Marisangila Alves, MSc
marisangila.alves@catolicasc.org.br
marisangila.com.br

Católica de Santa Catarina

2025/2

Estrutura de Dados

Estrutura de Dados
Lista

Sumário

Definição

Lista

> *Uma lista é uma coleção de elementos do mesmo tipo dispostos linearmente que podem ou não seguir determinada organização*
>
> (PUGA; RISSETTI, 2009)

Ao remover um item da lista, ele é imediatamente destruído.

Exemplos I

- **Lista de pagamentos;**
- **Lista de chamada de alunos;**
- **Lista de compras;**
- **Lista de contatos telefônicos.**

Implementações

- Pode ser implementada com vetor ou com elementos encadeados (alocados dinamicamente).
- A implementação com vetor é mais simples e intuitiva, porém possui desvantagens:
  - Tamanho máximo fixo — é necessário testar se a lista está cheia.
  - Mesmo vazia, ocupa todo o espaço previamente alocado.
  - Operações no meio da lista exigem deslocamento de dados, resultando em baixo desempenho.

Operações

O funcionamento de uma lista se resume em nas seguintes funções ou operações:

- As principais funções utilizadas para manipular uma lista são:
  - As principais operações sobre listas incluem inicialização, manipulação, testes, visualização e desalocação.
  - A função `inicializar` cria e retorna uma nova lista vazia.
  - A função `desalocar` libera toda a memória utilizada pela lista.
  - Testes comuns envolvem:
    - Verificar se a lista está vazia.
    - Localizar a posição de uma informação na lista.
  - A lista pode ser manipulada de diferentes formas:
    - Inserção no início, no fim, em uma posição específica ou em ordem.
    - Remoção de elementos em qualquer uma dessas posições.
    - Modificação do valor de um elemento existente.
- A função `mostrar` exibe todos os dados da lista na tela.
- Também é possível implementar funções de `busca` e `ordenação` dos elementos da lista.

- `inicializa_lista(p)` — Cria uma lista vazia.
- `lista_vazia(p)` — Verifica se a lista `p` está vazia.
- `lista_cheia(p)` — Verifica se a lista `p` está cheia.
- `acessa_topo(p)` — Retorna o valor no topo da lista `p`, sem removê-lo (somente leitura).
- `mostra_lista(p)` — Mostra na tela os elementos contidos na lista `p` (função auxiliar).
- `desaloca_lista(p)` — Libera o espaço de memória ocupado pela lista `p`.

- Por não fazer alocação dinâmica, esta versão da lista não necessita de uma função para desalocação.
- Não é possível modificar a capacidade da lista em tempo de execução.
- É necessário saber antecipadamente o número máximo de valores que precisam estar armazenados simultaneamente na lista.

Lista Encadeada

Lista Encadeada

> *Uma lista encadeada (ou lista linear) é um conjunto linear de estruturas autoreferenciadas, chamadas nós, conectadas por links (ligações ou encadeamento)deponteiros daí o termo lista "encadeada"("linked"list, em inglês). Uma lista encadeada é acessada por meio de um ponteiro para o primeiro nó da lista. Os nós subseqüentes são acessados por meio do membro ponteiro de ligação (link) armazenado em cada nó*
>
> (Deitel; Deitel, 2011)

Os dados são armazenados dinamicamente em uma lista encadeada cada nó é criado quando necessário. Um nó pode conter dados de qualquer tipo, incluindooutras structs.

- Os elementos de uma lista encadeada devem ser acessados sequencialmente, a partir do início da lista.
- Essa é a única forma de armazenar e recuperar dados em uma lista encadeada.
- Não é permitido acesso randômico direto a um item específico da lista.

- Listas encadeadas são recomendadas quando não se sabe previamente a quantidade de dados.
- São estruturas dinâmicas: seu tamanho pode crescer ou diminuir conforme necessário.
- Vetores (*Arrays*) têm tamanho fixo, pois sua memória é alocada em tempo de compilação.
- Vetores (*Arrays*) podem ficar cheios; listas encadeadas só ficam cheias se faltar memória no sistema.
- Permitem inserção ordenada, mantendo a lista em ordem ao adicionar elementos.
- Nós de listas encadeadas não são armazenados de forma contígua na memória física.
- Apesar disso, logicamente os nós parecem estar em sequência.

O que é?

Uma estrutura autorreferenciada é uma estrutura que possui um ou mais **ponteiros que apontam para ela mesma**.

Isso permite representar conjuntos encadeados de elementos, como:

- Listas encadeadas
- Pilhas e filas dinâmicas
- Árvores e grafos

```c
struct no
{
    int dados;
    struct no *proximo;
};
```

Código 1: Exemplo de estrutura autoreferenciada.

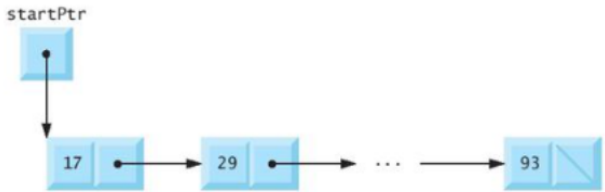

Figura 2: Estrutura de uma lista encadeada (Deitel; Deitel, 2011).

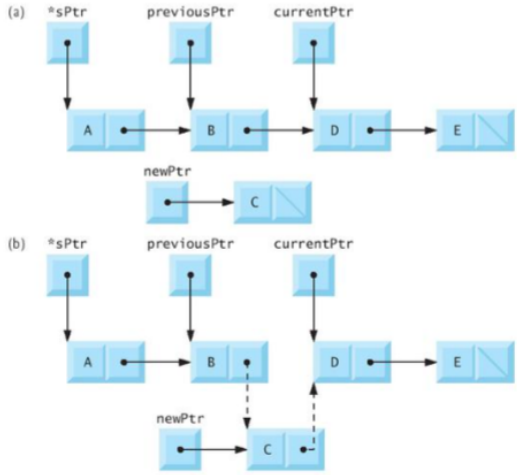

Figura 3: Operação de inserção de um item da lista (Deitel; Deitel, 2011).

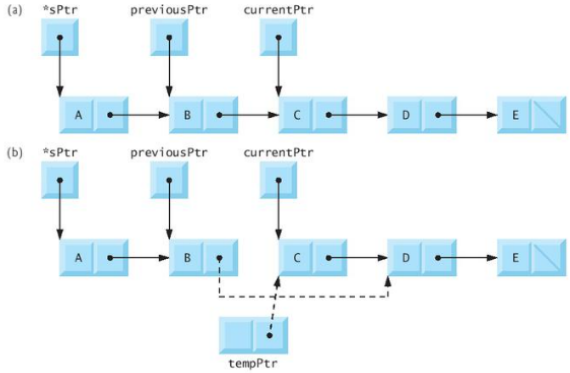

Figura 4: Operação de exclusão de um item da lista (Deitel; Deitel, 2011).

```c
typedef struct No
{
    int dado;
    struct No *proximo;
}No;
typedef struct
{
    No *cabeca;
}Lista;
```

Código 2: Estrutura da lista.

```h
#ifndef LISTA_H
#define LISTA_H
Lista *inicializar();
void desalocar(Lista *lista);
int vazia(Lista *lista);
int mostrar(Lista *lista);
int buscar(int dado, Lista *lista);
int buscar_editar(int dado, int novo_dado, Lista *lista);
int buscar_remover(int dado, Lista *lista);
int inserir_inicio(int dado, Lista *lista);
int editar_inicio(int dado, int novo_dado, Lista *lista);
int remover_inicio(Lista *lista);
int inserir_fim(int dado, Lista *lista);
int editar_fim(int dado, Lista *lista);
int remover_fim(Lista *lista);
int inserir_ordenado(int dado, Lista *lista);
#endif
```

Código 3: Exemplo de protótipos das funções.

```c
Lista *inicializar()
{
    Lista *lista = malloc(sizeof(Lista));
    lista->cabeca = NULL;
    lista->quantidade_nos = 0;
    lista->tamanho_dado = 0;
    return lista;
}
```

Código 4: Exemplo de inicialização.

```c
int mostrar(Lista *lista)
{
    if (vazia(lista))
        return LISTA_VAZIA;
    No *atual = lista->cabeca;
    while (atual != NULL)
    {
        printf(" [ %d ] \n", atual->dado);
        atual = atual->proximo;
    }
    return SUCESSO;
}
```

Código 5: Exemplo de visualização.

```c
int vazia(Lista *lista)
{
    return lista->cabeca == NULL;
}
```

Código 6: Exemplo de verificação de lista vazia.

```c
void desalocar(Lista *lista)
{
    No *atual = lista->cabeca;
    while (atual != NULL)
    {
        No *temp = atual;
        atual = atual->proximo;
        free(temp);
    }
    free(lista);
}
```

Código 7: Exemplo de verificação de lista cheia.

```c
int inserir_inicio(int dado, Lista *lista)
{
    No *novo = malloc(sizeof(No));
    if (novo == NULL)
        return ERRO;
    novo->dado = dado;
    novo->proximo = lista->cabeca;
    lista->cabeca = novo;
    lista->quantidade_nos++;
    return SUCESSO;
}
```

Código 8: Exemplo de inserção no início.

```c
int remover_inicio(Lista *lista)
{
    if (vazia(lista))
        return LISTA_VAZIA;
    No *no = lista->cabeca;
    lista->cabeca = no->proximo;
    free(no);
    lista->quantidade_nos--;
    return SUCESSO;
}
```

Código 9: Exemplo de remoção no início.

```c
int buscar_editar(int dado, int novo_dado, Lista *lista)
{
    if (vazia(lista))
        return LISTA_VAZIA;
    No *atual = lista->cabeca;
    while (atual != NULL)
    {
        if (atual->dado == dado)
        {
            atual->dado = novo_dado;
            return SUCESSO;
        }
        atual = atual->proximo;
    }
    return ITEM_NAO_ENCONTRADO;
}
```

Código 10: Exemplo de busca e edição.

```c
#include "lista.h"
#include <stdio.h>
int main(void)
{
    Lista *lista = inicializar();
    printf("Inserindo no inicio 20!\n");
    inserir_inicio(20, lista);
    mostrar(lista);
    printf("Inserindo no inicio 10!\n");
    inserir_inicio(10, lista);
    mostrar(lista);
    printf("Inserindo no inicio 50!\n");
    inserir_inicio(50, lista);
    mostrar(lista);
    printf("Removendo inicio!\n");
    remover_inicio(lista);
    mostrar(lista);
    printf("Inserindo no inicio 5!\n");
    inserir_inicio(5, lista);
    mostrar(lista);
    return 0;
}
```

Código 11: Exemplo de utilização da lista.

Lista Duplamente Encadeada

- Listas encadeadas possuem a desvantagem de permitir o percurso apenas em uma direção.
- Para acessar elementos já visitados, é necessário um esforço adicional, o que impacta o desempenho:
  - Utilização de uma estrutura auxiliar, como uma pilha.
  - Uso de processo recursivo.
  - Reinício do percurso desde o início da lista.

```c
typedef struct No
{
    int dado;
    struct No *proximo, *anterior;

}No;
typedef struct
{
    No *cabeca, *calda;

}Lista;
```

Código 12: Exemplo de estrutura.

(Deitel; Deitel, 2011) - Capítulo 12

DEITEL, Paul; DEITEL, Harvey. **C: Como Programar**. 6. ed. São Paulo: Pearson Universidades, 2011.

PUGA, Sandra Gavioli; RISSETTI, Gerson. **Lógica de programação e estrutura de dados: com aplicações em Java**. 2. ed. São Paulo, SP: Pearson, 2009. E-book. Disponível em: `https://plataforma.bvirtual.com.br`. Acesso em: 24 jul. 2025.

SCHILDT, Herbert. **C Completo e Total: O Guia Definitivo para Programação em C**. 3. ed. São Paulo: Makron Books, 1996. ISBN 978-8534606928.

Estes slides estão protegidos por uma licença Creative Commons

Este modelo foi adaptado de Maxime Chupin.

Marisangila Alves, MSc
marisangila.alves@catolicasc.org.br
marisangila.com.br

Católica de Santa Catarina

2025/2

Estrutura de Dados

Estrutura de Dados
Lista