Estrutura de Dados para Armazenamento de Informações

Listas Encadeadas em Linguagem C

Conceitos Iniciais: Listas Encadeadas

- Porque usar Lista Encadeada?

VETOR (Lista Linear)	LISTA ENCADEADA
Tamanho fixo	Tamanho variável
Alocação contínua de memória	Alocação dinâmica (pedaços separados)
Inserções/remoções no meio são custosas	Inserções/remoções são mais eficientes

➡️ Uma **Lista Encadeada** é uma **estrutura de dados dinâmica** que armazena uma sequência de elementos (**nós**), onde cada nó aponta para o próximo.

•Diferente de **vetores** ou **matrizes**, a Lista Encadeada **não precisa de um tamanho fixo**.

•Cada elemento é ligado ao próximo através de **ponteiros**.

Estrutura de um **Nó** (Elemento da Lista)

Em C, cada **nó** da lista é representado por uma **struct** que possui:
1. **Dado**: valor que queremos armazenar.
2. **Ponteiro para o próximo nó**.

Exemplo:

```c
struct No {
    int valor;          // Dado do nó
    struct No *prox;    // Ponteiro para o próximo nó
};
```

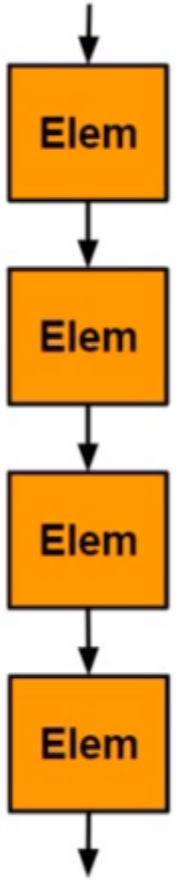
✅ **Como funciona visualmente?**
🔗 Exemplo de uma Lista Encadeada com 3 nós:

```
[10 | * ] → [20 | * ] → [30 | NULL]
```

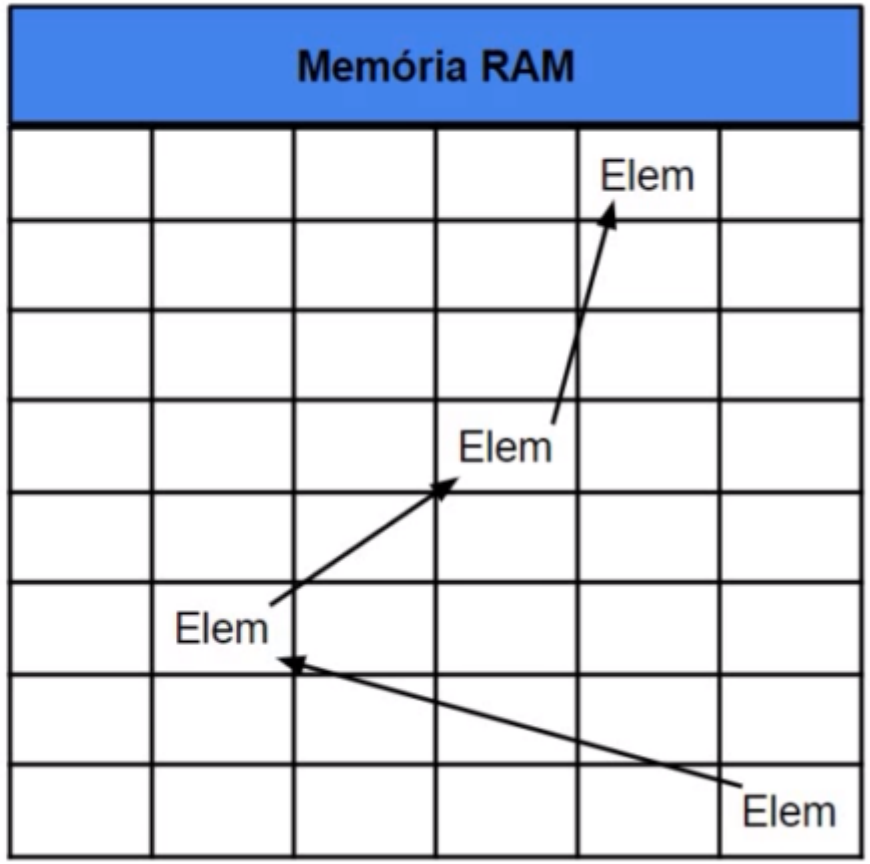
O ponteiro do último nó aponta para NULL, indicando o **fim da lista**.

✅ **Vantagens:**
✅ Cresce conforme a necessidade (tamanho dinâmico).
✅ Remover e inserir elementos não exige realocação geral.
✅ Ótimo para aplicações com inserções/remover frequentes.
❌ **Desvantagens:**
❌ Acesso lento a elementos (precisa percorrer um por um).
❌ Maior consumo de memória (devido aos ponteiros).

Lista Encadeada



Lógica



Fisicamente

Principais Operações em uma Lista Encadeada

Operação	O que faz
Inserir no início	Adiciona um nó no começo da lista
Inserir no fim	Adiciona um nó no final da lista
Remover um elemento	Remove um nó da lista
Percorrer a lista	Visitar cada elemento da lista
Buscar um elemento	Verifica se um elemento existe

Lista Encadeada: Operação: Inserir no Início (Lista Encadeada Simples)

Adicionar um novo nó **no começo da lista**.

- Ex: antes → [10 | *] → [20 | *] → [30 | NULL]
- Inserir 5 no início → [5 | *] → [10 | *] → [20 | *] → [30 | NULL]

Passos Lógicos:

1. Criar um novo nó.
2. Colocar o valor dentro do novo nó.
3. Fazer o ponteiro do novo nó apontar para o antigo primeiro nó.
4. Atualizar o "início" da lista (head) para ser o novo nó.

Implementando

```c
#include <stdio.h>
#include <stdlib.h>
// Definindo a estrutura do nó da lista
struct No {
    int valor;            // Valor armazenado no nó
    struct No *prox;      // Ponteiro para o próximo nó
};
// Função para inserir um novo nó no início da lista
struct No* inserirNoInicio(struct No *head, int novoValor) {
    struct No *novoNo = (struct No*)malloc(sizeof(struct No)); // Aloca memória para o novo nó
    novoNo->valor = novoValor;     // Atribui o valor desejado ao novo nó
    novoNo->prox = head;           // Faz o ponteiro 'prox' do novo nó apontar para o antigo primeiro nó (head)
    return novoNo;                 // Retorna o novo nó como sendo o novo início (head) da lista
}
// Função para imprimir a lista encadeada
void imprimirLista(struct No *head) {
    struct No *atual = head;       // Cria um ponteiro auxiliar para percorrer a lista
    printf("Lista: ");
    while (atual != NULL) {        // Enquanto não chegar no final (NULL)
        printf("%d -> ", atual->valor); // Imprime o valor do nó atual
        atual = atual->prox;       // Avança para o próximo nó
    }
    printf("NULL\n");              // Indica o fim da lista
}
int main() {
    struct No *lista = NULL;       // Inicialmente a lista está vazia (NULL)
    // Vamos construir a lista inicial [10 | *] → [20 | *] → [30 | NULL]
    lista = inserirNoInicio(lista, 30); // Insere 30 no início → lista = [30 | NULL]
    lista = inserirNoInicio(lista, 20); // Insere 20 no início → lista = [20 | *] → [30 | NULL]
    lista = inserirNoInicio(lista, 10); // Insere 10 no início → lista = [10 | *] → [20 | *] → [30 | NULL]
    // Agora vamos inserir o valor 5 no início → queremos [5 | *] → [10 | *] → [20 | *] → [30 | NULL]
    lista = inserirNoInicio(lista, 5);
    // Imprimir a lista final
    imprimirLista(lista);
    return 0;
}
```

Lista Encadeada: Operação: Inserir no Fim (Lista Encadeada Simples)

Adicionar um novo nó **no final da lista**.
Exemplo:
Antes:
[10 | *] → [20 | *] → [30 | NULL]
Inserir 40 no fim:
[10 | *] → [20 | *] → [30 | *] → [40 | NULL]

Passos Lógicos:
1. Criar um novo nó.
2. Colocar o valor no novo nó.
3. Fazer o ponteiro prox do novo nó apontar para NULL.
4. Percorrer a lista até o último nó.
5. Fazer o ponteiro prox do último nó apontar para o novo nó.

Implementando

```c
#include <stdio.h>
#include <stdlib.h>
// Definindo a estrutura do nó
struct No {
    int valor;
    struct No *prox;
};
// Função para inserir no fim da lista
struct No* inserirNoFim(struct No *head, int novoValor) {
    // 1. Alocar memória para o novo nó
    struct No *novoNo = (struct No*)malloc(sizeof(struct No));
    // 2. Atribuir o valor ao novo nó
    novoNo->valor = novoValor;
    // 3. O novo nó será o último, portanto 'prox' será NULL
    novoNo->prox = NULL;
    // 4. Se a lista estiver vazia (head == NULL), o novo nó será o primeiro nó
    if (head == NULL) {
        return novoNo;
    }
    // 5. Percorrer a lista até o último nó
    struct No *atual = head;
    while (atual->prox != NULL) {
        atual = atual->prox;
    }
    // 6. Fazer o último nó apontar para o novo nó
    atual->prox = novoNo;
    // 7. Retorna o início da lista (head não muda)
    return head;
}
// Função para imprimir a lista encadeada
void imprimirLista(struct No *head) {
    struct No *atual = head;
    printf("Lista: ");
    while (atual != NULL) {
        printf("%d -> ", atual->valor);
        atual = atual->prox;
    }
    printf("NULL\n");
}
```

```c
int main() {
    struct No *lista = NULL;  // Lista vazia inicialmente

    // Construindo a lista inserindo no fim
    lista = inserirNoFim(lista, 10);  // lista = [10 | NULL]
    lista = inserirNoFim(lista, 20);  // lista = [10 | *] → [20 | NULL]
    lista = inserirNoFim(lista, 30);  // lista = [10 | *] → [20 | *] → [30 | NULL]

    // Inserindo 40 no fim
    lista = inserirNoFim(lista, 40);  // lista = [10 | *] → [20 | *] → [30 | *] → [40 | NULL

    // Imprimindo a lista final
    imprimirLista(lista);

    return 0;
}
```

Lista Encadeada: Operação: Remover um Elemento

Remover um nó que **contém um valor específico** da lista.

Exemplo:

Lista antes:

[10 | *] → [20 | *] → [30 | *] → [40 | NULL]

Remover o valor **30**:

[10 | *] → [20 | *] → [40 | NULL]

Passos Lógicos:

1. Verificar se a lista está vazia.
2. Verificar se o valor a ser removido está **no primeiro nó** (head).
3. Se não estiver no primeiro:
   - Percorrer a lista com dois ponteiros: um para o **nó atual** e outro para o **anterior**.
   - Procurar o nó com o valor desejado.
4. Se encontrar:
   - Ajustar os ponteiros para **pular esse nó**.
   - Liberar a memória com free.
5. Se não encontrar: informar que o valor não está na lista.

```c
#include <stdio.h>
#include <stdlib.h>
// Estrutura do nó
struct No {
    int valor;
    struct No *prox;
};
// Função para remover um nó com valor específico
struct No* removerElemento(struct No *head, int valorRemover) {
    // 1. Verifica se a lista está vazia
    if (head == NULL) {
        printf("Lista vazia.\n");
        return NULL;
    }
    // 2. Verifica se o valor está no primeiro nó
    if (head->valor == valorRemover) {
        struct No *temp = head;          // Guarda o nó a ser removido
        head = head->prox;               // Atualiza o head para o próximo nó
        free(temp);                      // Libera o nó removido
        return head;                     // Retorna o novo head
    }
    // 3. Caso contrário, percorre a lista procurando o valor
    struct No *anterior = head;
    struct No *atual = head->prox;
    while (atual != NULL && atual->valor != valorRemover) {
        anterior = atual;
        atual = atual->prox;
    }
    // 4. Se não encontrou o valor
    if (atual == NULL) {
        printf("Valor %d não encontrado na lista.\n", valorRemover);
        return head;
    }
    // 5. Encontrou o valor: remove o nó
    anterior->prox = atual->prox; // Pula o nó a ser removido
    free(atual);                  // Libera a memória do nó
    return head;                  // Retorna o head da lista
}
```

```c
// Função para imprimir a lista
void imprimirLista(struct No *head) {
    struct No *atual = head;
    printf("Lista: ");
    while (atual != NULL) {
        printf("%d -> ", atual->valor);
        atual = atual->prox;
    }
    printf("NULL\n");
}
// Função auxiliar para inserir no fim (reaproveitando o que já vimos)
struct No* inserirNoFim(struct No *head, int valor) {
    struct No *novo = (struct No*)malloc(sizeof(struct No));
    novo->valor = valor;
    novo->prox = NULL;
    if (head == NULL) return novo;
    struct No *atual = head;
    while (atual->prox != NULL) {
        atual = atual->prox;
    }
    atual->prox = novo;
    return head;
}
int main() {
    struct No *lista = NULL;
    // Criando a lista [10 -> 20 -> 30 -> 40]
    lista = inserirNoFim(lista, 10);
    lista = inserirNoFim(lista, 20);
    lista = inserirNoFim(lista, 30);
    lista = inserirNoFim(lista, 40);
    imprimirLista(lista); // Mostra a lista inicial
    // Remover o valor 30
    lista = removerElemento(lista, 30);
    imprimirLista(lista); // Esperado: 10 -> 20 -> 40 -> NULL
    // Tentar remover valor inexistente
    lista = removerElemento(lista, 99);
    // Remover o primeiro elemento (10)
    lista = removerElemento(lista, 10);
    imprimirLista(lista); // Esperado: 20 -> 40 -> NULL
    return 0;
}
```

Lista Encadeada: Operação: Percorrer a Lista

Percorrer a lista significa **visitar cada elemento (nó)**, do primeiro até o último, **em sequência**.

- É a operação mais básica:

- Serve para **imprimir valores**.
- Pode ser usada para **buscar**.
- Também é base para outras operações (contar, somar, etc).

Passos Lógicos:

1. Se a lista estiver vazia, não faz nada.
2. Cria um **ponteiro auxiliar** para percorrer a lista.
3. Começa no **head**.
4. Enquanto não chegar no fim (NULL):
    - Acessa o valor do nó atual.
    - Avança para o próximo nó.

Implementando

```c
#include <stdio.h>
#include <stdlib.h>
// Estrutura do nó
struct No {
    int valor;
    struct No *prox;
};
// Função para percorrer e imprimir a lista
void imprimirLista(struct No *head) {
    struct No *atual = head;  // Cria um ponteiro auxiliar que começa no head
    printf("Lista: ");
    while (atual != NULL) {  // Enquanto o ponteiro não for NULL (fim da lista)
        printf("%d -> ", atual->valor);  // Imprime o valor do nó atual
        atual = atual->prox;  // Avança para o próximo nó
    }
    printf("NULL\n");  // Indica o fim da lista
}
// Função auxiliar para inserir no fim (já conhecida)
struct No* inserirNoFim(struct No *head, int valor) {
    struct No *novo = (struct No*)malloc(sizeof(struct No));
    novo->valor = valor;
    novo->prox = NULL;
    if (head == NULL) return novo;
    struct No *atual = head;
    while (atual->prox != NULL) {
        atual = atual->prox;
    }
    atual->prox = novo;
    return head;
}
int main() {
    struct No *lista = NULL;
    // Criando a lista: 10 -> 20 -> 30 -> NULL
    lista = inserirNoFim(lista, 10);
    lista = inserirNoFim(lista, 20);
    lista = inserirNoFim(lista, 30);
    // Percorrendo e imprimindo a lista
    imprimirLista(lista);
    return 0;
}
```

Lista Encadeada: Operação: Buscar um Elemento

É **percorrer a lista** procurando um **valor específico**.

- Se encontrar → avisa que achou (e pode até retornar o endereço).
- Se não encontrar → avisa que não existe na lista.

Passos Lógicos:

1. Verificar se a lista está vazia.
2. Criar um ponteiro auxiliar para percorrer.
3. Enquanto não chegar no fim:
    1. Comparar o valor do nó atual.
    2. Se achar → retorna (ou imprime).
    3. Se não → avança para o próximo nó.
4. Se chegar no fim sem encontrar → avisa que não existe.

Implementando

```c
#include <stdio.h>
#include <stdlib.h>
// Estrutura do nó
struct No {
    int valor;
    struct No *prox;
};
// Função para buscar um valor na lista
struct No* buscarElemento(struct No *head, int valorBusca) {
    struct No *atual = head;  // Ponteiro auxiliar para percorrer
    while (atual != NULL) {   // Percorre até o fim da lista
        if (atual->valor == valorBusca) {  // Se encontrou o valor
            return atual;  // Retorna o endereço do nó encontrado
        }
        atual = atual->prox;  // Avança para o próximo nó
    }
    return NULL;  // Se não encontrou, retorna NULL
}
// Função para imprimir a lista
void imprimirLista(struct No *head) {
    struct No *atual = head;
    printf("Lista: ");
    while (atual != NULL) {
        printf("%d -> ", atual->valor);
        atual = atual->prox;
    }
    printf("NULL\n");
}
// Função auxiliar para inserir no fim (reaproveitando)
struct No* inserirNoFim(struct No *head, int valor) {
    struct No *novo = (struct No*)malloc(sizeof(struct No));
    novo->valor = valor;
    novo->prox = NULL;
    if (head == NULL) return novo;
    struct No *atual = head;
    while (atual->prox != NULL) {
        atual = atual->prox;
    }
    atual->prox = novo;
    return head;
}
```

```c
int main() {
    struct No *lista = NULL;

    // Criando a lista: 10 -> 20 -> 30 -> NULL
    lista = inserirNoFim(lista, 10);
    lista = inserirNoFim(lista, 20);
    lista = inserirNoFim(lista, 30);

    imprimirLista(lista);

    // Buscar valor 20
    struct No *resultado = buscarElemento(lista, 20);
    if (resultado != NULL) {
        printf("Valor %d encontrado no endereço %p.\n", resultado->valor, (void*)resultado);
    } else {
        printf("Valor 20 não encontrado.\n");
    }

    // Buscar valor inexistente
    resultado = buscarElemento(lista, 99);
    if (resultado != NULL) {
        printf("Valor %d encontrado no endereço %p.\n", resultado->valor, (void*)resultado);
    } else {
        printf("Valor 99 não encontrado.\n");
    }

    return 0;
}
```