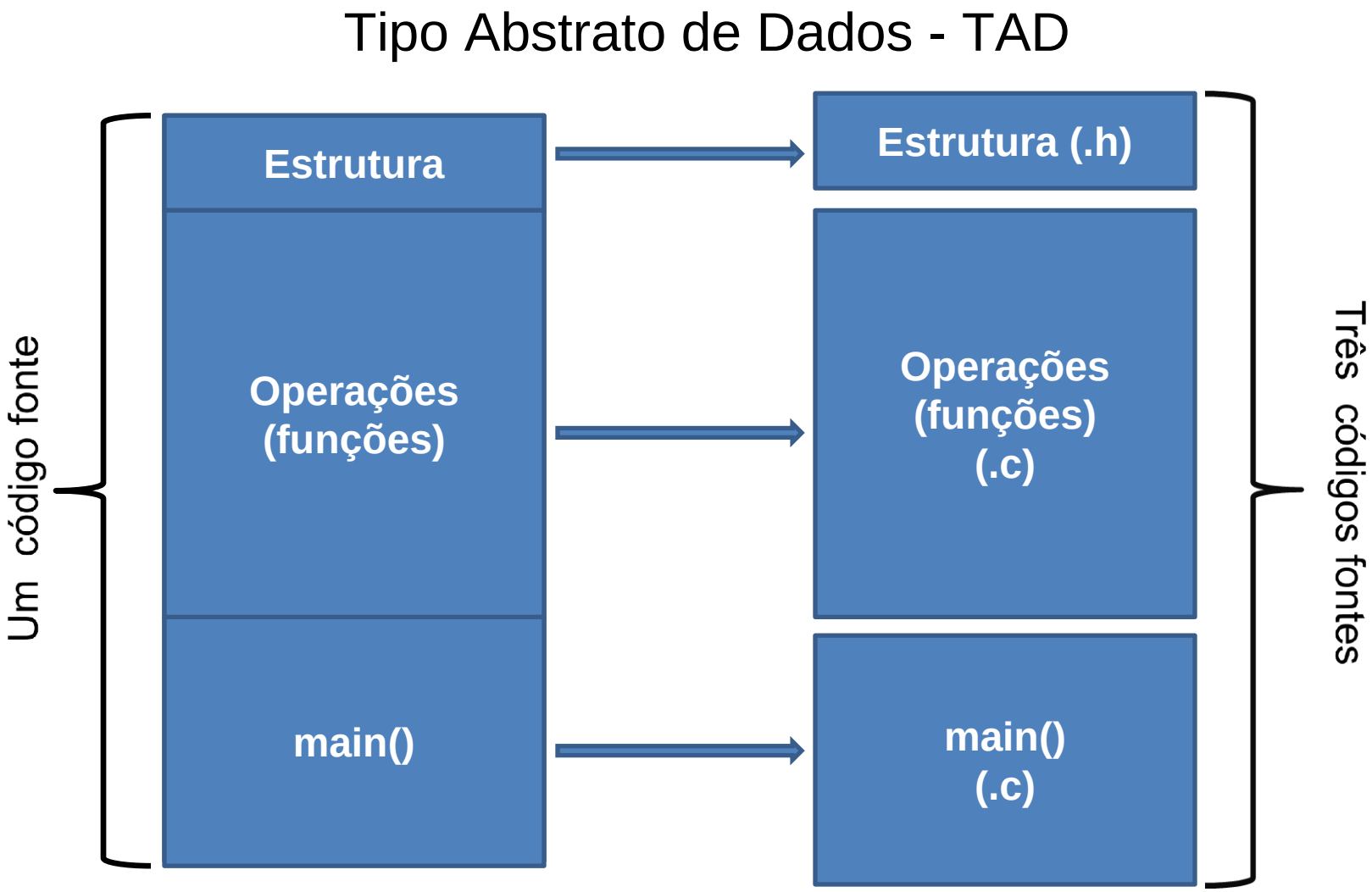
1. Primeiro momento: Revisão

- ED sobre TADs

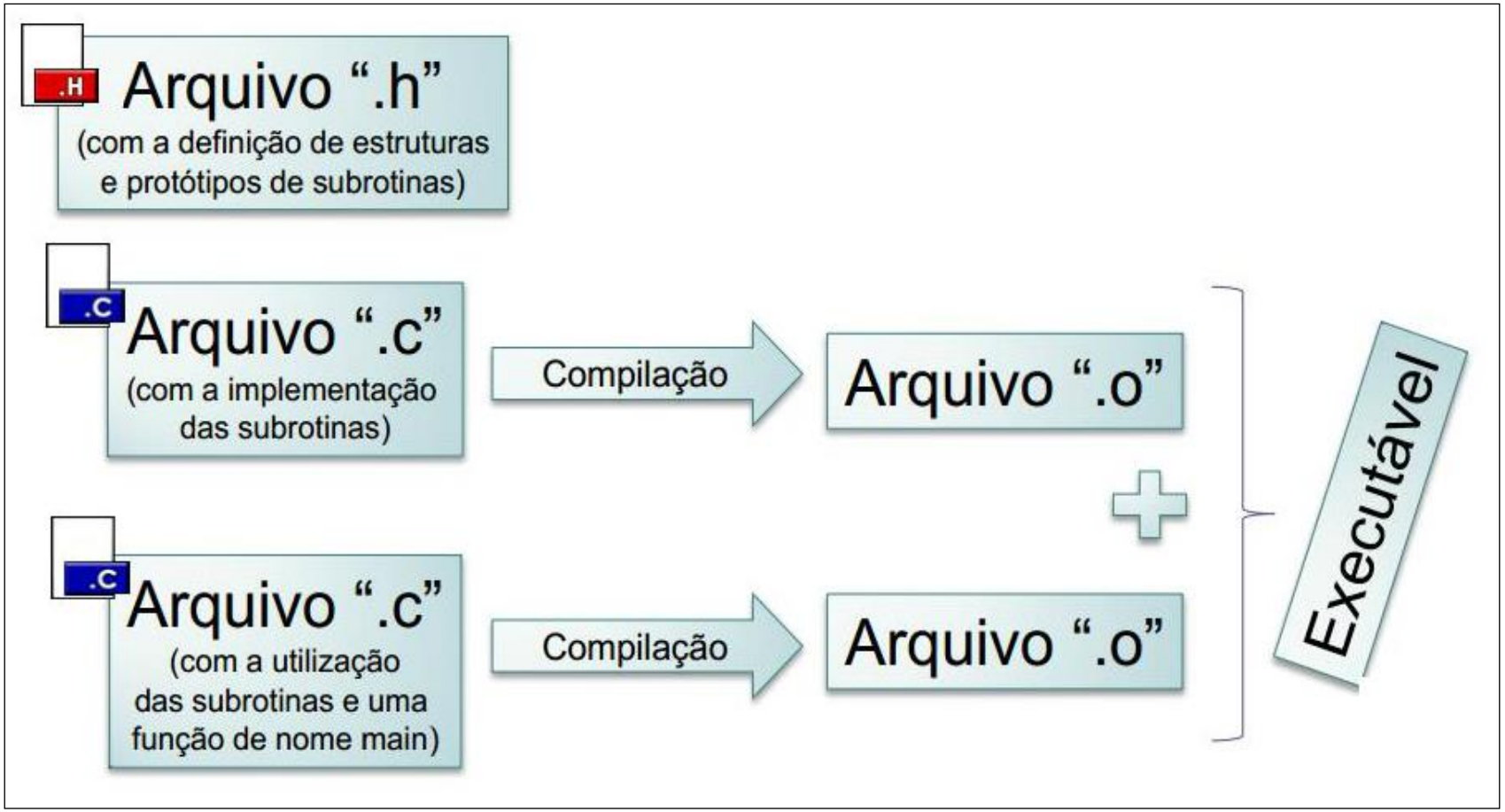
TAD	Dados	Operações
Estudante	Struct: char nome[] int idade int ra float notas[] flat media	void fazerAniversário() int gerarRA() void atribuirNota() float calcularMedia()

- Como pode ser divido a implementação de um TAD?

1. Primeiro momento: Revisão



1. Primeiro momento: Revisão



1. Primeiro momento: Revisão

O nome tipos abstrato de dados vem da matemática, do estudo de estruturas algébricas compostas por domínios de valores e de operações sobre esses domínios. Um domínio de valores corresponde em programação a um tipo, e as operações sobre valores do domínio correspondem a procedimentos e funções que manipulam os valores do tipo. Usamos o adjetivo abstrato para enfatizar que devemos nos abstrair da forma exata de implementação, durante a utilização do tipo, levando em consideração apenas as propriedades especificadas dos valores do tipo e das operações (Rangel, 1993).

Referência Bibliográfica

RANGEL, J. L. **Linguagens de programação.** Rio de Janeiro, 1993.

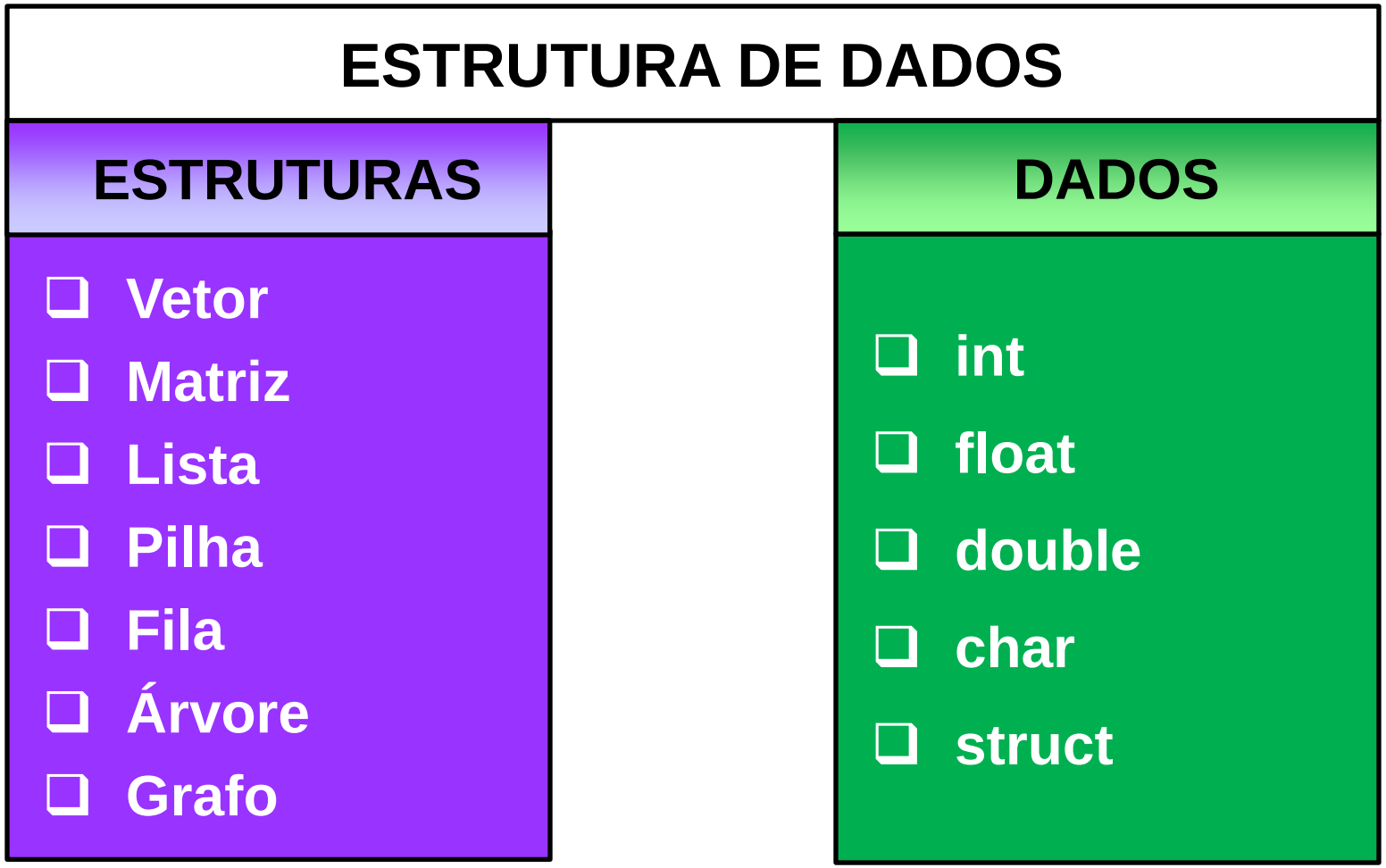
1. Primeiro momento: revisão

Correção de exercícios

2. Segundo momento

Estruturas de Dados Lineares

2. Estrutura de Dados (ED)



2. Estrutura de Dados (ED)

EDs comumente utilizadas na computação:

- Estáticas: Vetores, Matrizes e Registros (structs).
- Dinâmicas e Lineares: **Listas**, **Filas** e **Pilhas**.
- Dinâmicas e Não-Lineares: Grafos e Árvores.

Observação: alguns autores consideram os Registros (structs) como estrutura de dados heterogêneas, pois permitem o armazenamento de diferentes tipos de dados em uma mesma estrutura.

2. Estrutura de Dados (ED)

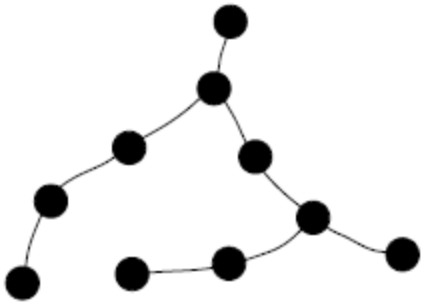
- Exemplos de EDs sobreTADs

TAD	Dados	Operações
Estudante	Struct: char nome[] int idade int ra float notas[] flat media	void fazerAniversário() int gerarRA() void atribuirNota() float calcularMedia()
Sacola	Struct: int MAX int numItens Item itens[]	void addItem() void delItem() bool sacolaVazia() bool sacolaCheia()

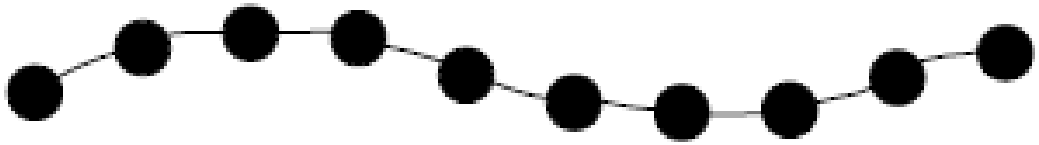
2. Estruturas lineares x não lineares

- **Listas**, **Filas** e **Pilhas** são chamadas de **Estruturas Lineares -** organizam-se em forma de “linha”.

- Árvores e Grafos são chamados de **Estruturas Não-Lineares** - não organizam-se em forma de “linha”.



2. Estruturas lineares



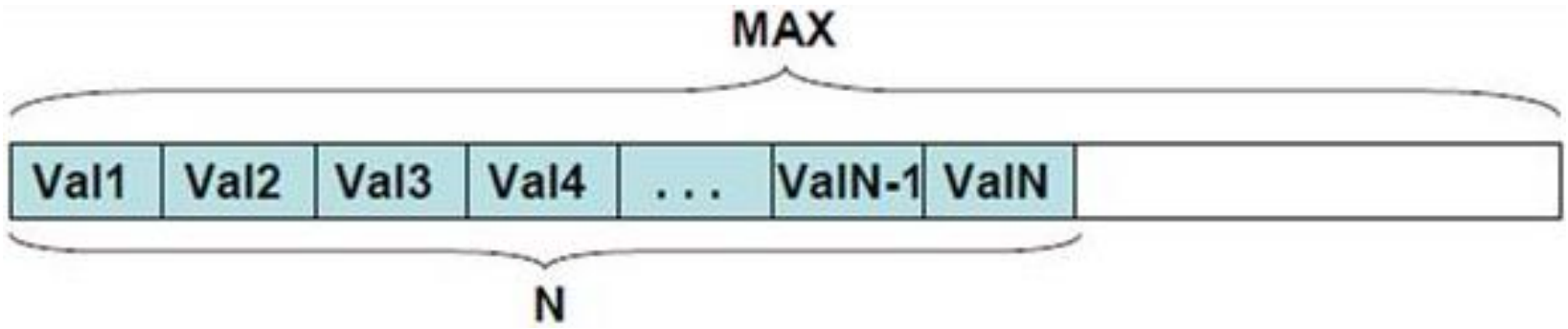
- Cada elemento (bola preta) é chamado de **nó**;
- Os nós são organizados de maneira sequencial;
- Apenas um único nó pode ser alcançado por vez;
- Segue uma ordem de conexão entre os nós;
- Podem ser **Estáticas** ou **Dinâmicas**.

2. Estruturas lineares **estáticas**

- Possuem tamanho fixo;
- Alocadas na memória na execução da declaração;
- Não podem crescer para além deste tamanho;
- São implementadas com vetores de tamanho fixo;
- Cada nó é uma posição do vetor.

2. Estruturas lineares **estáticas**

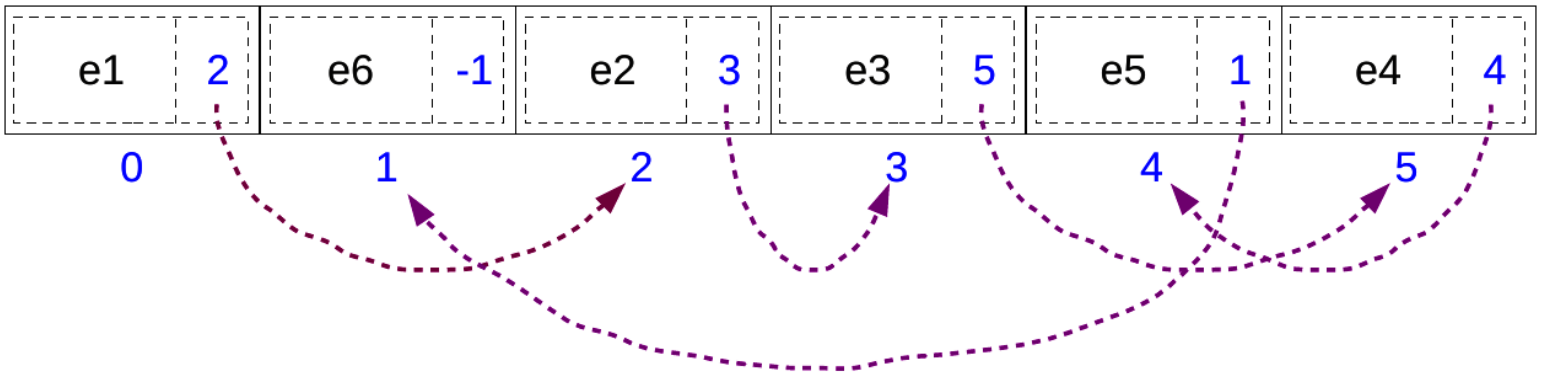
- **Lista Estática Sequencial:** o nó é conectado ao próximo nó pela posição subsequente no vetor.



2. Estruturas lineares **estáticas**

- **Lista Estática Não-Sequencial:** os nós são conectados por meio de “ponteiros“ que “apontam“ para o índice da posição do próximo nó.

```
typedef struct {
    int dado;
    int proximo;
} No;
```



2. Estruturas lineares **dinâmicas**

- O tamanho máximo não é pré-determinado;
- A estrutura aumenta com a adição de um novo elemento;
- A estrutura diminui quando elementos são retirados;
- Podem ser de três tipos:
  - **Listas Encadeadas**
  - **Filas**
  - **Pilhas**

3. Lista encadeada (ligada)

- Do inglês: *Linked List* - **Lista Ligada**
- Um nó a ser adicionado na estrutura deve, primeiramente, ser alocado em memória e só depois ligado aos demais nós já devidamente alocados.
- Cada nó da estrutura possui, ao menos, **um ponteiro** que aponta para a real posição de memória do próximo nó.

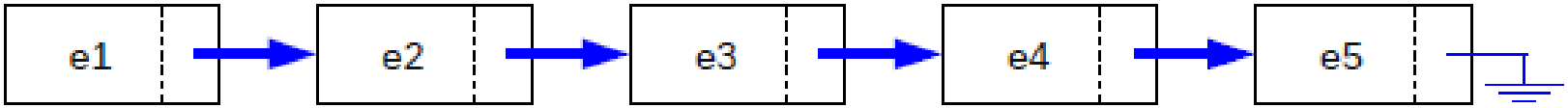
3. Lista encadeada (ligada)

- Pode haver diferentes tipos de Lista Ligada.
- Em relação à direção dos ponteiros:
  - **Lista Simplesmente Encadeada**
  - **Lista Duplamente Encadeada**
- Em relação à ligação dos ponteiros dos nós da última posição da estrutura:
  - **Lista Encadeada Não-Circular**
  - **Lista Encadeada Circular**

4. Lista **simplesmente** encadeada

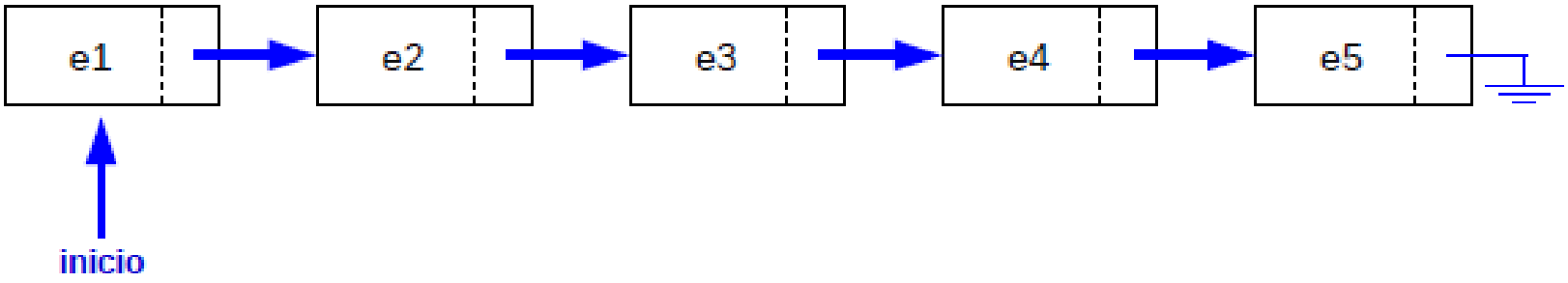
- Cada nó da estrutura possui um ponteiro que aponta para a real posição de memória do próximo nó.

```
typedef struct _no {
    int dado;
    struct _no *proximo;
} No;
```

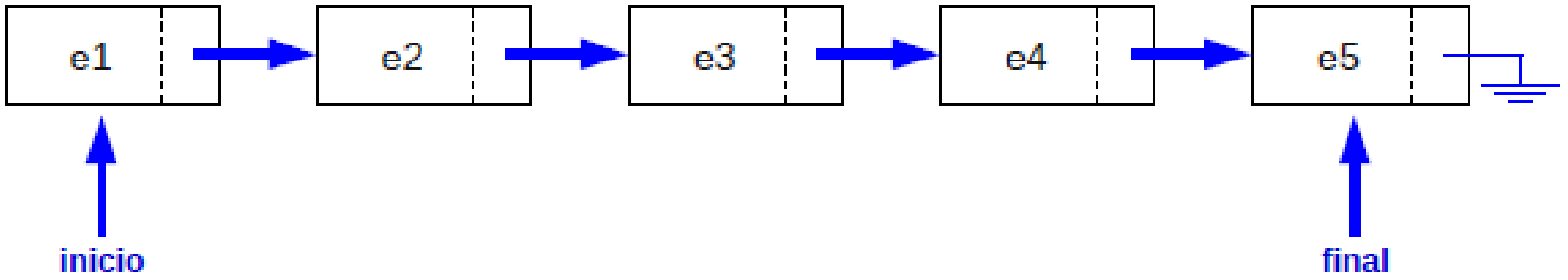


4. Lista **simplesmente** encadeada

- Tem um ponteiro que sempre aponta para o início da lista (permite o acesso à estrutura).



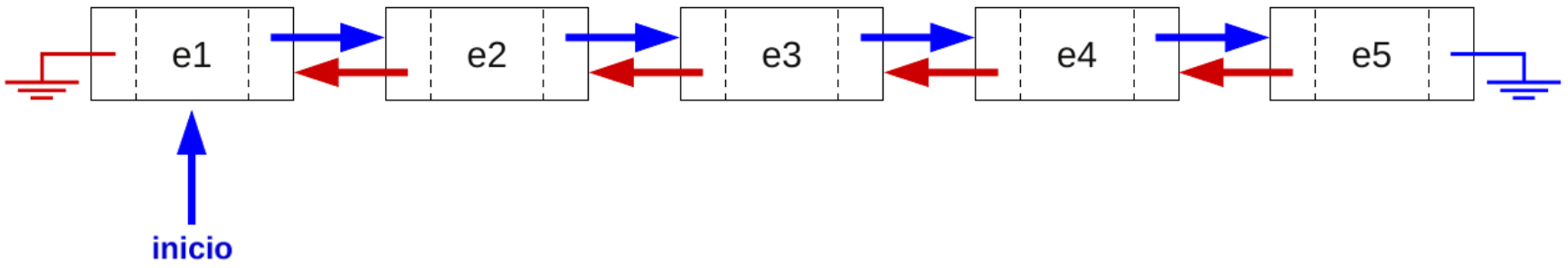
- Pode ter, também, um ponteiro que sempre aponta para o final da lista.



5. Lista **duplamente** encadeada

- Cada nó da estrutura possui dois ponteiros: um que aponta o **próximo** nó e outro que aponta para o nó **anterior**.

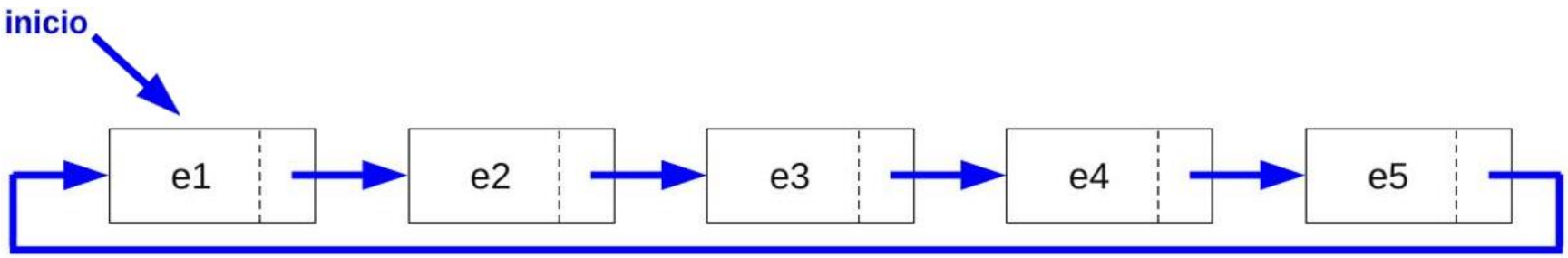
```
typedef struct _no {
    int dado;
    struct _no *proximo;
    struct _no *anterior;
} No;
```



6. Lista encadeada **circular**

- O ponteiro do último nó da estrutura aponta para a posição de memória do primeiro nó.

```
typedef struct _no {
    int dado;
    struct _no *proximo;
} No;
```

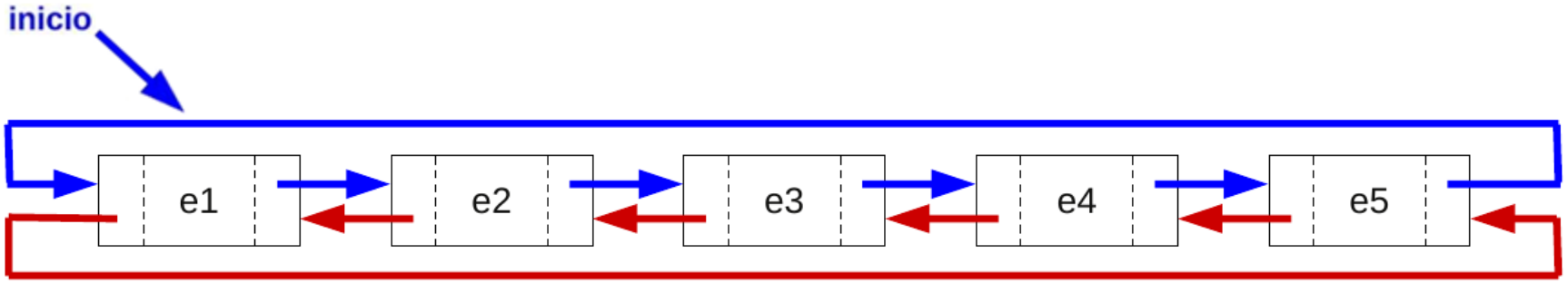


Questão: como interromper uma busca por um elemento inexistente?

6. Lista **duplamente** encadeada e **circular**

- Estrutura duplamente encadeada onde o nó de um extremo aponta para o nó do outro extremo.

```
typedef struct _no {
    int dado;
    struct _no *proximo;
    struct _no *anterior;
} No;
```



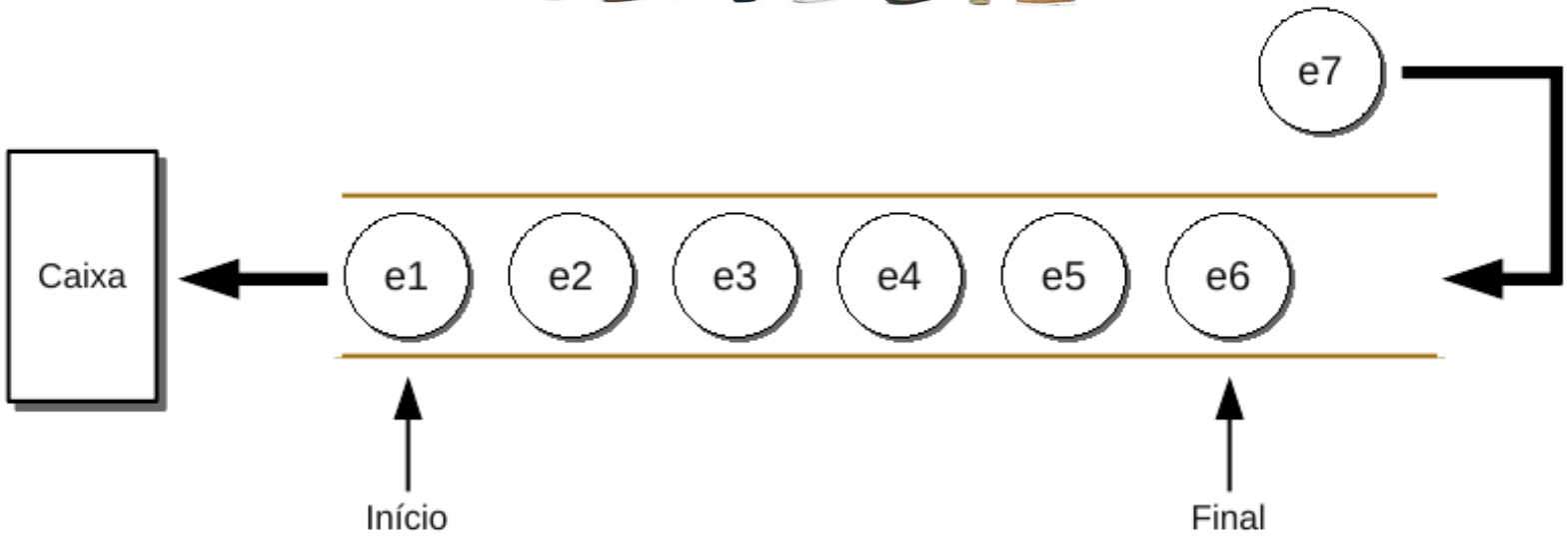
7. Lista encadeada: operações

- **Operações** (funções) de listas encadeadas:
  - Inicializar a Lista
  - Adicionar elemento (em qualquer posição) na Lista
  - Remover elemento (de qualquer posição) da Lista
  - Pesquisar/Buscar um elemento na Lista
  - Alterar dados de um elemento da Lista
  - Verificar se a Lista está vazia
  - Esvaziar a Lista
  - Contar o número de elementos atuais na Lista
  - Imprimir os dados dos elementos existentes na Lista
  - ...

8. Fila

- Do inglês: *Queue*
- Contém a mesma estrutura de **Listas Simplesmente Encadeadas**
- A diferença está na inserção e remoção de elementos:
  - A **inserção** ocorre sempre no **final** da estrutura
  - A **remoção** ocorre sempre do **início** da estrutura
- Filas são estruturas do tipo "**FIFO**" (*First In, First Out*)
  - "O primeiro elemento a entrar é o primeiro a sair"

8. Fila



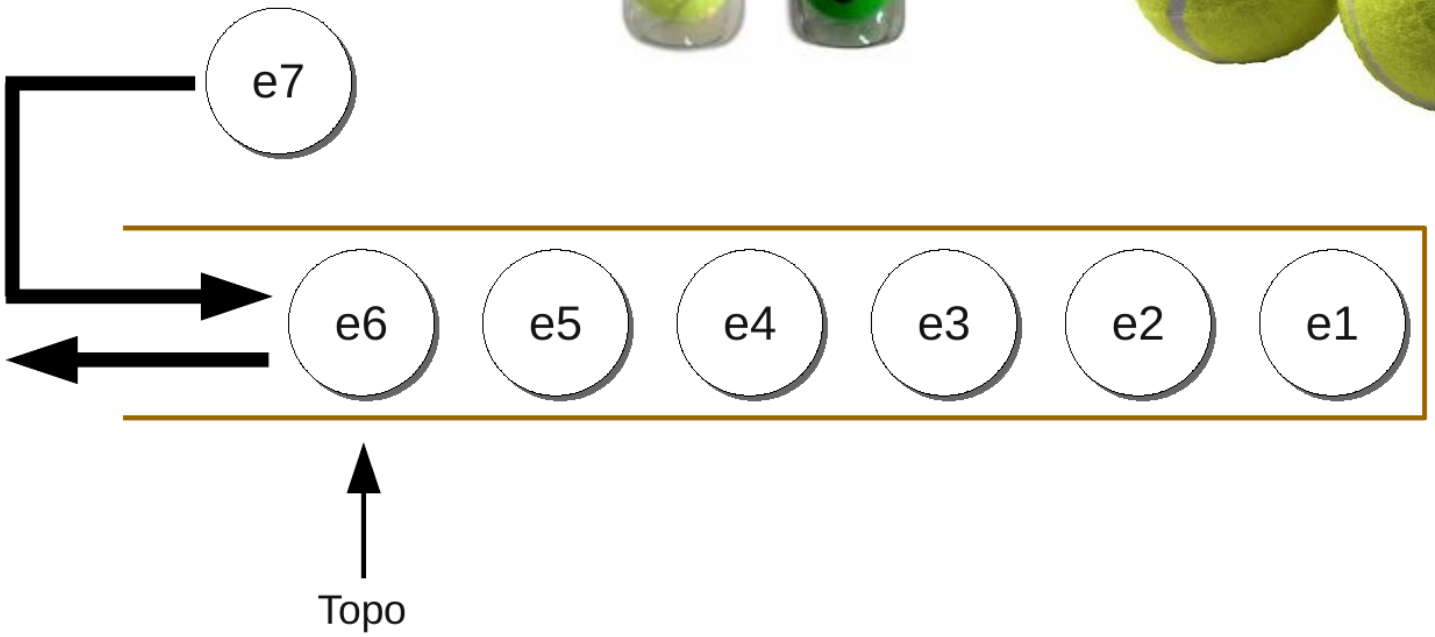
8. Fila

- **Operações** (funções) de filas:
  - Inicializar a Fila
  - Adicionar elemento (**no final**) da Fila
  - Remover elemento (**do início**) da Fila
  - Alterar os dados de um elemento da Fila
  - Verificar se um elemento existe na Fila
  - Verificar se a Fila está vazia
  - Esvaziar a Fila
  - Contar o número de elementos atuais na Fila
  - Imprimir os dados dos elementos da Fila
  - ...

9. Pilha

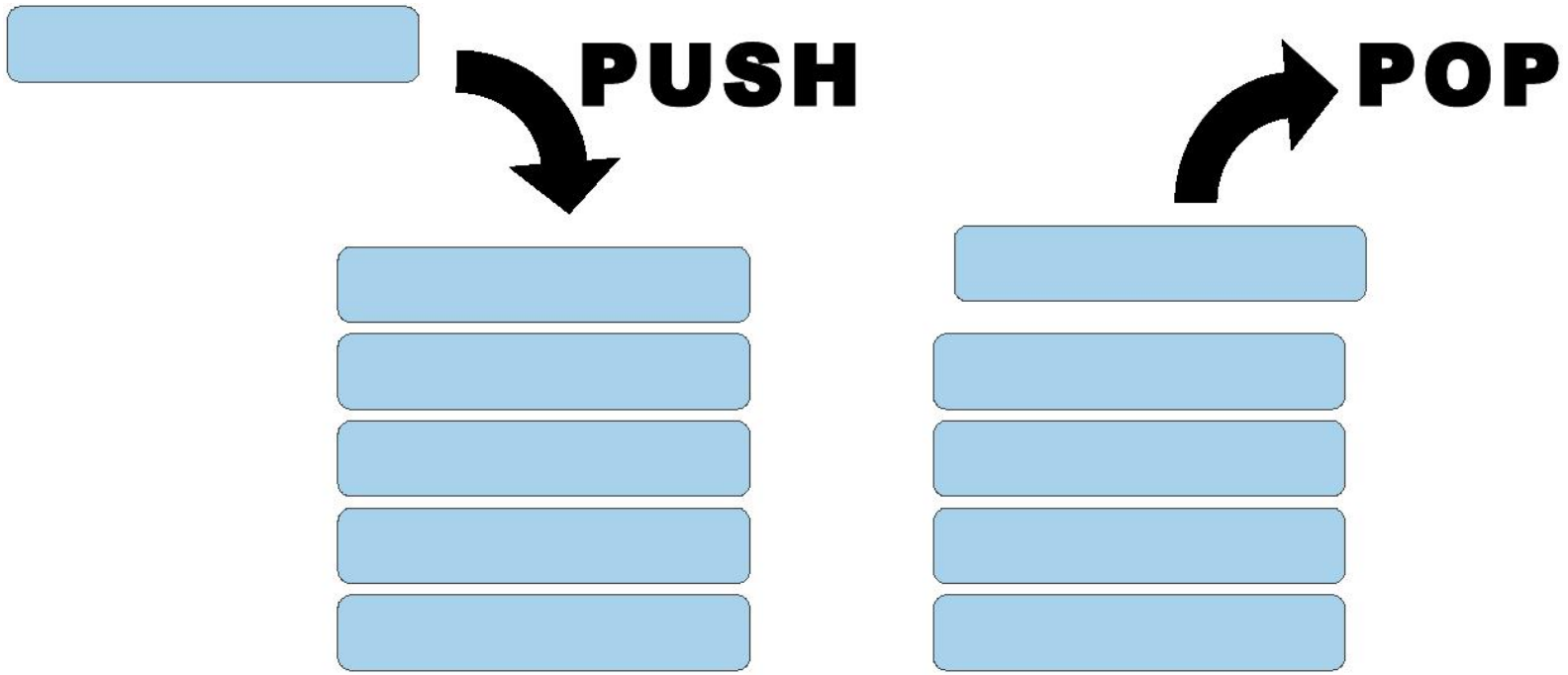
- Do inglês: *Stack*
- Contém a mesma estrutura de **Listas Simplesmente Encadeadas**
- A diferença está na inserção e remoção de elementos:
  - A **inserção** ocorre sempre no **topo** (início) da estrutura
  - A **remoção** ocorre sempre do **topo** (início) da estrutura
- Pilhas são estruturas do tipo "**LIFO**" (*Last In, First Out*)
  - "O último elemento a entrar é o primeiro a sair"

9. Pilha



9. Pilha

- **Operações** (funções) de pilhas:



9. Pilha

- **Operações** (funções) de pilhas:
  - Inicializar a Fila
  - **PUSH** (adicionar elemento no topo da Pilha)
  - **POP** (remover o elemento do topo da Pilha)
  - **TOP** (ler os dados do elemento do topo da Pilha)
  - **PULL** (alterar os dados do elemento do topo da Pilha)
  - Verificar se a Pilha está vazia
  - Esvaziar a Pilha
  - Contar o número de elementos atuais na Pilha
  - Imprimir os dados dos elementos da Pilha
  - ...

10. Simuladores

- Simuladores de estruturas lineares:
  - Listas, Filas e Pilhas
    https://visualgo.net/pt/list
  - Filas
    https://www.cs.usfca.edu/~galles/visualization/QueueLL.html
  - Pilhas
    https://www.cs.usfca.edu/~galles/visualization/StackLL.html

11. Exercícios

Vamos Praticar!

11. Exercícios

1) No desenvolvimento de um software, foram utilizadas as estruturas de dados: pilha e fila. Considere que, se uma sequência representa uma pilha, o topo da pilha é o elemento mais à esquerda; e se uma sequência representa uma fila, a frente da fila é o elemento mais à esquerda.

   Foram realizadas algumas ações, como seguem:

   - ✓ Durante a execução do programa uma sequência de cinco números inteiros, a saber: 13, 15, 40, 45 e 50, foram armazenados em uma fila, do maior valor para o menor valor.
   - ✓ Logo depois, cada elemento foi retirado da fila e inserido em uma pilha.
   - ✓ Por fim, todos os elementos foram retirados da pilha e armazenados em uma outra pilha.

   Mostre como ficaram os elementos em cada uma das estruturas, principalmente a sequência final dos números após inserção dos mesmos na última pilha.

11. Exercícios

2) Considere uma estrutura de dados do tipo LIFO. Entidades são inseridas nessa estrutura com a operação push() e removidas com a operação pop().

   Considere, também, que a extremidade esquerda representa o início (topo) da estrutura, mostre o resultado de cada uma das operações abaixo.

   push(8), push(7), push(5), push(2), pop(), push(8), push(7), pop(), push(5), push(2), pop(), pop().

   Se possível, desenhe uma imagem das estruturas preenchidas.

11. Exercícios

3) Usando os conceitos de TAD, desenvolver um programa dividido em três arquivos.

   O arquivo “header.h” deve ter a definição da estrutura da lista e declarações das funções das operações nessa estrutura.

   O arquivo “operacoes.c” deve conter a codificação necessária para:

   a) inicializar a lista;
   b) adicionar elemento (em qualquer posição - início, meio e final) na lista;
   c) remover elemento (de qualquer posição - início, meio e final) da lista;
   d) pesquisar/buscar um elemento na lista;
   e) alterar dados de um elemento da lista;
   f) verificar se a lista está vazia;
   g) esvaziar a lista;
   h) contar o número de elementos na lista;
   i) exibir os dados dos elementos existentes na lista.

   O arquivo “lista.c” deve conter o programa principal que faça uso dessas operações e simule dados para operacionalizar a lista.

4. Exercícios

a) Codificar as seguintes operações auxiliares

- inicializar a lista
- verificar se a lista está vazia
- exibir todo conteúdo da lista

4. Exercícios

Pseudocódigo de operações auxiliares

- Inicializar a lista

```
void inicializaLista (No **lista) {

    fazer *lista igual a nulo

} /* fim da funcao inicializaLista */
```

4. Exercícios

Pseudocódigo de operações auxiliares

- Verificar se a lista está vazia

```
int listaVazia (No *lista) {

    se conteúdo de lista for igual a nulo
        retornar um

    retornaz zero

} /* fim da funcao listaVazia */
```

4. Exercícios

Pseudocódigo de operações auxiliares

- Exibir todo conteúdo da lista

```
void imprimeLista (No *lista) {

    declarar no auxiliar
    fazer no auxiliar igaul a lista

    testar se a lista esta vazia
        exibir esagem de lista vazia
        encerrar a função

    fazer loop enquanto ponteiro auxiliar for diferente de nulo
        exibir conteúdo do nó
        fazer ponteiro auxilar ser igual ao atributo próximo do ponteiro auxilar

} /* fim da funcao imprimeLista */
```

12. Terceiro momento: síntese

- Listas lineares são recursos computacionais importantes que possibilitam algoritmos que implementam Filas e Pilhas, muito usados na computação.
- As listas ligadas (encadeadas) podem ser classificadas como **simplesmente** ou **duplamente** encadeada.
- Elas também podem ser classificadas como **circulares**.
- Ainda podem ser classificadas como **ordenadas**.