

ESTRUTURA DE DADOS I

Aula 12

Prof. Sérgio Luis Antonello

FHO - Fundação Hermínio Ometto

19/05/2025

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; 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.**



Bônus P2

Quantos estão com bônus na prova 2 (levantamento em 16/05)?

1 alunos com 1.0 de bônus	0 alunos com 0,5 de bônus
0 alunos com 0,9 de bônus	2 alunos com 0,4 de bônus
0 alunos com 0,8 de bônus	2 alunos com 0,3 de bônus
0 alunos com 0,7 de bônus	1 alunos com 0,2 de bônus
0 alunos com 0,6 de bônus	1 alunos com 0,1 de bônus



 **beecrowd** academic


Olá, Antonello
antonello@uniararas.br 

VOCÊ ESTÁ AQUI DISCIPLINAS / ESTRUTURA DE DADOS I - TURMA A 2025














LISTA DE EXERCÍCIOS

CONSTRUA LISTAS COM PROBLEMAS DO NOSSO REPOSITÓRIO!

NOVA

#	ID	LISTA DE EXERCÍCIOS	TÉRMINO	OPÇÕES
1	048305	Trabalho A1	7 de abril de 2025 22:10	    
2	049244	Bônus nota P2	15 de junho de 2025 23:57	    

Bônus P2

10		PUC-GOIAS	Pontifícia Universidade Católica de Goiás	125.364
11		INATEL	Instituto Nacional de Telecomunicações	120.790
12		UNIFOR	Universidade de Fortaleza	115.810
13		IFSULMINAS-GERAL	Instituto Federal do Sul de Minas Gerais	113.076
14		UNB-GAMA	Universidade de Brasília - Faculdade do Gama	106.513
15		IFCE-GERAL	Instituto Federal do Ceará	105.031
16		IFPB-GERAL	Instituto Federal da Paraíba	97.687
17		FMM	Fundação Matias Machline	91.593
18		URI	Universidade Regional Integrada do Alto Uruguai e das Missões	79.940
19		IFRN-GERAL	Instituto Federal do Rio Grande do Norte	79.279
20		FHO-UNIARARAS	Fundação Hermínio Ometto	77.399
21		CEFETMG	Centro Federal de Educação Tecnológica - Minas Gerais	75.552
22		UTN	Universidad Tecnológica Nacional	73.982

Sumário

■ Primeiro momento

■ Rotinas para aplicação de listas ligadas

- Exibir todo conteúdo 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

■ Segundo momento

■ Desenvolver rotinas para aplicação de listas ligadas

- Pesquisar se um ID está na lista
- Adicionar nó em uma lista ordenada
- Remover um nó a partir do seu ID
- Esvaziar a lista

■ Terceiro momento

■ Síntese da aula

1. Primeiro momento: Revisão

Inicializar lista; testar se a lista está vazia

```
8  /* funcao que inicializa a lista ----- */
9  void inicializaLista (No **lista) {
10
11      *lista = NULL;
12
13  } /* fim da funcao inicializaLista */
14
15  /* funcao que verifica se a lista esta vazia - */
16  int listaVazia (No *lista) {
17
18      if (lista == NULL)
19          return 1;
20
21      return 0;
22  } /* fim da funcao listaVazia */
```

1. Primeiro momento: Revisão

Exibir todo conteúdo da lista

```
25  /* funcao que imprime o conteudo da lista ----- */
26  void imprimeLista (No *lista) {
27
28      No *aux = lista;
29
30      if (listaVazia(lista) ){
31          printf("A lista esta vazia!\n");
32          return;
33      }
34
35      printf("Lista:  [  ");
36
37      while (aux != NULL) {
38          printf("%d  ", aux->dado);
39          aux = aux->proximo;
40      }
41
42      printf("]\n");
43
44  } /* fim da funcao imprimeLista */
```


1. Primeiro momento: Revisão

Adicionar nó no início da lista

```
61  /* funcao que insere um elemento no inicio da lista -----
62  void insereInicio (No **lista, int dado) {
63
64      No *novo;
65
66      /* cria o novo noh a ser inserido na lista */
67      novo = (No*) malloc(sizeof(No));
68      novo->dado = dado;
69
70      /* se a lista estiver vazia, novo noh aponta para NULL */
71      if (listaVazia(*lista) == 1)
72          novo->proximo = NULL;
73      /* caso contrario, novo noh aponta para o inicio da lista */
74      else
75          novo->proximo = *lista;
76
77      /* novo noh passa a ser o inicio da lista */
78      *lista = novo;
79
80  } /* fim da funcao insereInicio */
```

1. Primeiro momento: Revisão

Adicionar nó no final da lista

```
82  /* funcao que insere um elemento no final da lista ----- */
83  void insereFinal (No **lista, int dadoRecebido) {
84      No *novo;
85      No *aux = *lista;
86
87      /* cria o novo noh a ser inserido na lista */
88      novo = (No*) malloc (sizeof(No));
89      novo->dado = dadoRecebido;
90      novo->proximo = NULL;
91
92      /* se a lista estiver vazia, novo noh passa a ser o inicio da lista */
93      if (listaVazia(*lista)) {
94          *lista = novo;
95          return;
96      }
97
98      /* caso contrario, caminha na lista ate parar no ultimo noh */
99      while (aux->proximo != NULL)
100          aux = aux->proximo;
101
102      /* ultimo noh aponta para o novo noh */
103      aux->proximo = novo;
104
105  } /* fim da funcao insereFinal */
```

1. Primeiro momento: Revisão

Remover nó do início da lista

```
153  /* funcao que remove um elemento do inicio da lista -----
154  void removeInicio (No **lista) {
155      No *aux = *lista;
156
157      if (listaVazia(*lista))
158          return;
159
160      /* inicio da lista passa a ser o segundo noh (ou NULL) */
161      *lista = aux->proximo;
162
163      /* remove o primeiro noh da memoria */
164      free(aux);
165
166  } /* fim da funcao removeInicio */
```

1. Primeiro momento: Revisão

Remover nó do final da lista

```
170 void removeFinal (No **lista) {
171     No *aux      = *lista;
172     No *anterior = *lista;
173
174     if (listaVazia(*lista))
175         return;
176
177     /* caminha ate o final da lista */
178     while (aux->proximo != NULL) {
179         anterior = aux;
180         aux = aux->proximo;
181     }
182
183     /* se houver apenas um elemento, entao, inicio torna-se NULL
184     if (aux == *lista)
185         *lista = NULL;
186
187     /* caso contrario, remove o ultimo noh da lista */
188     else
189         anterior->proximo = NULL;
190
191     /* liberar a memória do último noh */
192     free(aux);
193 } /* fim da funcao removeFinal */
```

1. Revisão: Lista encadeada

- Simulador de estruturas lineares:

- Listas

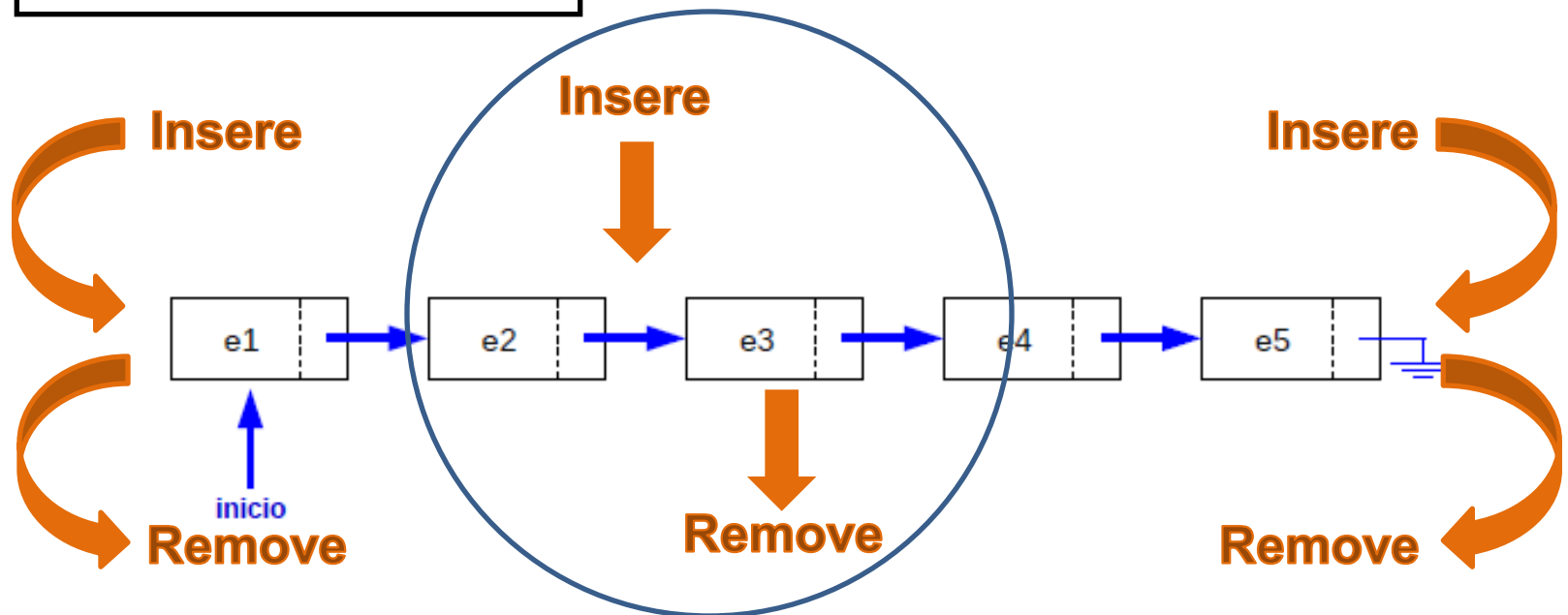
- <https://visualgo.net/pt/list>

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
 - esvaziar a lista

3. Exercícios

**Vamos
Praticar!**



3. 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;
```

3. Exercício

Codificar as seguintes operações

- Pesquisar se um ID está na lista
- Adicionar elemento em uma lista ordenada
- Remover um elemento a partir do seu ID
- Esvaziar a lista

3. Exercício

Inserir em uma lista ordenada

```
void insereOrdenado (No **lista, int id) {  
  
    declarar ponteiro para receber o endereço do novo nó  
    declarar ponteiro auxiliara para andars na lista  
    declarara ponteiro anterior para ajudar no reapontamento  
  
    alocar memória e atribuir valores aos atributos do nó  
  
    testar se a lista está vazia  
        fazer os apontamentos necessários  
        encerrar  
  
    pensando em uma lista ordenada crescente por ID  
    caminhar na lista até achar a posicao correta da inclusão  
  
    se a posicao for o inicio, entao insere no inicio  
  
    caso contrario, se a posicao for o final, entao insere no final  
  
    caso contrario, insere o nó na posicao encontrada  
  
}
```

3. Exercício

Remover um nó a partir do seu ID

```
void removeEspecifico (No **lista, int id){  
    declarar ponteiro aux para andar na lista  
    declarara ponteiro anterior para ajudar no reapontamento  
  
    encerrar se a lista está vazia  
  
    caminhar na lista ate encontrar o nó correspondente ao id  
  
    exibir mensagem de erro se o elemento não for encontreado na lista  
  
    se o elemento estiver no início, remover do início  
  
    caso contrario, se o elemento estiver no final, entao remove do final  
  
    caso contrario, remove o elemento do meio da lista  
  
}
```

3. Exercício

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 */
```

3. Exercício

Esvaziar a lista

```
void esvaziaLista (No **lista) {  
    enquanto a lista nao estiver vazia, entao  
        remove o primeiro noh  
  
}
```

4. Atividade extra

Codificar as operações para uma lista simplesmente encadeada circular

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.