Estrutura de Dados I

Listas Estáticas

Prof. Dr. Marcelo Otone Aguiar

Universidade Federal do Espírito Santo - UFES

10 de Novembro de 2025

Conteúdo

- **Lista estática**
- Lista simplesmente encadeada
- Lista duplamente encadeada
- Lista circular

Listas Lineares

- Estrutura de dados que corresponde a uma sequência ordenada de elementos do mesmo tipo.
- Os elementos de uma lista são chamados de nós (nodos) e podem conter um dado primitivo ou composto.
- Exemplos de listas:
  - Lista de alunos
  - Lista telefônica
  - Palavras de uma frase

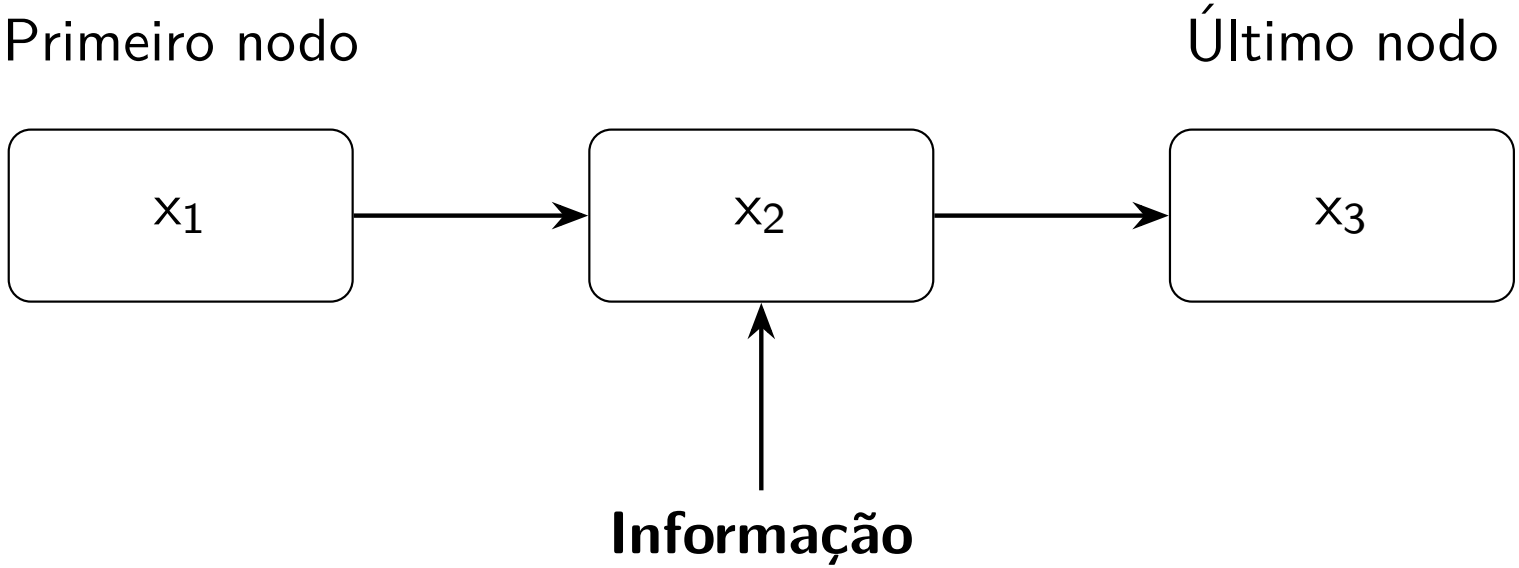
Definição Formal

Uma lista linear é uma coleção de $n \geq 0$ nodos $x_1, x_2, \ldots, x_n$, sendo todos do mesmo tipo, cujas propriedades estruturais relevantes envolvem apenas as posições relativas lineares entre nodos:

- $n = 0$: lista vazia
- $n > 0$: x_1 é o primeiro nodo e x_n o último nodo
- $1 < k < n$: x_k é precedido por x_{k-1} e sucedido por x_{k+1}

O que é o Nodo?

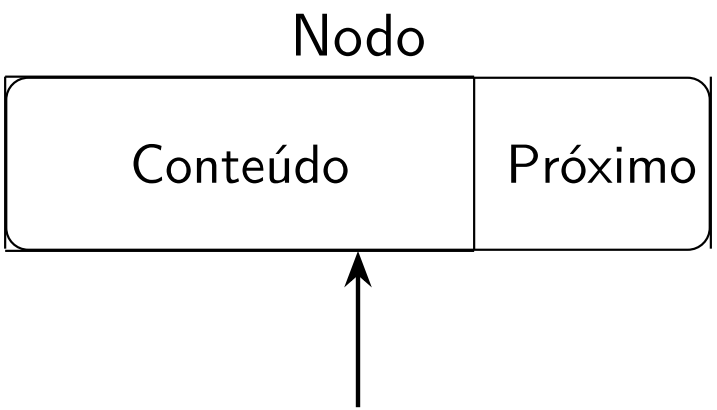
Parte da estrutura de dados responsável por armazenar a informação.



O que é o Nodo?

Estrutura do Nodo

- Estrutura interna é abstraída
- Pode ter uma complexidade arbitrária
- Enfatiza o conjunto de relações existentes



Exemplo de conteúdo:

Matrícula: 101

Nome: Ana

Operações em Listas

Operações básicas:

- Criação da lista vazia
- Inserção de novo elemento
- Remoção de elemento
- Percurso por todos os elementos
- Busca (consultar/alterar)
- Destruição da lista

Outras Operações

- Emendar duas listas
- Partir listas
- Copiar lista
- Determinar número de nodos
- Pesquisar valor
- Ordenar elementos

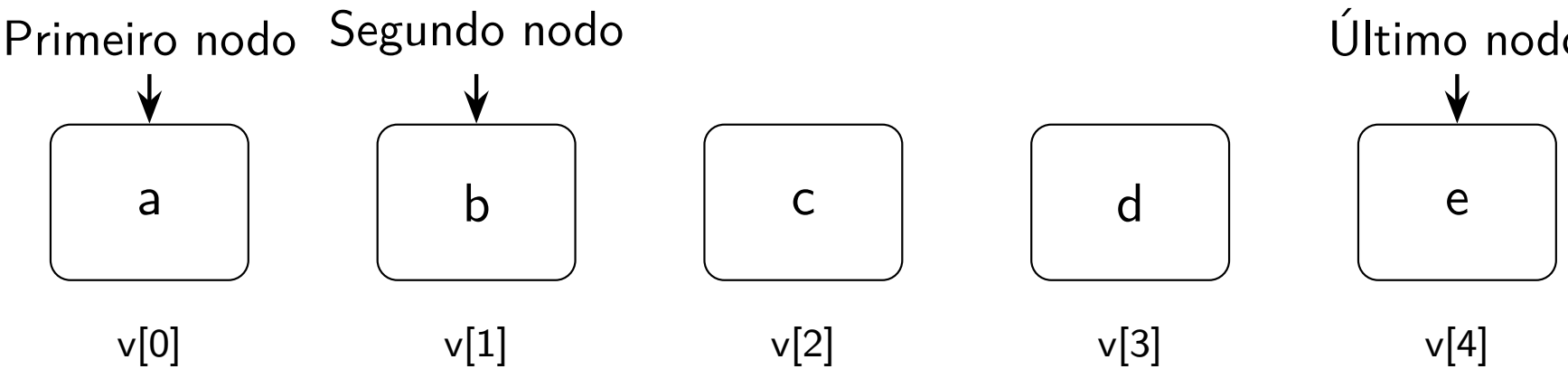
Representação de Listas

Como armazenar elementos da lista dentro do computador?

- Alocação Estática
- Alocação Dinâmica

Alocação Estática

- O espaço de memória é alocado no momento da compilação.
- Exige a definição do número máximo de elementos da **lista**.
- **Acesso sequencial:** elementos consecutivos na memória.



Alocação Estática

- **Vantagens:**
  - Dado o endereço inicial de cada área alocada e o índice i de um elemento qualquer da lista, podemos acessá-lo diretamente.
- **Desvantagens:**
  - Tamanho máximo pré-estimado;
  - Inexistência de grandes espaços contíguos na memória;
  - Maior esforço computacional para operações de inserção e remoção de nós.

Alocação Estática

Quando usar:

- Listas pequenas;
- Inserção / Remoção no fim da lista;
- Tamanho máximo bem definido;
- A busca é a operação mais frequente.

Alocação Estática

- A inserção de um novo item pode ser realizada após o último item com custo constante.
- A remoção de um item no meio da lista requer um deslocamento de itens para preencher o espaço deixado vazio.
- Os itens são armazenados em um vetor de tamanho suficiente para armazenar a lista.
- Último armazena a posição do próximo a ser inserido na lista.
- O *i-ésimo* item da lista está armazenado na *i-ésima* posição do vetor – 1.
- **MAX** define o tamanho máximo permitido para a lista.

Implementação

- **ListaSequencial.h:** definir
  - Os protótipos das funções
  - O tipo de dado armazenado na lista
  - O ponteiro **lista**
  - Tamanho do vetor usado na lista
- **ListaSequencial.c:** definir
  - O tipo de dados **lista**
  - Implementar as suas funções.

Código em C - Estrutura da Lista

```
1  //Arquivo ListaSequencial.h
2  #define MAX 100
3  struct aluno {
4    int matricula;
5    char nome[30];
6    float n1, n2, n3;
7  };
8  typedef struct lista Lista;
```

Código em C - Estrutura da Lista

```c
//Arquivo ListaSequencial.c
struct lista {
  int qtd;
  struct aluno dados[MAX];
};
```

```c
//Programa principal
Lista *li;
```

Código em C - Criação da Lista

```c
//Arquivo ListaSequencial.c
Lista* cria_lista(){
    Lista *li;
    li = (Lista*) malloc(sizeof(struct lista));

    if (li != NULL)
        li->qtd = 0;

    return li;
}
```

```c
//Programa principal
li = cria_lista();
```

Código em C - Liberação da Lista

```c
//Arquivo ListaSequencial.c
void libera_lista(Lista* li){
   free(li);
}
```

```c
//Programa principal
libera_lista(li);
```

Código em C - Tamanho da Lista

```c
//Arquivo ListaSequencial.c
int tamanho_lista(Lista* li){
  if (li == NULL)
      return -1;
  else
      return li->qtd;
}
```

```c
//Programa principal
int x = tamanho_lista(li);
```

Código em C - Verifica se a Lista está cheia

```c
//Arquivo ListaSequencial.c
int lista_cheia(Lista* li){
  if (li == NULL)
       return -1;
  return (li->qtd == MAX);
}
```

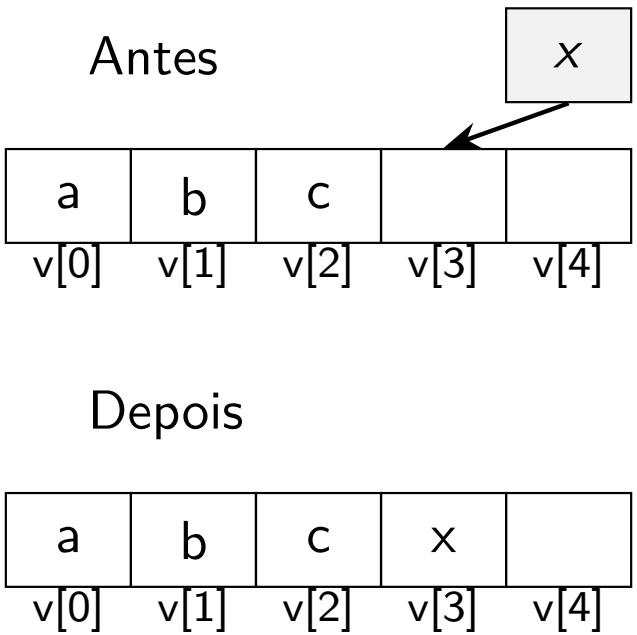
```c
//Programa principal
int x = lista_cheia(li);
if (lista_cheia(li))
```

Código em C - Verifica se a Lista está vazia

```c
//Arquivo ListaSequencial.c
int lista_vazia(Lista* li){
  if (li == NULL)
       return -1;
  return (li->qtd == 0);
}
```

```c
//Programa principal
int x = lista_vazia(li);
if (lista_vazia (li))
```

Inserção no final

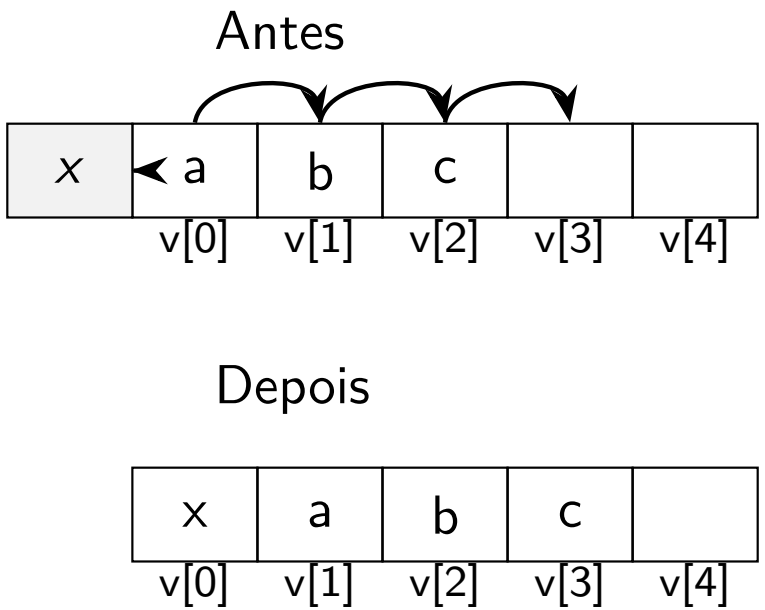


Código em C - Inserir no final

```c
//Arquivo ListaSequencial.c
int insere_lista_final(Lista* li, struct aluno al){
  if (li == NULL)
      return 0;
  if (lista_cheia(li))
      return 0;
  li->dados[li->qtd] = al;
  li->qtd++;
  return 1;
}
```

```c
//Programa principal
int x = insere_lista_final(li, dados_aluno);
```

Inserção no início

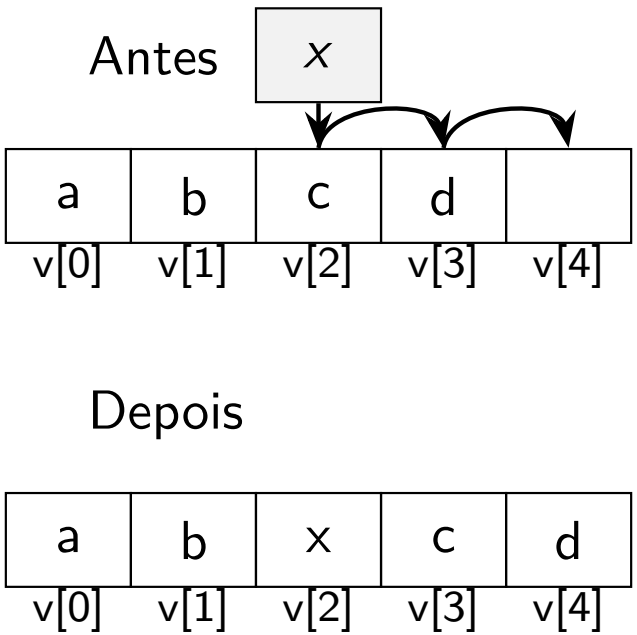


Código em C - Inserir no início

```c
//Arquivo ListaSequencial.c
int insere_lista_inicio(Lista* li, struct aluno al){
  if (li == NULL)
      return 0;
  if (lista_cheia(li))
      return 0;
  int i;
  for (i=li->qtd-1; i>=0; i--)
      li->dados[i+1] = li->dados[i];
  li->dados[0] = al;
  li->qtd++;
  return 1;
}
```

```c
//Programa principal
int x = insere_lista_inicio(li, dados_aluno);
```

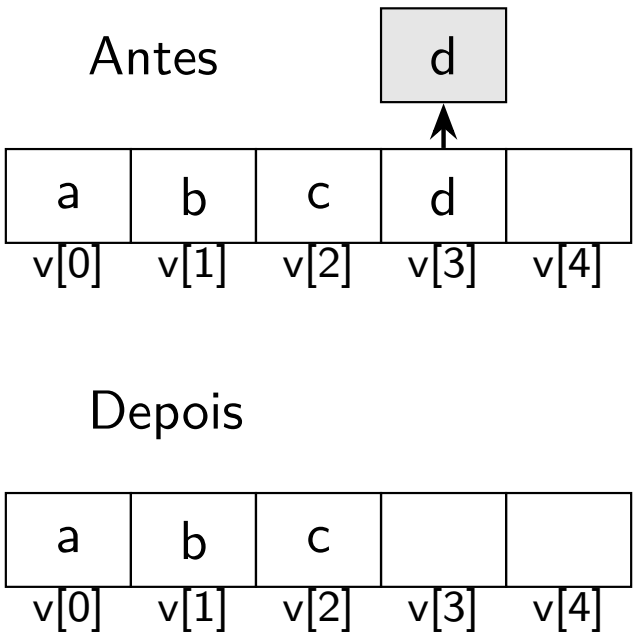
Inserção no meio



Código em C - Inserir no meio

```
1  //Arquivo ListaSequencial.c
2  int insere_lista_ordenada(Lista* li, struct aluno al){
3    if (li == NULL) return 0;
4    if (lista_cheia(li)) return 0;
5    int k, i = 0;
6    while (i<li->qtd && li->dados[i].matricula < al.
         matricula)
7        i++;
8    for (k=li->qtd-1; k >= i; k--)
9        li->dados[k+1] = li->dados[k];
10   li->dados[i] = al;
11   li->qtd++;
12   return 1;
13 }
```

Remoção no final

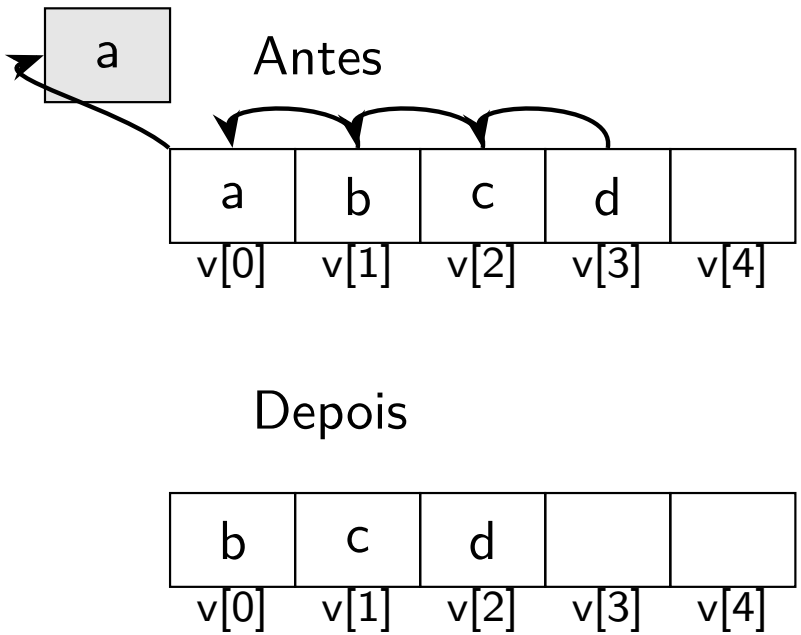


Código em C - Remover no final

```c
//Arquivo ListaSequencial.c
int remove_lista_final(Lista* li){
  if (li == NULL)
      return 0;
  if (li->qtd == 0)
      return 0;
  li->qtd--;
  return 1;
}
```

```c
//Programa principal
int x = remove_lista_final(li);
```

Remoção no início

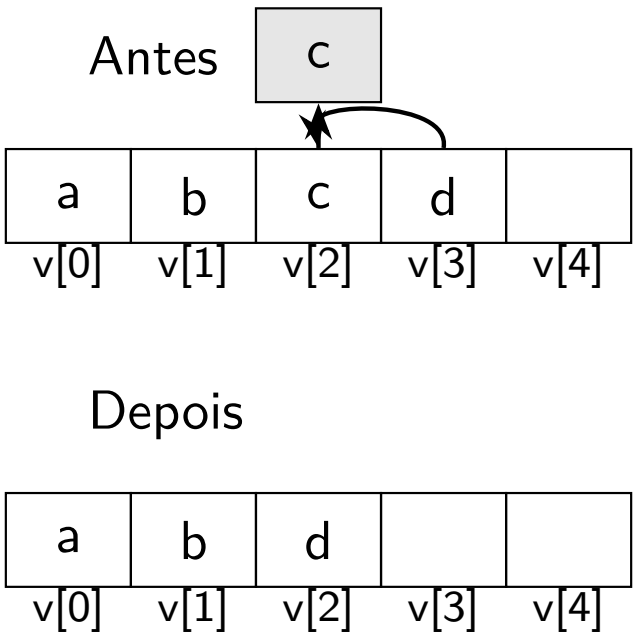


Código em C - Remover no início

```c
//Arquivo ListaSequencial.c
int remove_lista_inicio(Lista* li){
  if (li == NULL)
      return 0;
  if (li->qtd == 0)
      return 0;
  int k = 0;
  for (k=0; k < li->qtd-1; k++)
      li->dados[k] = li->dados[k+1];
  li->qtd--;
  return 1;
}
```

```c
//Programa principal
int x = remove_lista_inicio(li);
```

Remoção no meio



Código em C - Remover no meio

```
1  //Arquivo ListaSequencial.c
2  int remove_lista(Lista* li, int mat){
3    if (li == NULL) return 0;
4    if (li->qtd == 0) return 0;
5    int k, i = 0;
6    while (i<li->qtd && li->dados[i].matricula != mat)
7         i++;
8    if (i == li->qtd) //elemento nao encontrado
9         return 0;
10
11   for (k=i; k < li->qtd-1; k++)
12        li->dados[k] = li->dados[k+1];
13   li->qtd--;
14   return 1;
15 }
```

Consulta

- Existem 2 maneiras de consultar um elemento de uma lista:
  - Pela **posição** (acesso direto)
  - Pelo **conteúdo** (necessidade de busca)

Código em C - Consulta pela posição

```c
//Arquivo ListaSequencial.c
int consulta_lista_pos(Lista* li, int pos, struct aluno *al){
  if (li == NULL  pos <= 0  pos > li->qtd)
      return 0;

  *al = li->dados[pos-1];
  return 1;
}
```

```c
//Programa principal
int x = consulta_lista_pos(li, posicao, &dados_aluno);
```

Código em C - Consulta pelo conteúdo

```c
//Arquivo ListaSequencial.c
int consulta_lista_mat(Lista* li, int mat, struct aluno *al){
  if (li == NULL)
      return 0;
  int k, i = 0;
  while (i<li->qtd && li->dados[i].matricula != mat)
      i++;
  if (i == li->qtd) //elemento nao encontrado
      return 0;
  *al = li->dados[i];
  return 1;
}
```

```c
//Programa principal
x = consulta_lista_mat(li, matricula, &dados_aluno);
```