

ESTRUTURA DE DADOS I

Aula 11

Prof. Sérgio Luis Antonello

FHO - Fundação Hermínio Ometto

12/05/2025

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 e recursividade
 - 1.2. Noções de complexidade computacional
 - 1.3. Conceitos e métodos de ordenação de dados
 - 1.4. Bubblesort, Insertsort e Selectsort
 - 1.5. Quicksort e Mergesort
 - 1.6. Shellsort e Radixsort
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 e 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.
- 15/06 - *Deadline* da atividade bônus para a P2.
- **16/06 - Prova 2.**
- **23/06 - Prova SUB.**

Sumário

■ Primeiro momento

- Listas ligadas

■ Segundo momento

- Desenvolver rotinas para aplicação de listas ligadas

- inicializar a lista
- verificar se a lista esta vazia
- imprimir nós da lista
- adicionar nó no início da lista
- adicionar nó no final
- remover nó do início da lista
- remover nó do final da lista
- adicionar nó em uma posição específica
- adicionar nó em lista ordenada
- remover nó do meio da lista

■ Terceiro momento

- Síntese da aula

1. Primeiro momento: Revisão

■ Estrutura de Dados Lineares



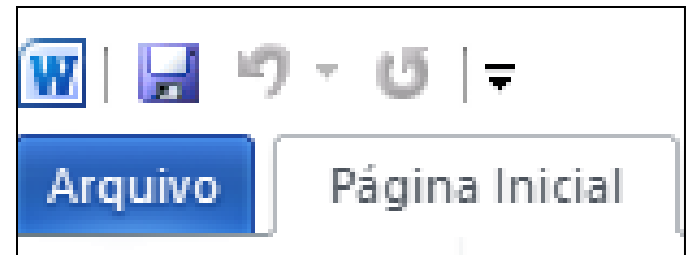
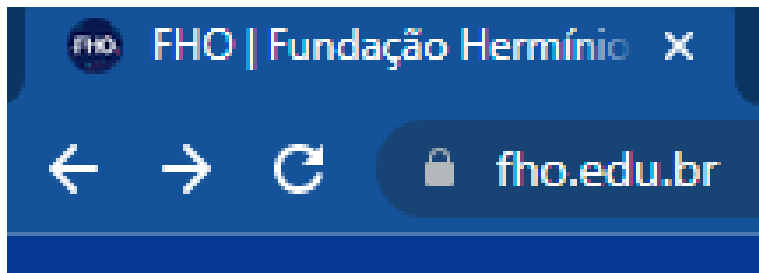
■ Estruturas lineares dinâmicas











- Listas ligadas ou encadeadas (linked list)
- Filas (queue)
- Pilhas (stack)



1. Primeiro momento: Revisão

- Gerenciamento de projetos: organizar as tarefas de um projeto em uma ordem específica.
- Rastreamento de produto: rastrear todo o processo de uma cadeia produtiva.
- Histórico de transações: armazenar informações transacionais do negócio. Ex: transações financeiras.



| FHO-UNIRARAS RANK | |
|--------------------------|--|
| FUNDAÇÃO HERMÍNIO OMETTO | |
| | |
| RANKING | USUÁRIO |
| 1 |  csantbr |
| 2 |  ElGuigs |
| 3 |  aisha_petronilho |
| 4 |  goAhead |
| 5 |  Vinicius_413 |
| 6 |  Sullivan |
| 7 |  Prata_ |
| 8 |  LuccaMenegatti |
| 9 |  IcaroVieira |
| 10 |  MatheusQuinalha |

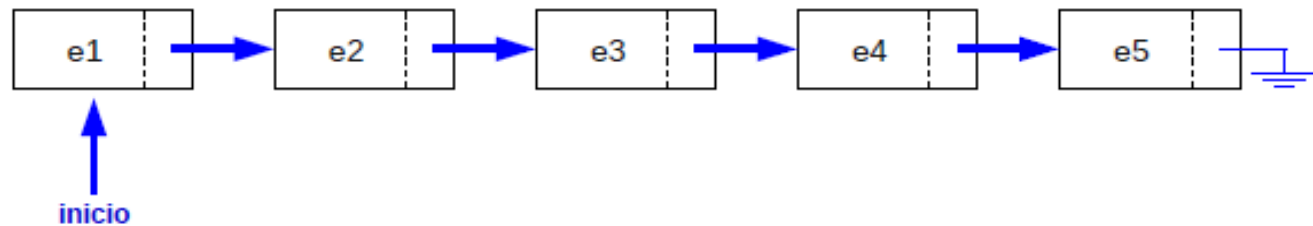
1. Primeiro momento: Revisão

- 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**.

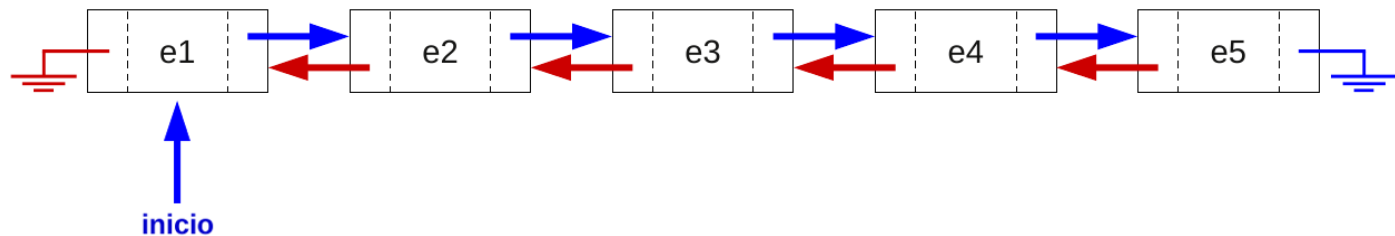
1. Revisão: Lista encadeada

- Em relação à direção dos ponteiros:

- **Lista Simplesmente Encadeada**



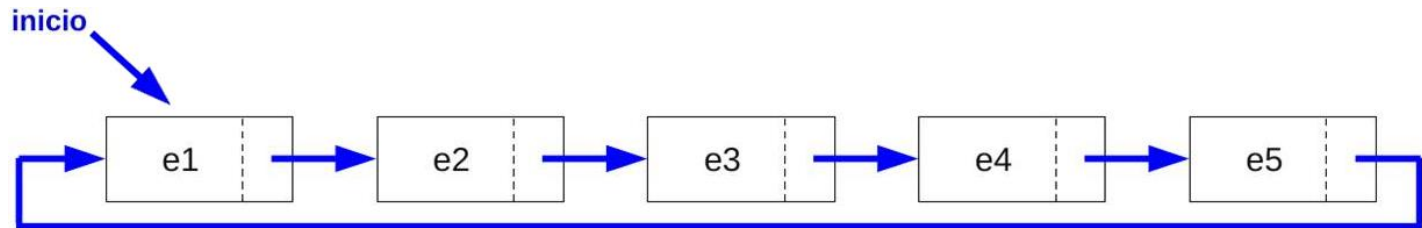
- **Lista Duplamente Encadeada**



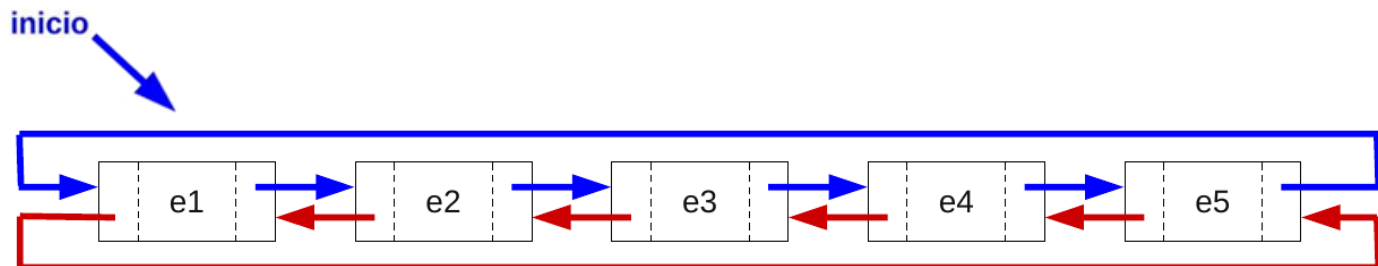
1. Revisão: Lista encadeada

- Em relação à ligação dos ponteiros dos nós da última posição da estrutura:

- **Lista Encadeada Não-Circular**



- **Lista Encadeada Circular**



1. Revisão: Lista encadeada

■ Simuladores de estruturas lineares:

■ Listas, Filas e Pilhas

<https://visualgo.net/pt/list>

■ Filas

<https://www.cs.usfca.edu/~galles/visualization/QueueLL.html>

■ Pilhas

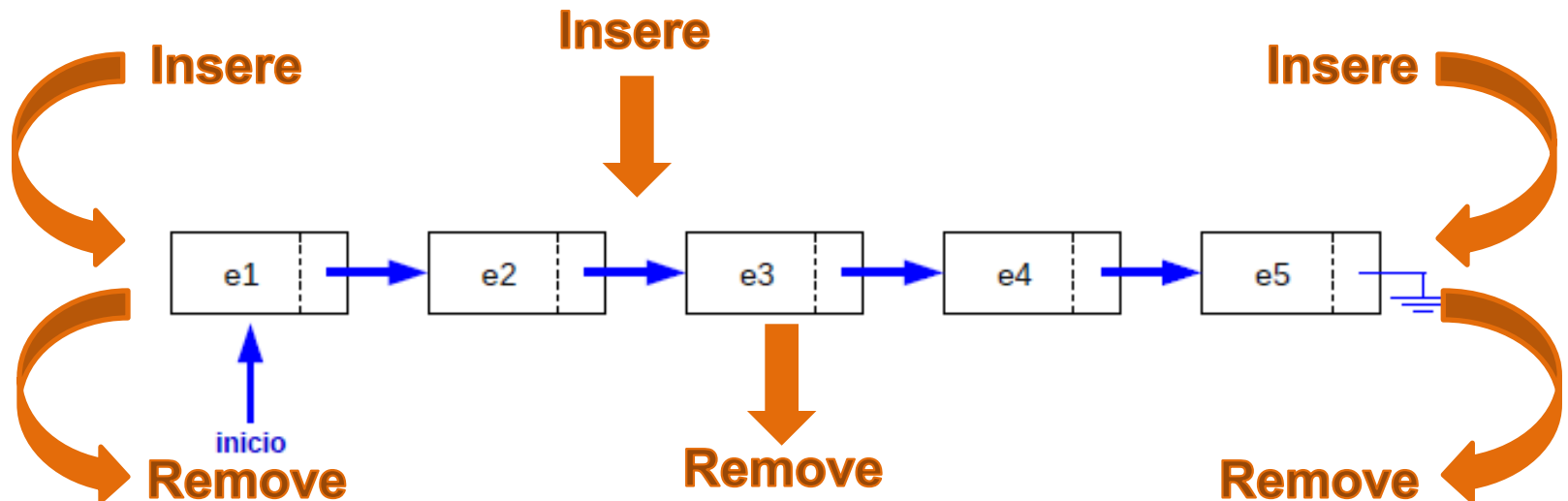
<https://www.cs.usfca.edu/~galles/visualization/StackLL.html>

2. Segundo momento

Operações em lista ligada

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

```
No *inicio;
```



3. Operações em lista encadeada

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

```
No *inicio;
```

- Baseada no tipo definido e em um ponteiro “início” que mostra o primeiro nó da lista, qual abstração necessária para:
 - inicializar a lista ✓
 - verificar se a lista esta vazia ✓
 - imprimir nós da lista ✓
 - adicionar nó no início da lista
 - adicionar nó no final da lista
 - remover nó do início da lista
 - remover nó do final da lista
 - pesquisar se um ID está na lista
 - adicionar nó em lista ordenada
 - remover nó do meio da lista

4. Exercícios

**Vamos
Praticar!**



4. Exercícios

- Considere a estrutura do nó dada abaixo
- Considere um ponteiro “início” que aponta para o primeiro nó da lista
- Considere que o endereço do ponteiro “início” será passado como parâmetro

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

```
No *inicio;
```

4. Exercícios

a) Codificar as seguintes operações

- adicionar elemento no início
- adicionar elemento no final
- pesquisar se um elemento está na lista
- remover elemento do início
- remover elemento do final

4. Exercícios

Adicionar elemento no início

```
void insereInicio (No **lista, int dado) {  
  
    declarar ponteiro para receber novo nó  
  
    /* cria o novo noh a ser inserido na lista */  
    alocar memória e atribuir para novo nó  
    alimentar dados para o novo nó  
  
    /* se a lista estiver vazia, novo noh aponta para NULL */  
    testar se a lista está vazia  
    .....  
    atribuir nulo para o atributo próximo do novo nó  
  
    /* caso contrario, novo noh aponta para o inicio da lista */  
    se a lista não está vazia  
    .....  
    atribuir *lista para o atributo próximo do novo nó  
  
    /* novo noh passa a ser o inicio da lista */  
    fazer o início da lista ser o novo nó  
} /* fim da funcao insereInicio */
```


4. Exercícios

Adicionar elemento no final

```
void insereFinal (No **lista, int dado) {
```

```
    declarar ponteiro para receber novo nó  
    declarar ponteiro auxiliar para andar na lista  
    fazer ponteiro auxiliar igual a *lista
```

```
    /* cria o novo noh a ser inserido na lista */
```

```
    alocar memória e atribuir para novo nó  
    alimentar dados para o novo nó  
    fazer atributo próximo do novo nó igual a nulo  
    novo->proximo = NULL;
```

```
    /* se a lista estiver vazia, novo noh passa a ser o inicio da lista */
```

```
    testar se a lista está vazia  
    fazer *lista igual ao novo nó  
    encerrar a função
```

```
    /* caso contrario, caminha na lista ate parar no ultimo noh */
```

```
    fazer loop enquanto atributo próximo do ponteiro auxiliar for diferente de nulo  
    fazer ponteiro auxiliar ser igual ao atributo próximo do ponteiro auxiliar
```

```
    /* ultimo noh aponta para o novo noh */
```

```
    fazer atributo próximo do nó auxiliar igual ao ponteiro novo
```

```
} /* fim da funcao insereFinal */
```

4. Exercícios

Pesquisar se um elemento está na lista

```
No* pesquisaElemento (No *lista, int dado) {  
  
    declarar ponteiro auxiliar para andar na lista  
    fazer ponteiro auxiliar igual a lista (início d lista)  
  
    /* caso contrario, caminha na lista ate encontrar o elemento */  
    fazer loop enquanto ponteiro auxiliar for diferente de nulo e atributo  
    dado do ponteiro auxiliar for deferente do parâmetro dado  
        fazer ponteiro auxiliar ser igual ao atributo próximo do ponteiro auxiliar  
  
    /* retorna o ponteiro para o elemento encontrado */  
    /* (se o elemento nao foi encontrado, retorna NULL) */  
    retornar o ponteiro auxiliar  
  
} /* fim da funcao pesquisaElemento */
```

4. Exercícios

Remover elemento do início

```
void removeInicio (No **lista) {  
  
    declarar ponteiro auxiliar  
    fazer ponteiro auxiliar igual a *lista  
  
    testar se a lista está vazia  
    |  
    encerrar a função  
  
    /* inicio da lista passa a ser o segundo noh (ou NULL) */  
    fazer *lista igual ao atribut próximo do ponteiro auxiliar  
  
    /* remove o primeiro noh da memoria */  
    liberar memória apontada pelo ponteiro auxiliar;  
  
} /* fim da funcao removeInicio */
```

4. Exercícios

Remover elemento do final

```
void removeFinal (No **lista) {
    declarar ponteiro auxiliar para andar na lista
    fazer ponteiro auxiliar igual a *lista (início d lista)
    declarar ponteiro anterior para ajudar no reapontamento
    fazer ponteiro anterior igual a *lista

    testar se a lista está vazia
    encerrar a função

    /* caminha ate o final da lista */
    fazer loop enquanto atributo próximo do ponteiro auxiliar for diferente de nulo
        fazer ponteiro anterior igual ao ponteiro auxiliar
        fazer ponteiro auxiliar ser igual ao atributo próximo do ponteiro auxiliar

    /* se houver apenas um elemento, entao, inicio torna-se NULL */
    se o ponteiro auxiliar for igual a *lista (início da lista)
        fazer inicio da lista igual a nulo
    /* caso contrario, remove o ultimo noh da lista */
    senao
        fazer atributo próximo do ponteiro anterior igual a nulo

    /* remove o utlimo noh da memoria */
    liberar memória apontada pelo ponteiro auxiliar;
} /* fim da funcao removeFinal */
```

5. Terceiro momento: Síntese

- Se for necessário processar todos os nós da lista ligada, basta caminhar na lista a partir de quem o ponteiro “início” aponta (primeiro nó) e por meio do ponteiro “próximo” de cada nó ir acessando sequencialmente os demais nós da estrutura.
- A inserção de um nó em uma lista ligada exige abstrações diferentes se ela ocorrer no início, no meio ou no final da lista.
- Do mesmo modo, a remoção de um nó de uma lista ligada exige abstrações diferentes se ela ocorrer no início, no meio ou no final da lista.
- Nós implementamos a lista ligada que contém um ponteiro “início” que aponta para o primeiro nó da lista. Porém, a inserção e remoção no final da lista pode ser diferente se existir um ponteiro “final” que aponta para o último nó da lista.