

Tópicos de Programação

Arthur Casals
(arthur.casals@usp.br)

IME - USP

Aula 5:

- Fundamentos de Estruturas de Dados

Na aula passada...

Listas lineares (*arrays*):

- › Caso específico: *array* com duas dimensões
 - Uma lista linear bidimensional pode ser utilizada para representar uma matriz

Na aula passada...

Listas lineares ligadas:

› Motivação:

- *Arrays* possuem tamanho fixo (problemas de subdimensionamento ou superdimensionamento)
- Algumas operações sobre *arrays* são computacionalmente caras

Na aula passada...

Listas lineares ligadas:

› Vantagens sobre *arrays* :

- Tamanho dinâmico (podem crescer ou diminuir de acordo com a necessidade)
- Facilidade em operações de inclusão/exclusão de dados

Na aula passada...

Listas lineares ligadas:

› Desvantagens:

- Acesso aleatório (via posição) não é permitido
- Cada elemento requer espaço extra para ponteiro
- *Caching*

Na aula passada...

Listas lineares ligadas:

› Estrutura:

- Uma lista ligada é formada por nós
- Cada lista possui um nó inicial (*head*)
- Cada nó possui uma referência (apontador) para o próximo nó da lista

Na aula passada...

Operações em listas ligadas:

- › Inserção de elementos
- › Remoção de elementos
- › Tamanho
- › Busca (verificar se um elemento pertence à lista)
- › Troca de posições de nós (sem trocar os dados)

Antes de começarmos

Exemplos em C: página do curso

8 janeiro - 12 janeiro

08/01: Técnicas sistemáticas de desenvolvimento: representação de algoritmos, divisão e conquista, recursividade, iteração, programação dinâmica, planejamento reverso, metodologia para a construção de algoritmos. Introdução à eficiência de algoritmos.

09/01: Fundamentos de estruturas de dados: Listas lineares (*arrays*)

10/01: Fundamentos de estruturas de dados: Introdução a listas ligadas

11/01: Fundamentos de estruturas de dados: Listas ligadas

 [Aula 2 - 08/01](#)

 [Aula 3 - 09/01](#)


 [Aula 4 - 10/01](#)

 [Construindo novos tipos em C](#)

Exemplo de código em C utilizando construção de novos tipos de dados (`typedef`, `struct`)

 [Ponteiros em C: operadores * e &](#)

 [Exemplos em C: ponteiros, aritmética de ponteiros, vetores e matrizes](#)

 [Exemplos em C: Lista ligada](#)

Antes de começarmos

Material relacionado (não é obrigatório!):

› <https://github.com/ossu/computer-science> (inglês)

Fundamentos de estruturas de dados

Listas lineares duplamente ligadas:

› Motivação:

- Algumas vezes pode ser necessário (em termos de eficiência) possuir fácil acesso a todos os nós adjacentes de cada nó na lista
- Exemplos práticos: lista de músicas, históricos de navegadores

Fundamentos de estruturas de dados

Listas lineares duplamente ligadas:

- › Vantagens sobre listas ligadas:
 - Pode ser percorrida em ambos os sentidos
 - Operação de exclusão pode ser mais eficiente

Fundamentos de estruturas de dados

Listas lineares duplamente ligadas:

- › Desvantagens em relação a listas ligadas:
 - Cada elemento requer espaço extra para ponteiro adicional
 - Ponteiro adicional tem que ser mantido em todas as operações

Fundamentos de estruturas de dados

Listas lineares duplamente ligadas:

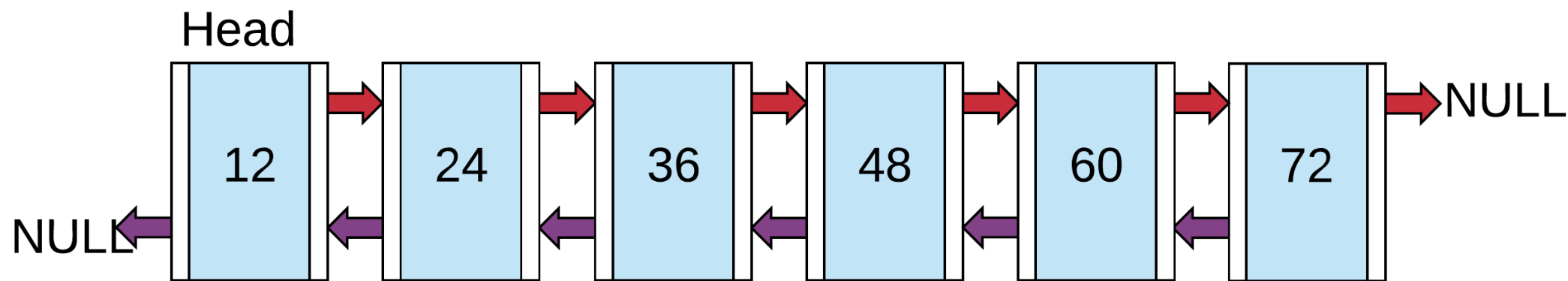
› Estrutura:

- Uma lista ligada é formada por nós
- Cada lista possui um nó inicial (*head*)
- Cada nó possui uma referência (apontador) para o próximo nó da lista
- Cada nó possui uma referência (apontador) para o nó anterior da lista

Fundamentos de estruturas de dados

Listas lineares duplamente ligadas:

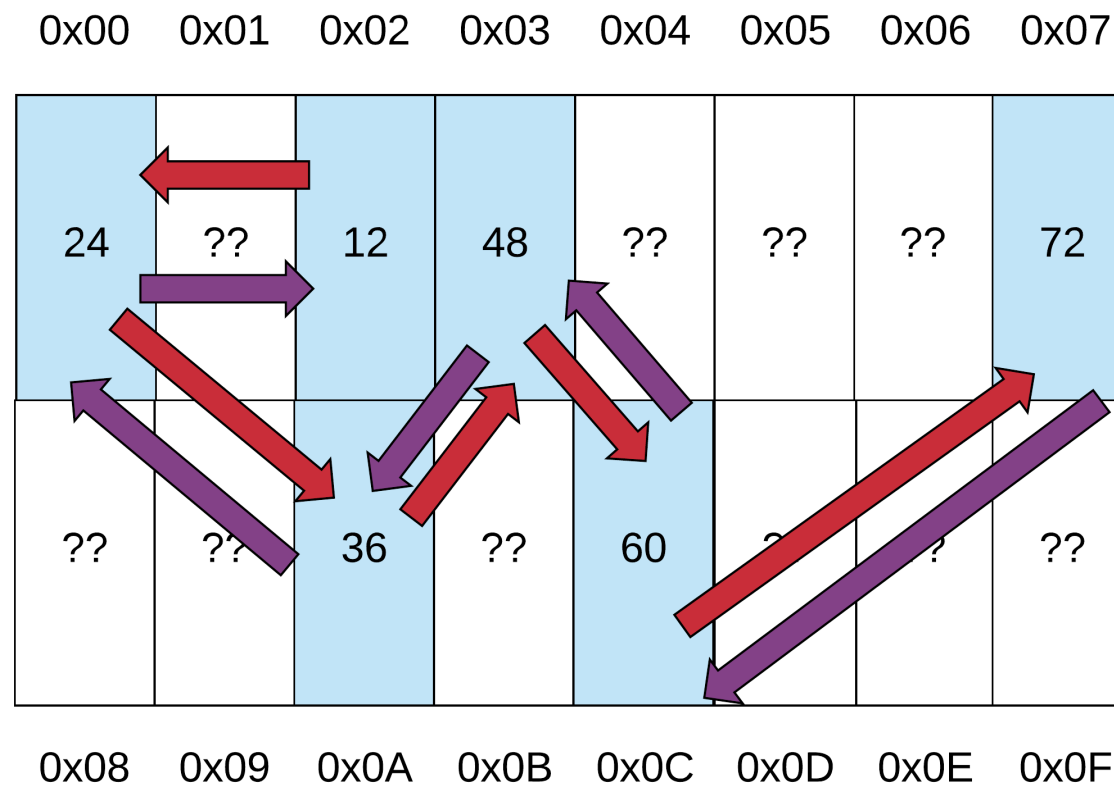
› Representação:



Fundamentos de estruturas de dados

Listas lineares duplamente ligadas:

› Em memória:



Fundamentos de estruturas de dados

Operações em listas duplamente ligadas:

- › Inserção de elementos
- › Remoção de elementos
- › Tamanho
- › Busca (verificar se um elemento pertence à lista)
- › Troca de posições de nós (sem trocar os dados)

Fundamentos de estruturas de dados

Listas duplamente ligadas:

› Representação em Java:

```
class Elemento
{
    int dado;
    Elemento proximo;
    Elemento anterior;
    Elemento(int i) {dado = d;}
}
```

Fundamentos de estruturas de dados

Listas duplamente ligadas:

› Representação em C:

//Elemento (nó) de uma lista ligada

```
struct Elemento {  
    int dado;  
    struct Elemento *proximo;  
    struct Elemento *anterior;  
};
```

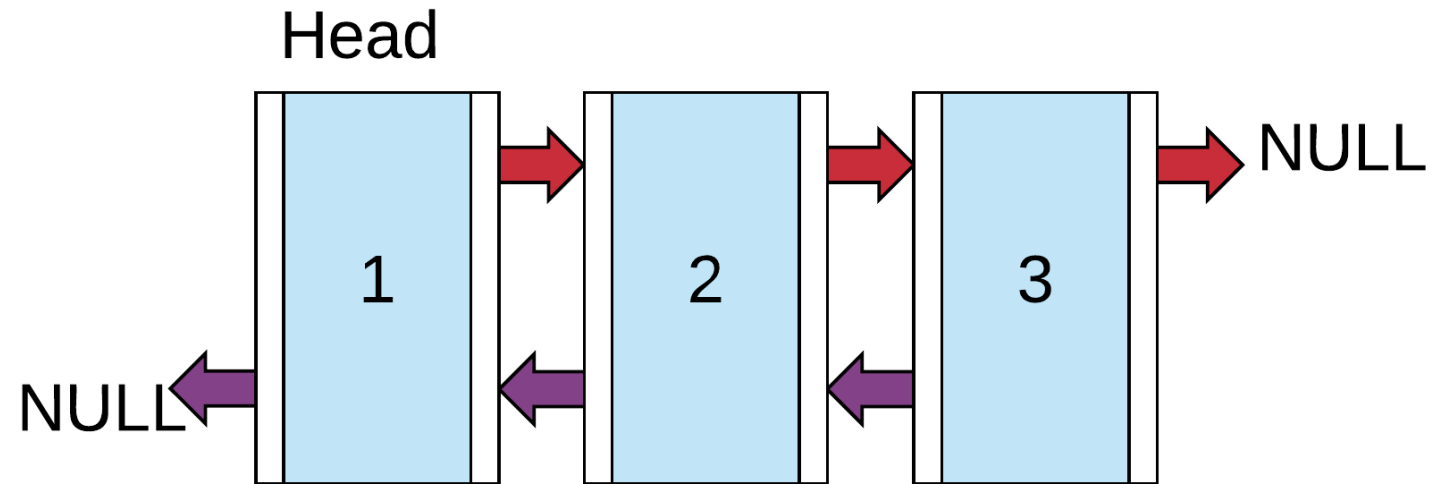
Fundamentos de estruturas de dados

Construindo uma lista duplamente ligada em C:

```
int main() {  
    struct Elemento* head = (struct Elemento*)malloc(sizeof(struct Elemento));  
    struct Elemento* segundo = (struct Elemento*)malloc(sizeof(struct Elemento));  
    struct Elemento* terceiro = (struct Elemento*)malloc(sizeof(struct Elemento));  
    head->dado = 1;  
    head->proximo = segundo;  
    head->anterior = NULL;  
    segundo->dado = 2;  
    segundo->proximo = terceiro;  
    segundo->anterior = head;  
    terceiro->dado = 3;  
    terceiro->proximo = NULL;  
    terceiro->anterior = segundo;  
    return 0;  
}
```

Fundamentos de estruturas de dados

Construindo uma lista duplamente ligada em C:



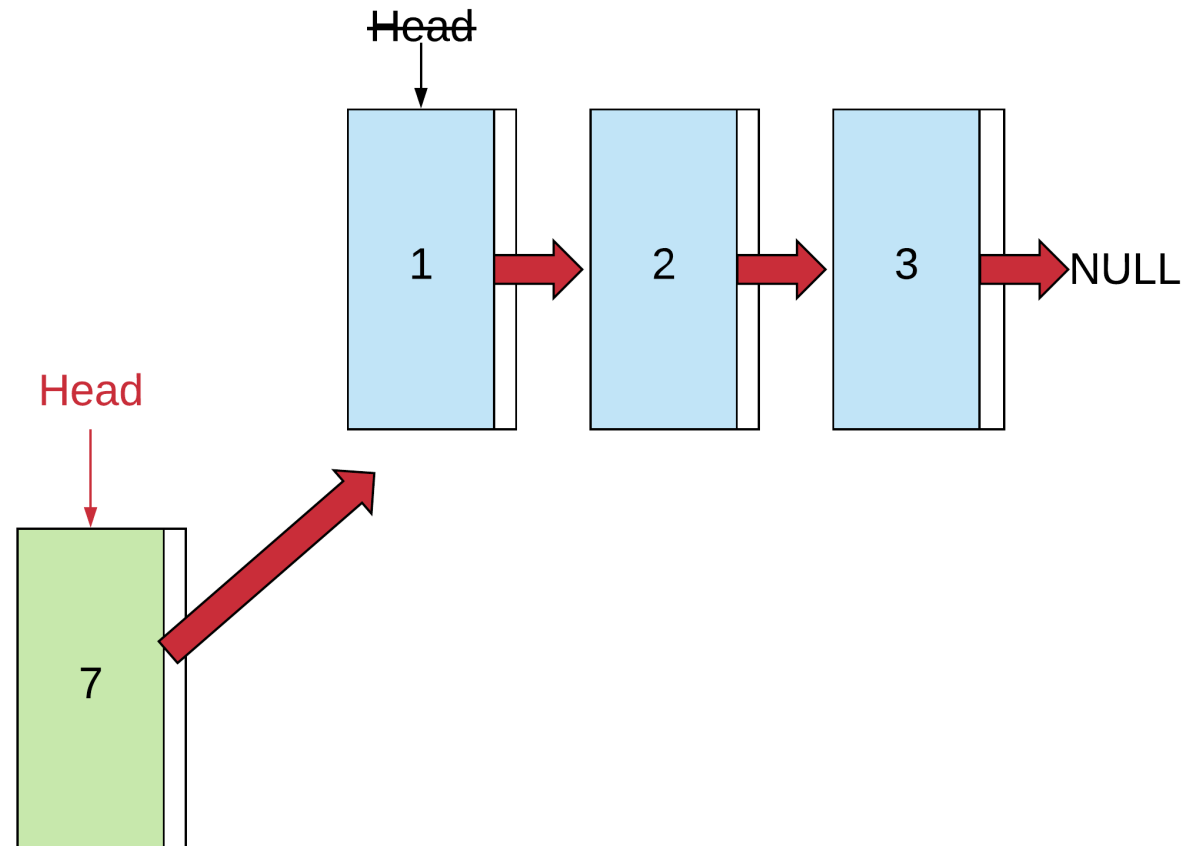
Fundamentos de estruturas de dados

Inserção de elementos:

- › No começo da lista
- › Depois de um determinado nó
- › Antes de um determinado nó
- › Ao final da lista

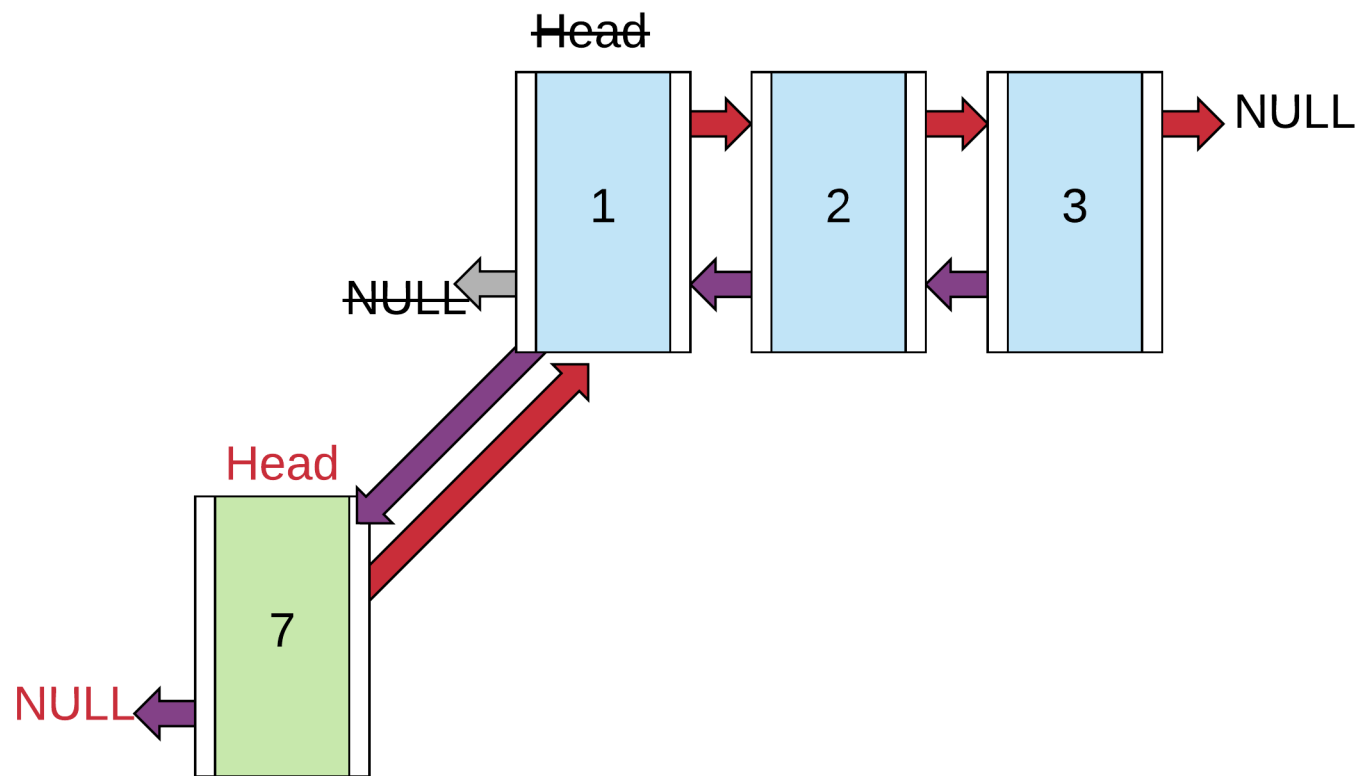
Fundamentos de estruturas de dados

Inserção de elementos no começo da lista ligada:



Fundamentos de estruturas de dados

Inserção de elementos no começo da lista duplamente ligada:



Fundamentos de estruturas de dados

```
void insere(struct Elemento** head_ref, int novo_dado)
{
    struct Elemento* no_novo = (struct Elemento*) malloc(sizeof(struct Elemento));
    no_novo->dado = novo_dado;
    no_novo->proximo = (*head_ref);
    no_novo->anterior = NULL;
    (*head_ref) ->anterior = no_novo;
    (*head_ref) = no_novo;
}
```


Fundamentos de estruturas de dados

```
no_novo->anterior = NULL;
```

-> O nó novo vai estar no início da lista, então não possui nó anterior.

Fundamentos de estruturas de dados

```
(*head_ref)->anterior = no_novo;
```

-> O atual nó inicial , referenciado por (*head_ref), vai deixar de ser o primeiro nó. Assim, ele passa a possuir um nó anterior, que é o novo element inserido na lista.

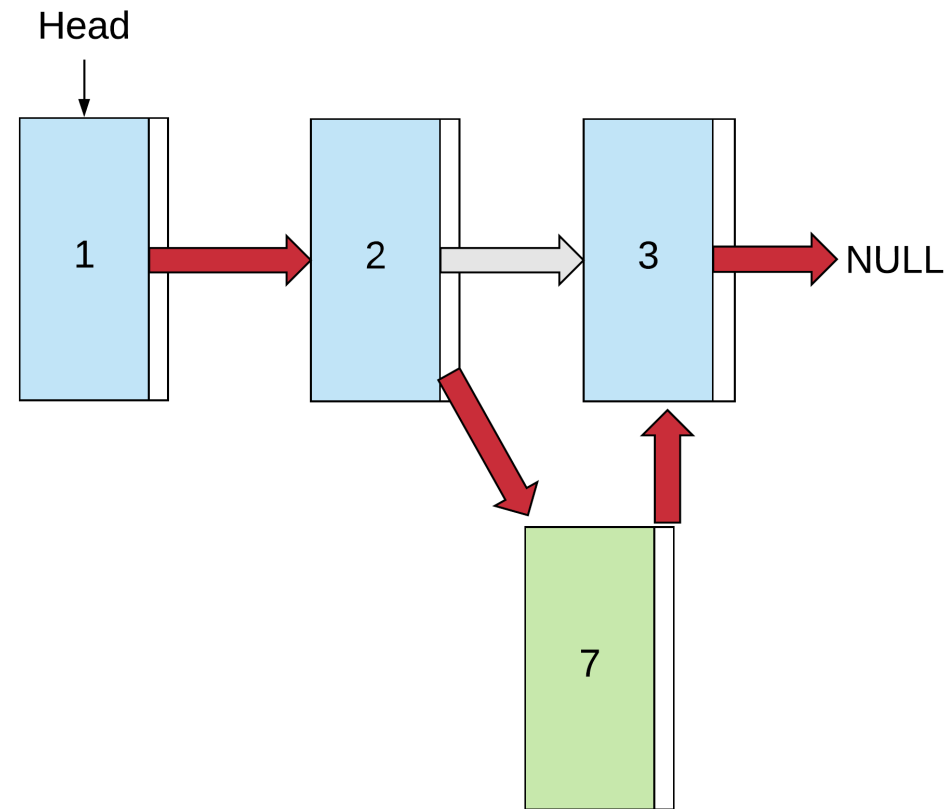
Fundamentos de estruturas de dados

```
(*head_ref) = no_novo;
```

-> No final do processo, a referência para o primeiro nó da lista (*head_ref) tem que ser atualizada. Esta referência passa a apontar para o novo primeiro nó da lista.

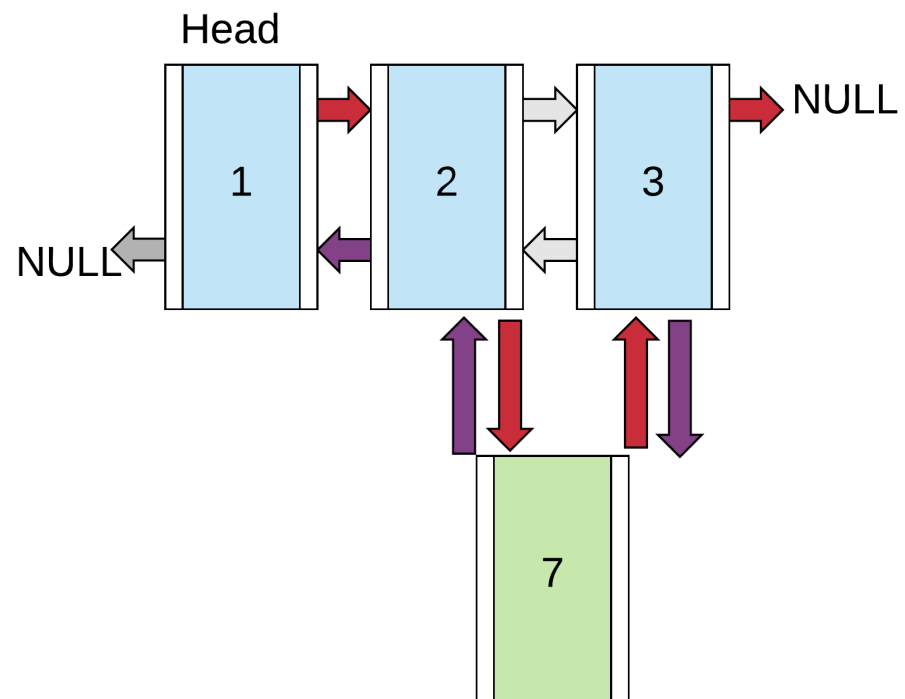
Fundamentos de estruturas de dados

Inserção de elementos depois de um determinado nó na lista ligada:



Fundamentos de estruturas de dados

Inserção de elementos antes ou depois de um determinado nó na lista duplamente ligada:



Fundamentos de estruturas de dados

```
void insereDepois(struct Elemento* no_anterior, int novo_dado)
{
    //checar se no_anterior é nulo
    struct Elemento* no_novo =(struct Elemento*) malloc(sizeof(struct Elemento));
    no_novo->dado  = novo_dado;
    no_novo->proximo = no_anterior->proximo;
    no_anterior->proximo = no_novo;
    no_novo->anterior = no_anterior;
    no_novo->proximo->anterior = no_novo;
}
```

Fundamentos de estruturas de dados

```
void insereAntes(struct Elemento* no_proximo, int novo_dado)
{
    //checar se no_anterior é nulo
    struct Elemento* no_novo =(struct Elemento*) malloc(sizeof(struct Elemento));
    no_novo->dado  = novo_dado;
    no_novo->proximo = no_proximo;
    no_novo->anterior = no_proximo->anterior;
    no_proximo->anterior = no_novo;
    no_novo->anterior->proximo = no_novo;
}
```

Fundamentos de estruturas de dados

```
no_novo->proximo = no_proximo;
```

```
no_novo->anterior = no_proximo->anterior;
```

-> Após criarmos o nó novo, o primeiro passo é atualizar os ponteiros do nó criado. Desta forma, não perdemos nenhuma referência.

Fundamentos de estruturas de dados

```
no_proximo->anterior = no_novo;
```

-> Atualizamos a referência "anterior" do **no_proximo** para apontar para o novo nó criado.

Fundamentos de estruturas de dados

```
no_novo->anterior->proximo = no_novo;
```

-> Atualizamos a referência "proximo" do nó que originalmente estava antes de **no_proximo** para apontar para o novo nó criado.

Fundamentos de estruturas de dados

Inserção de elementos ao final da lista:

- Caso particular do exemplo anterior (inserção depois do último nó)
- Basta percorrer a lista até achar um nó que não aponta para ninguém (`elemento->proximo == NULL`)

Fundamentos de estruturas de dados

Remoção de elementos da lista:

- A partir de uma determinada posição
- A partir de um determinado nó (valor)

Fundamentos de estruturas de dados

Remoção de elementos da lista por valor/posição:

- Achar o nó anterior ao nó a ser removido;
- Modificar o ponteiro "próximo" do nó achado;
- Modificar o ponteiro "anterior" do nó achado;
- Liberar a memória do nó removido

Fundamentos de estruturas de dados

Tamanho de uma lista:

- Por iteração
- Por recursão

Fundamentos de estruturas de dados

Tamanho de uma lista:

- Por iteração
- Por recursão
- Algoritmos idênticos!

Fundamentos de estruturas de dados

Busca em uma lista:

- Por iteração
- Por recursão
- Algoritmos idênticos!

Fundamentos de estruturas de dados

EXERCÍCIO: remover nós de uma lista duplamente ligada:

- Exemplo de lista original: 3<->4<->7<->9<->11<->13
- Exemplo de entradas: *head, (apontando para o nó com valor igual a 3), x = 4,
- A remoção pode ser por valor ou posição
- Exemplo de resultado (valor): 3<->7<->9<->11<->13
- Exemplo de resultado (posição): 3<->4<->7<->9<->13
- Façam o algoritmo para uma lista qualquer

Fundamentos de estruturas de dados

Listas circulares:

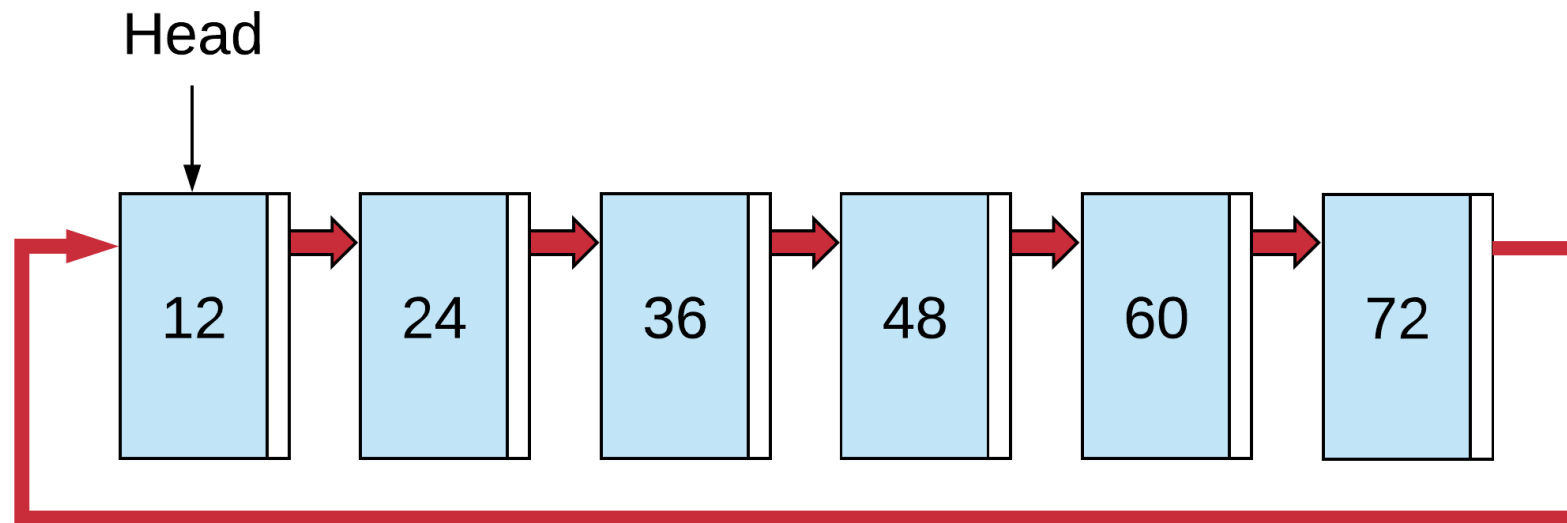
› Motivação:

- Alocação de memória previsível (evita *malloc/free*)
- É mais simples percorrer uma lista circular
- Sequenciamento de dados limitados
- Exemplo prático: de quem é a vez?, listas de música

Fundamentos de estruturas de dados

Listas circulares ligadas:

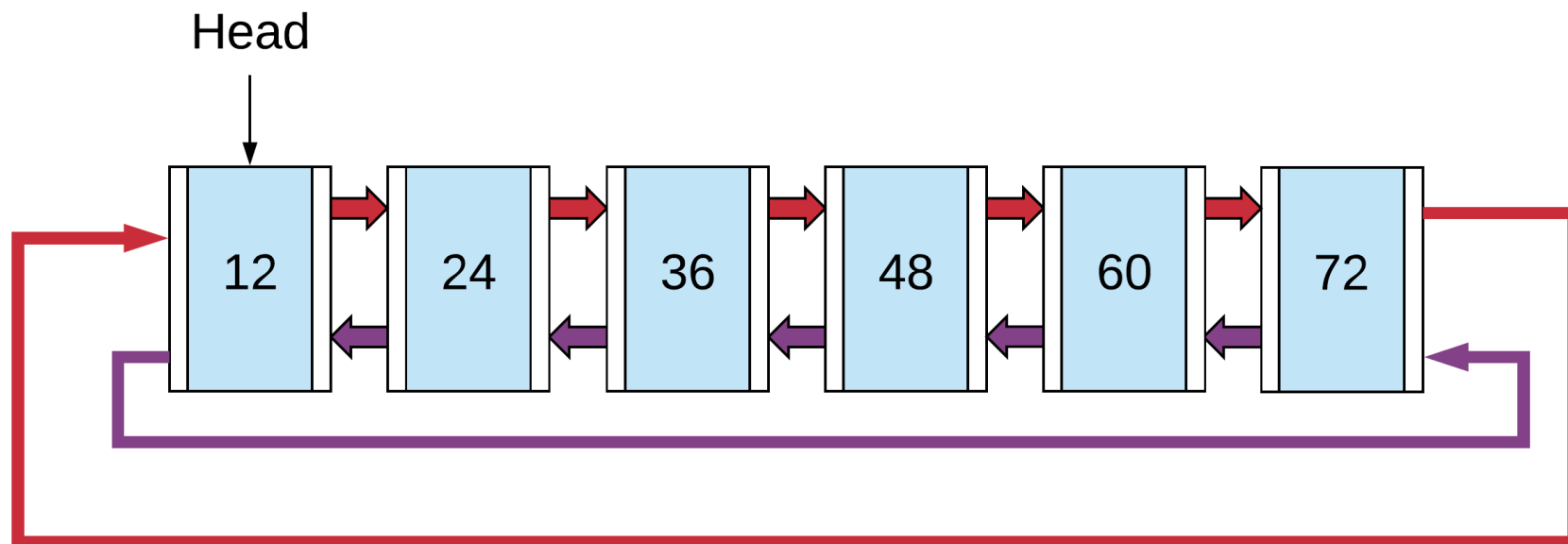
› Representação:



Fundamentos de estruturas de dados

Listas circulares duplamente ligadas:

› Representação:



Fundamentos de estruturas de dados

Operações em listas circulares:

- › Inserção de elementos
- › Remoção de elementos
- › Tamanho
- › Busca (verificar se um elemento pertence à lista)
- › Troca de posições de nós (sem trocar os dados)

...são iguais ou mais simples que listas linearmente ligadas!

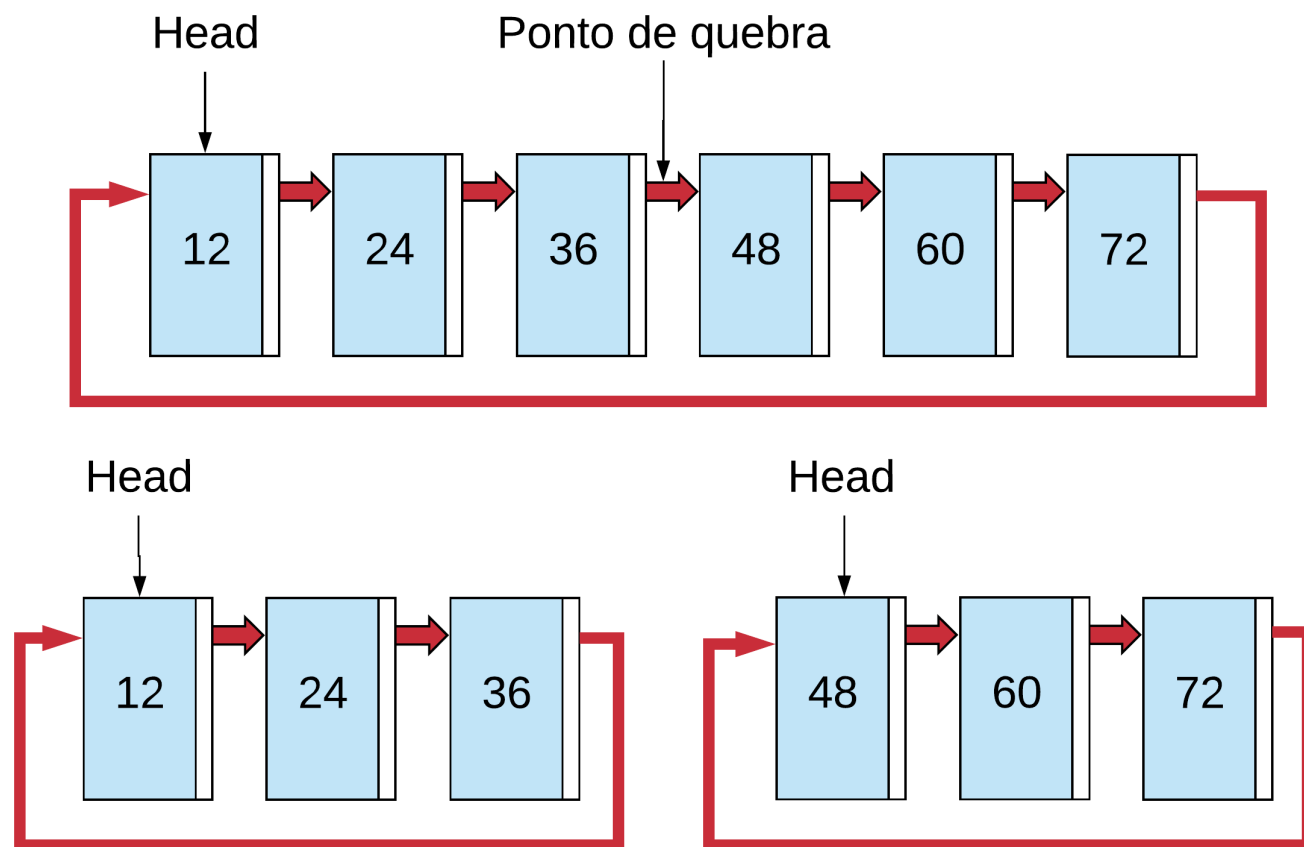
Fundamentos de estruturas de dados

Outras operações em listas circulares:

- › Dividir uma lista circular em duas listas circulares menores
- › Repetir uma operação n vezes
- › Auto-organização

Fundamentos de estruturas de dados

Dividir uma lista circular em duas menores:



Fundamentos de estruturas de dados

Dividir uma lista circular em duas menores:

- › Achar o ponto de quebra
- › Apontar o "último nó" para o nó do ponto de quebra
- › Apontar o nó anterior ao ponto de quebra para *head*
- › Atualizar ponteiro de *head* das listas, quando necessário