

ED1 – Estrutura de Dados I

Aulas 5, 6 e 7 Listas Encadeadas Simples

José Américo
jose.americo@ifsp.edu.br
Samuel Martins (Samuka)
samuel.martins@ifsp.edu.br

1º Mandamento de Estruturas de Dados

“Sempre desenhar a estrutura de dados antes de implementá-la...”



Um problema em vetores/array

- Suponha que queremos armazenar um lista de alunos em um **vetor/array**;
- Sabemos, inicialmente, que a **quantidade de alunos** é 5;
- Então, alocamos um vetor de 5 posições e inserimos os alunos;

Um problema em vetores/array

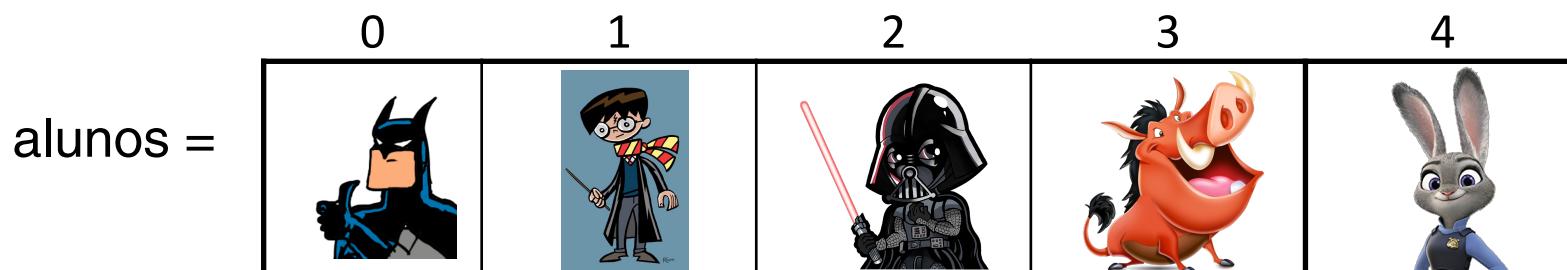
- Suponha que queremos armazenar um lista de alunos em um **vetor/array**;
- Sabemos, inicialmente, que a **quantidade de alunos** é 5;
- Então, alocamos um vetor de 5 posições e inserimos os alunos;

```
Alunos **alunos = (Alunos**) malloc(5, sizeof(Alunos*));
for (int i = 0; i < 5; i++) {
    // alunos[i] = (Aluno*) malloc(1, sizeof(Aluno));
    aluno[i] = CreateAluno();
}
```

Um problema em vetores/array

- Suponha que queremos armazenar um lista de alunos em um **vetor/array**;
- Sabemos, inicialmente, que a **quantidade de alunos** é 5;
- Então, alocamos um vetor de 5 posições e inserimos os alunos;

```
Alunos **alunos = (Alunos**) malloc(5, sizeof(Alunos*));
for (int i = 0; i < 5; i++) {
    // alunos[i] = (Aluno*) malloc(1, sizeof(Aluno));
    aluno[i] = CreateAluno();
}
```



Um problema em vetores/array

- E se agora quiséssemos **adicionar** um novo aluno neste vetor?
- O vetor atual **não tem espaço suficiente**. O que fazer?



alunos =

0	1	2	3	4

Um problema em vetores/array

Opção 1:

- Existe uma função em C chamada **realloc** que realoca um espaço de memória —> **comando perigoso**
- Quando mal usado, acarretará em perda de dados
- **Não indicado!**



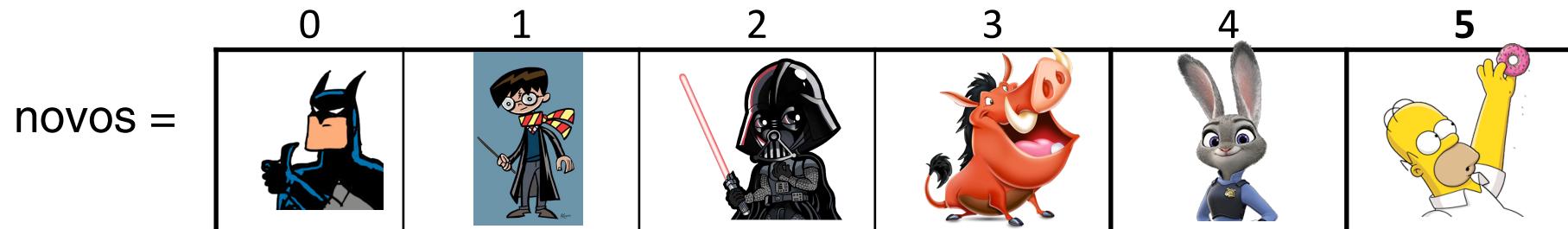
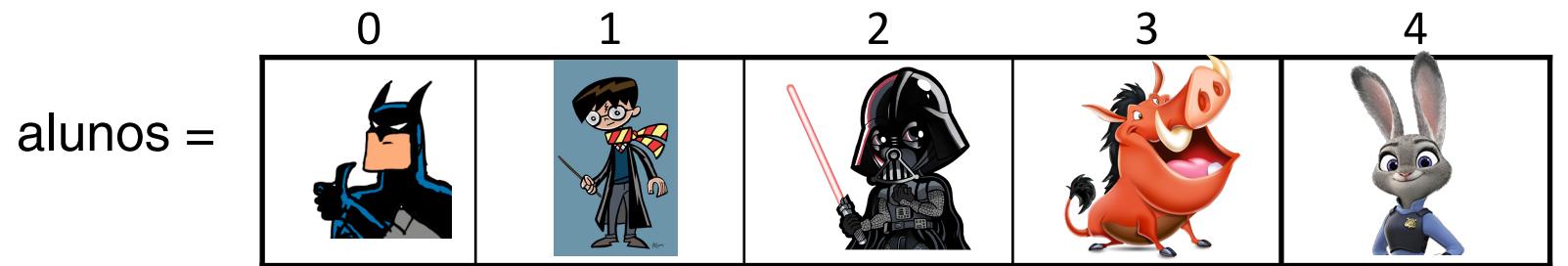
alunos =

0	1	2	3	4
A cartoon illustration of Batman, showing his upper body and head. He is wearing his classic black suit with the cowl and bat emblem. He is giving a thumbs-up gesture with his right hand.	A cartoon illustration of Harry Potter, showing him from the waist up. He is wearing his signature round glasses, a blue shirt, and a Gryffindor house robe with a yellow and red striped scarf around his neck. He is holding a wand in his right hand.	A cartoon illustration of Darth Vader, showing him from the waist up. He is wearing his iconic black armor with the white帝 logo on the chest. He is holding a red lightsaber in his right hand.	A cartoon illustration of Pumbaa, the warthog from 'The Lion King'. He is shown from the waist up, wearing a small blue vest over a red shirt. He has a wide-open mouth showing his tongue and teeth, and his mouth is drooping downwards.	A cartoon illustration of Judy Hopps from 'Zootopia', showing her from the waist up. She is a rabbit with long ears, wearing a blue police uniform with a badge on her chest. She is standing with her hands at her sides.

Um problema em vetores/array

Opção 2:

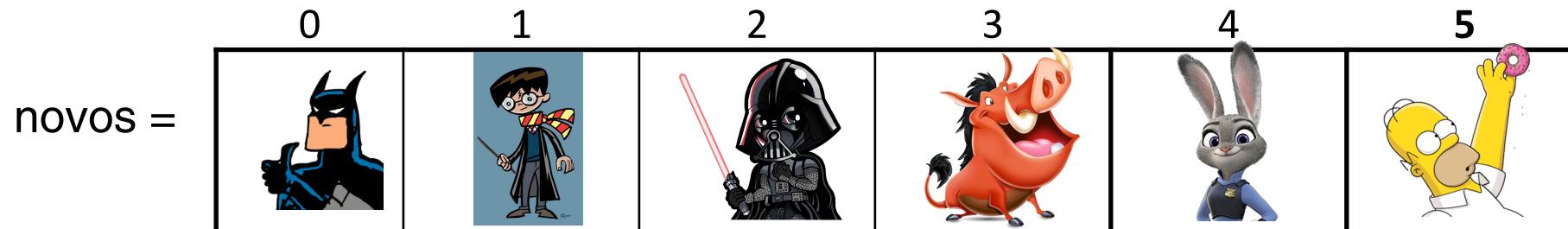
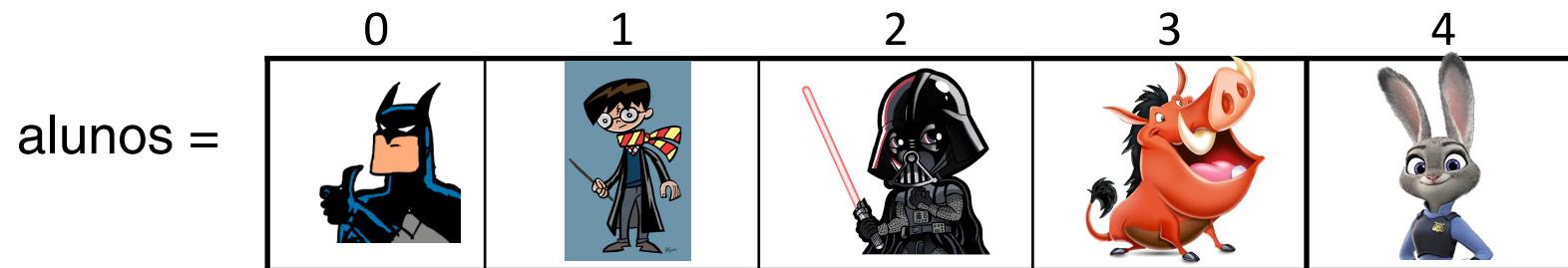
- Criar um vetor com 6 posições e **copiar todos** os valores do antigo vetor e **adicionar** o novo aluno;



Um problema em vetores/array

Opção 2:

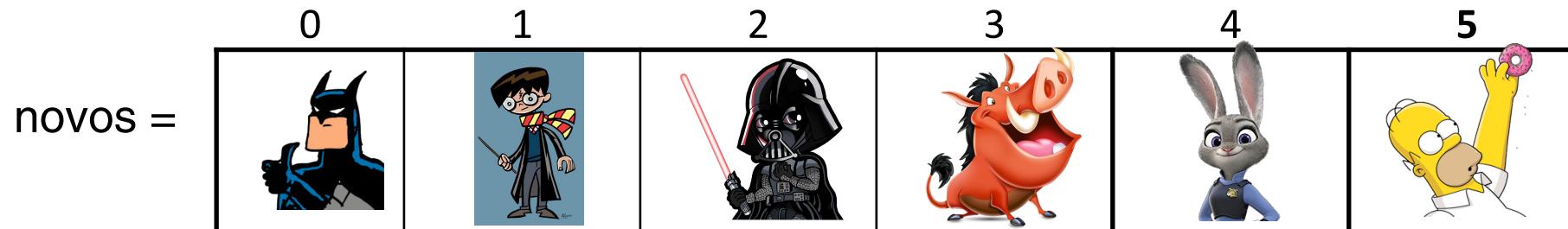
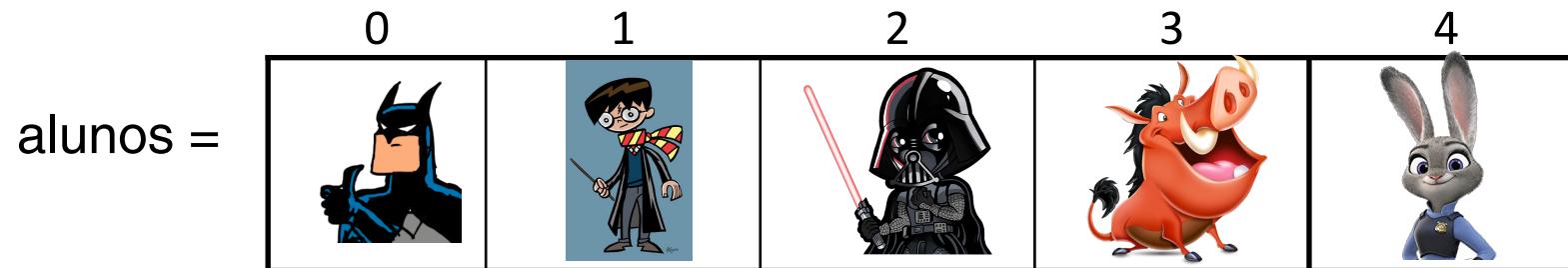
- Criar um vetor com 6 posições e **copiar todos** os valores do antigo vetor e **adicionar** o novo aluno;
- Problemas?



Um problema em vetores/array

Opção 2:

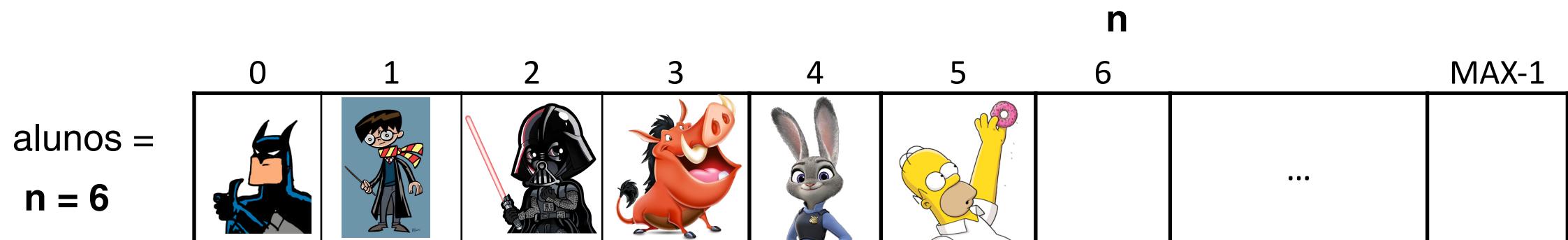
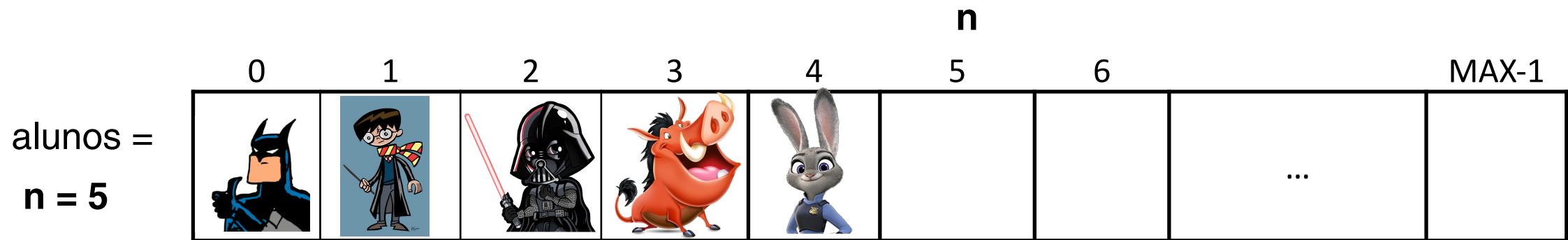
- Criar um vetor com **6** posições e **copiar todos** os valores do antigo vetor e **adicionar** o novo aluno;
- **Problemas?**
 - A cada novo aluno, precisaremos alocar um novo vetor;
 - Copiar os dados de um vetor a outro... **ineficiente**;



Um problema em vetores/array

Opção 3:

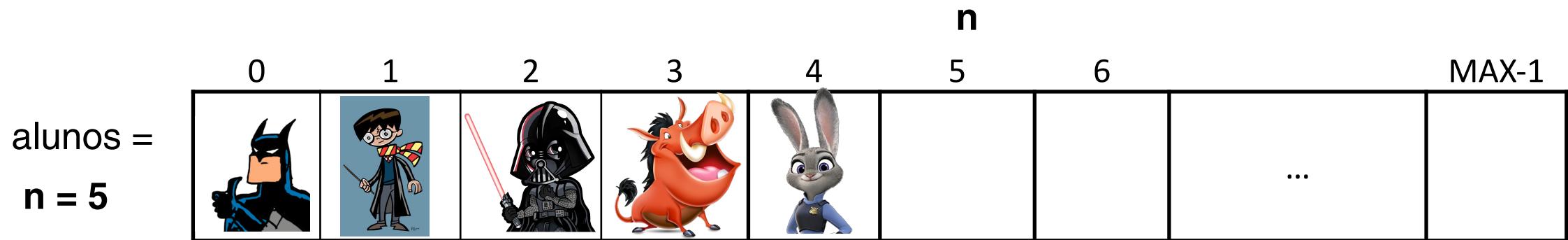
- Criamos um **vetor muito grande**, com muitas posições, de modo que “**sempre**” teremos espaço para inserir um novo aluno;



Um problema em vetores/array

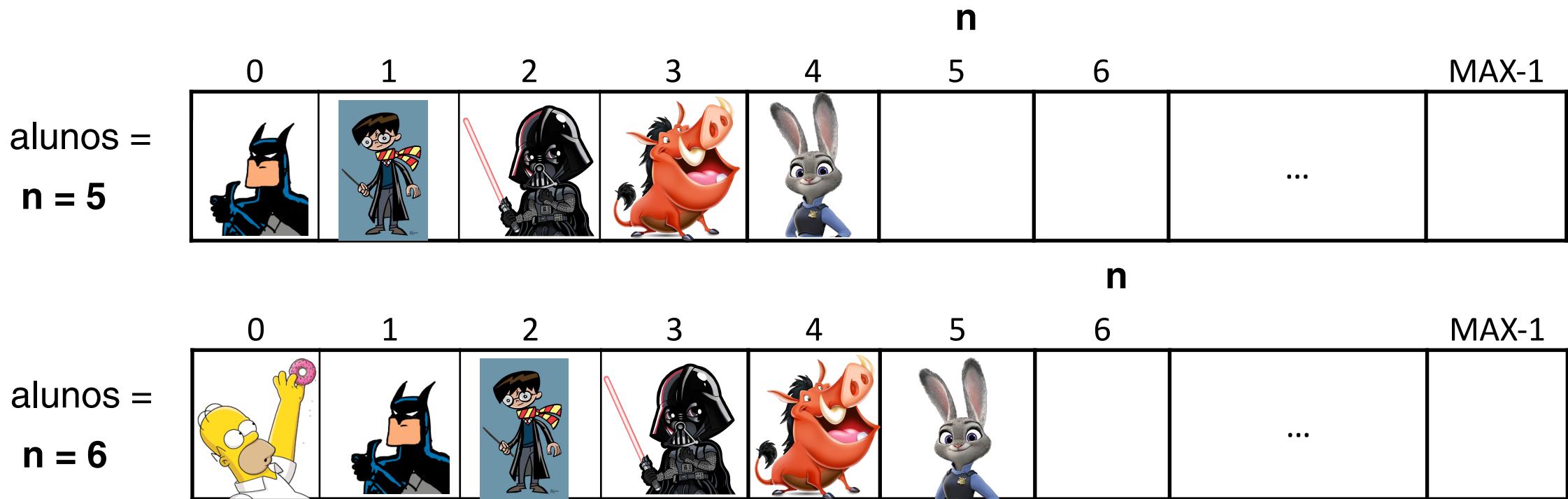
Opção 3:

- Criamos um **vetor muito grande**, com muitas posições, de modo que “**sempre**” teremos espaço para inserir um novo aluno;
- **Problemas?**



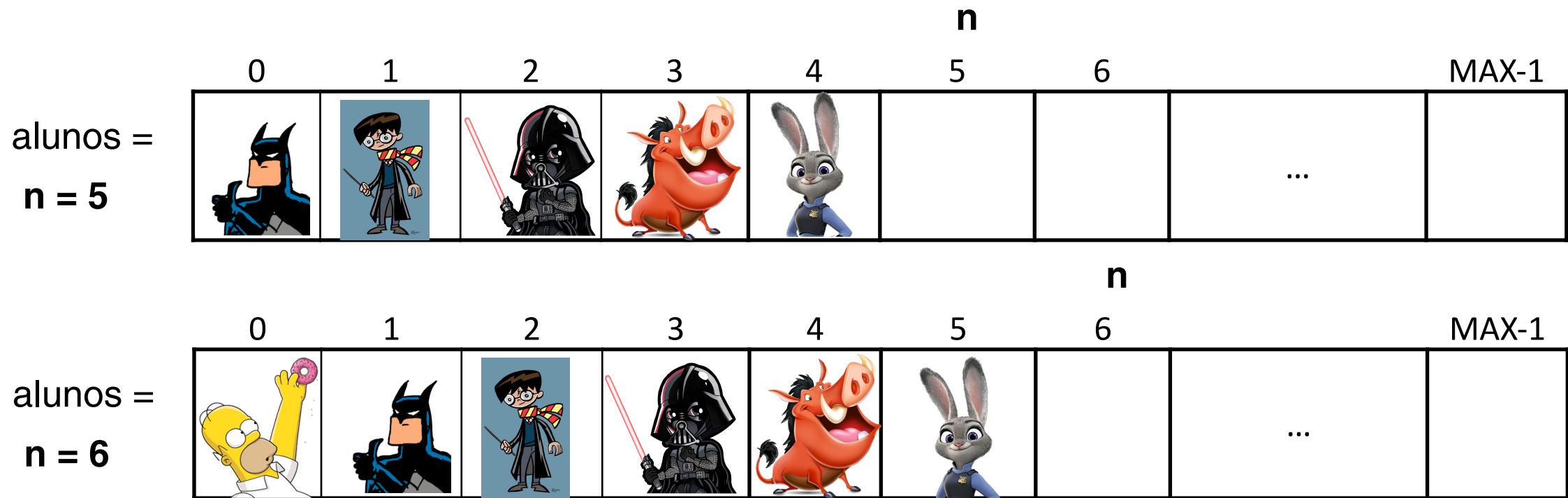
Outro problema em vetores/array

- Suponha que adotamos a **opção 3** para a implementação;
- Suponha que um novo aluno **sempre** deva ser inserido no começo do vetor;
- O que fazer?



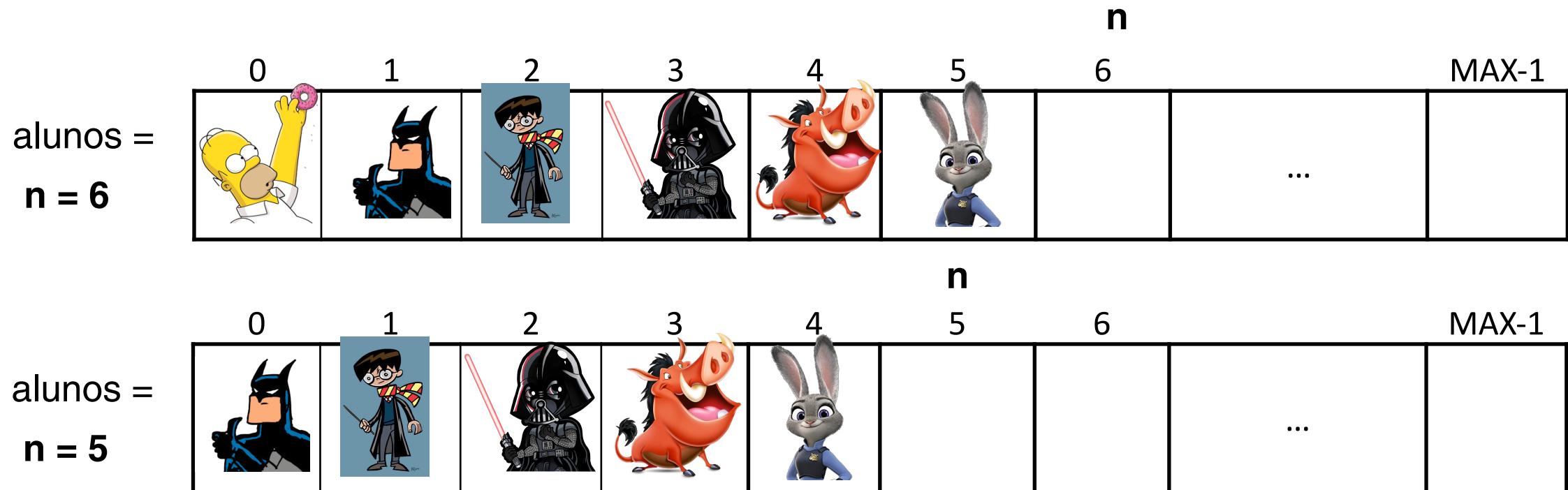
Outro problema em vetores/array

- Suponha que adotamos a **opção 3** para a implementação;
- Suponha que um novo aluno **sempre** deva ser inserido no começo do vetor;
- O que fazer?
- **Problemas?**



Outro problema em vetores/array

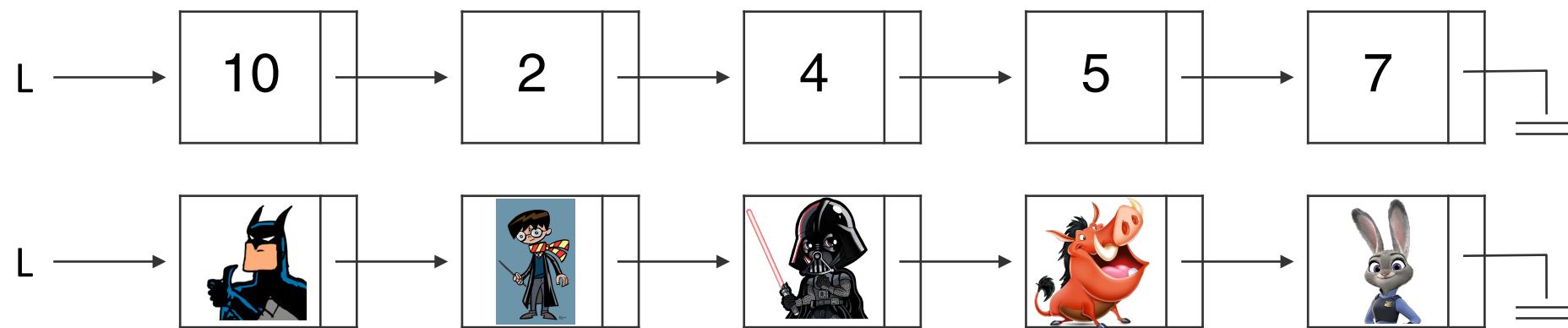
- O mesmo **problema** acontecerá se quisermos **remover** um aluno, mas quisermos **manter a ordem** a qual eles foram inseridos no vetor;



Listas Encadeadas

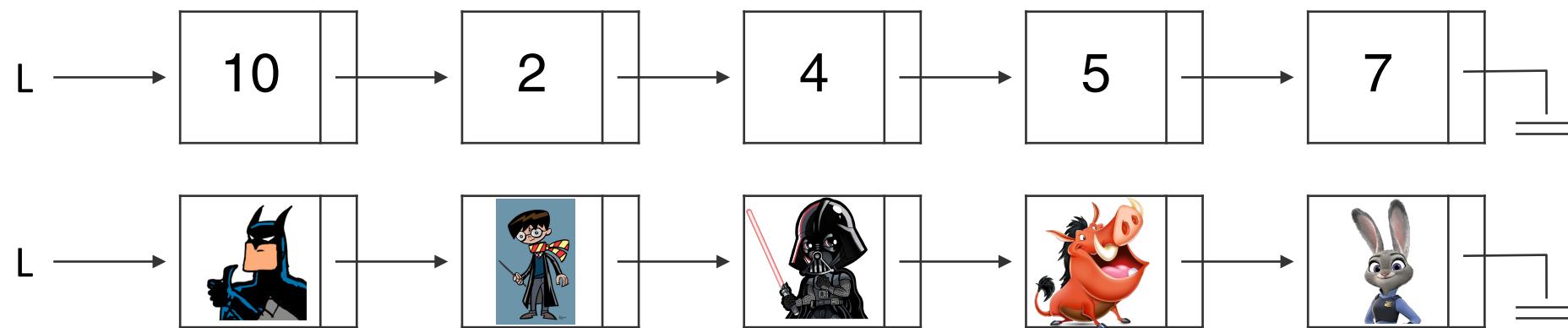
Lista Encadeada

- Uma **lista encadeada (ligada)** é uma representação de uma sequência de elementos/objetos na memória do computador;
- Os elementos, **nós da lista**, são armazenados em **posições quaisquer da memória** e são ligados por **ponteiros**;
 - Logo, elementos consecutivos da lista não ficam necessariamente em posições consecutivas na memória;



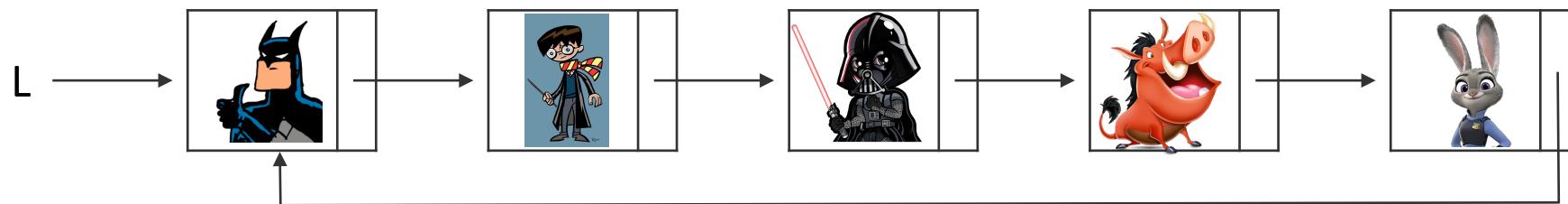
Lista Encadeada

- Cada **nó** contém um **elemento/objeto** de determinado tipo e um **ponteiro para o próximo elemento** da lista —> **Lista Encadeada Simples**;
- No caso do **último nó**, este ponteiro aponta para **NULL**;



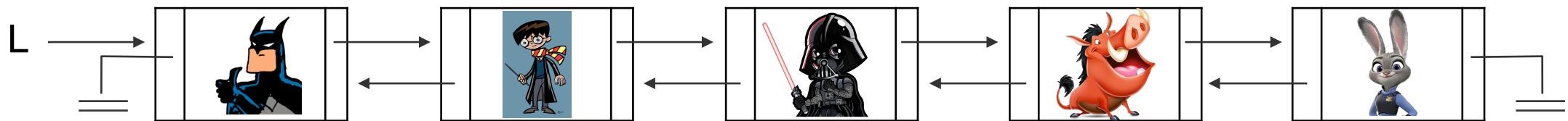
Lista Encadeada

- Podemos ainda ter outros tipos de listas (que detalharemos depois):
 - **Listas Circulares;**



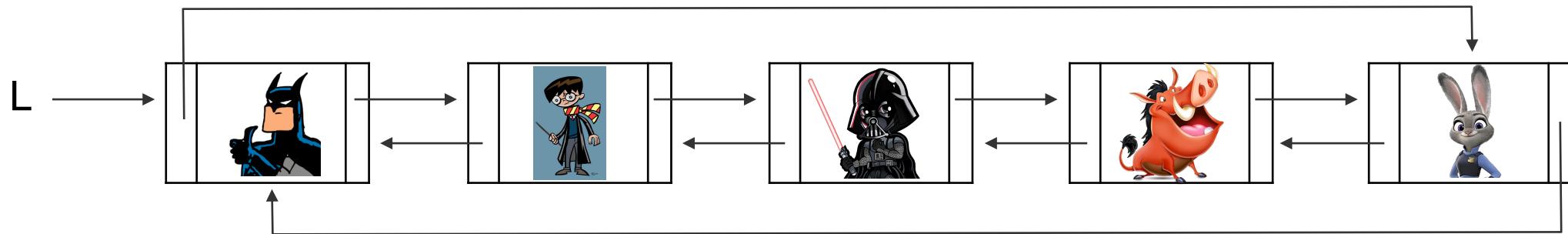
Lista Encadeada

- Podemos ainda ter outros tipos de listas (que detalharemos depois):
 - Listas Circulares;
 - [Listas Duplamente Encadeadas](#);



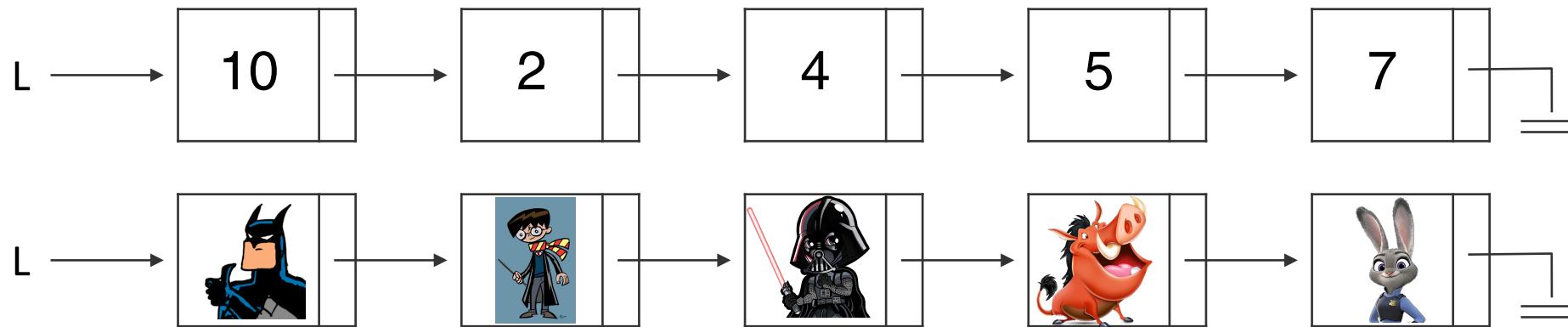
Lista Encadeada

- Podemos ainda ter outros tipos de listas (que detalharemos depois):
 - Listas Circulares;
 - Listas Duplamente Encadeadas;
 - **Listas Circulares Duplamente Encadeadas;**



Listas Encadeadas Simples

Lista Encadeada Simples



Lista Encadeada Simples

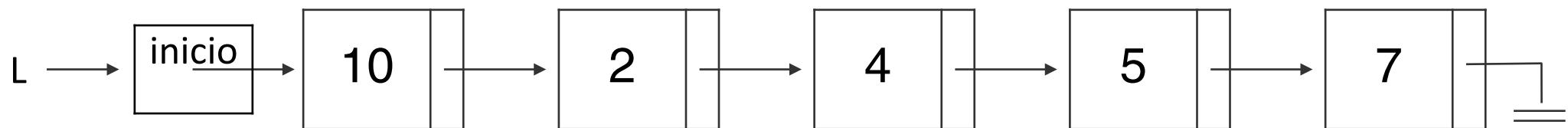
- **Implementação:**

- Uma Lista Encadeada é uma sequência de **Nós**;
- Se o ponteiro L aponta para o **primeiro nó** da Lista, podemos dizer que **L é uma Lista**;

```
typedef struct _no {  
    int val;  
    struct _no *prox;  
} No;
```

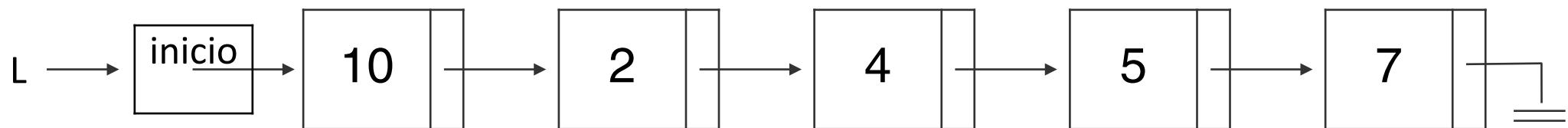
Lista Encadeada Simples

- **Outra Implementação:**
 - Temos um tipo próprio para a Lista, que guardará um ponteiro para seu **início**;



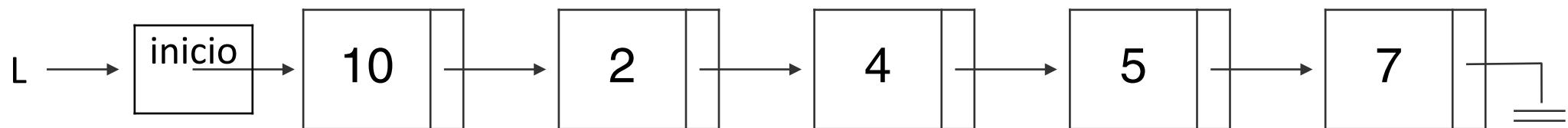
Lista Encadeada Simples

- **Outra Implementação:**
 - Temos um tipo próprio para a Lista, que guardará um ponteiro para seu **início**;
 - Este novo tipo nos dá a liberdade para guardarmos outras informações, como o número de nós da Lista, outros ponteiros, etc...



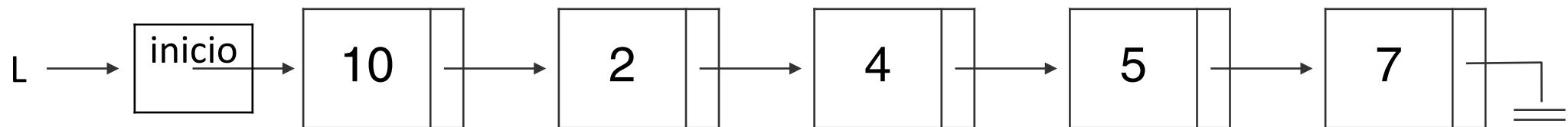
Lista Encadeada Simples

- **Outra Implementação:**
 - Temos um tipo próprio para a Lista, que guardará um ponteiro para seu **início**;
 - Este novo tipo nos dá a liberdade para guardarmos outras informações, como o número de nós da Lista, outros ponteiros, etc...
 - Com isso, podemos **otimizar** certas operações comuns de Listas Encadeadas;



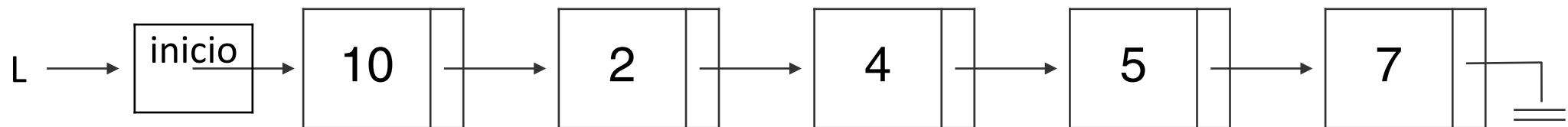
Lista Encadeada Simples

- **Outra Implementação:**
 - Temos um tipo próprio para a Lista, que guardará um ponteiro para seu **íncio**;
 - Este novo tipo nos dá a liberdade para guardarmos outras informações, como o número de nós da Lista, outros ponteiros, etc...
 - Com isso, podemos **otimizar** certas operações comuns de Listas Encadeadas;
 - Esta solução é parecida com a Lista Encadeada **com Cabeça** (ver referência), porém **mais robusta**;



Lista Encadeada Simples

- **Outra Implementação:**
 - Temos um tipo próprio para a Lista, que guardará um ponteiro para seu **íncio**;
 - Este novo tipo nos dá a liberdade para guardarmos outras informações, como o número de nós da Lista, outros ponteiros, etc...
 - Com isso, podemos **otimizar** certas operações comuns de Listas Encadeadas;
 - Esta solução é parecida com a Lista Encadeada **com Cabeça** (ver referência), porém **mais robusta**;
 - Utilizaremos essa versão;

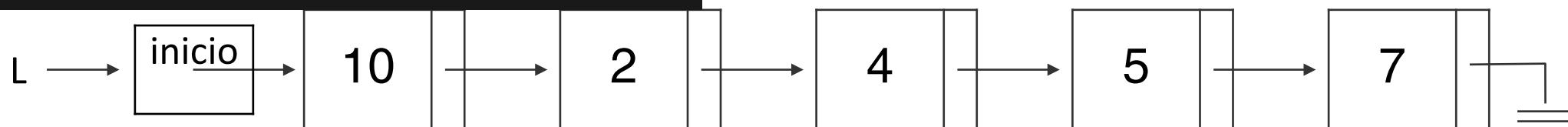


Lista Encadeada Simples

- Outra Implementação

```
typedef struct _no {  
    int val;  
    struct _no *prox;  
} No;
```

```
typedef struct _lista {  
    No *inicio;  
} Lista;
```



Listas Encadeadas Simples

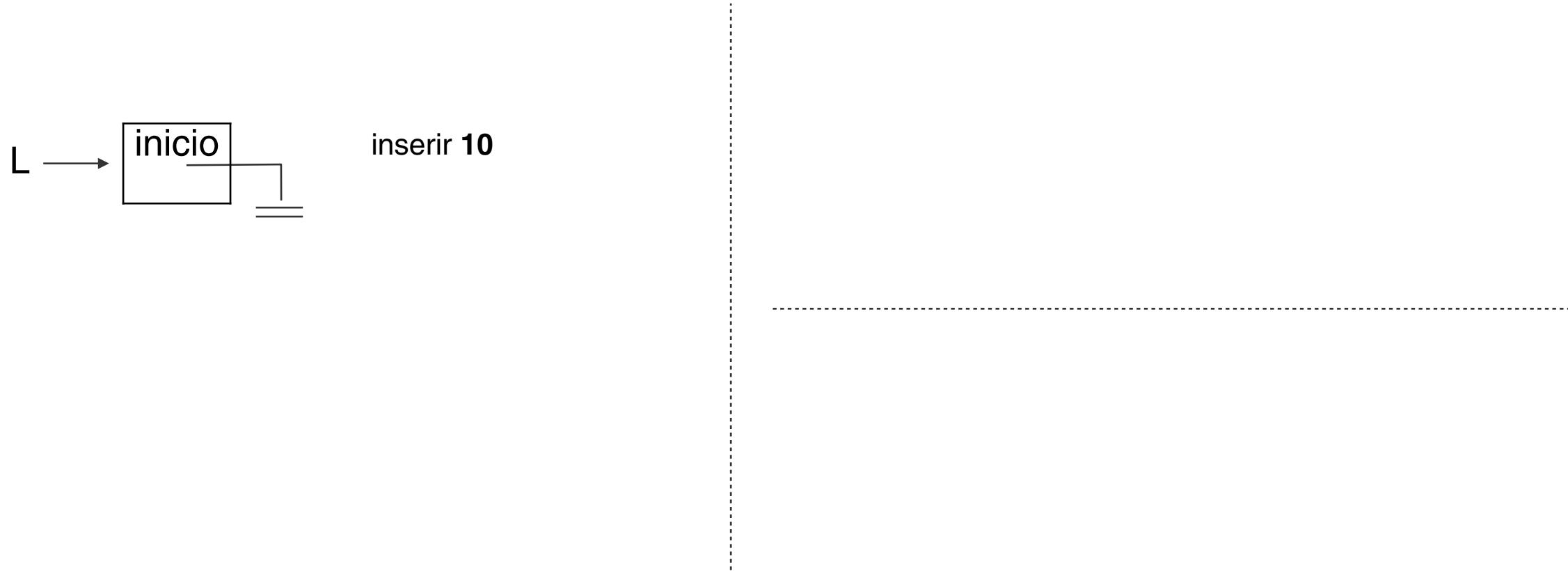
- Diversos tipos de **operações**:
 - Inserção na cabeça (índice 0) da lista;
 - Impressão dos Elementos da Lista
 - Inserção na cauda (índice n-1) da lista;
 - Remover elementos da lista;
 - Contar o número de elementos da Lista;
 - Verificar se a lista está vazia e retornar verdadeiro/falso
 - Concatenar de duas listas;
 - Copiar uma lista;
 - Ordenação de uma lista;
 - Buscar um dado elemento na lista e retornar seu ponteiro;
 - Inserção ordenada;

Listas Encadeadas Simples

- Diversos tipos de **operações**:
 - Inserção na cabeça (início) da lista;
 - Impressão dos Elementos da Lista
 - Inserção na cauda (fim) da lista;
 - Remover elementos da lista;
 - Contar o número de elementos da Lista;
 - Verificar se a lista está vazia e retornar verdadeiro/falso
 - Concatenar de duas listas;
 - Copiar uma lista;
 - Ordenação de uma lista;
 - Buscar um dado elemento na lista e retornar seu ponteiro;
 - Inserção ordenada;

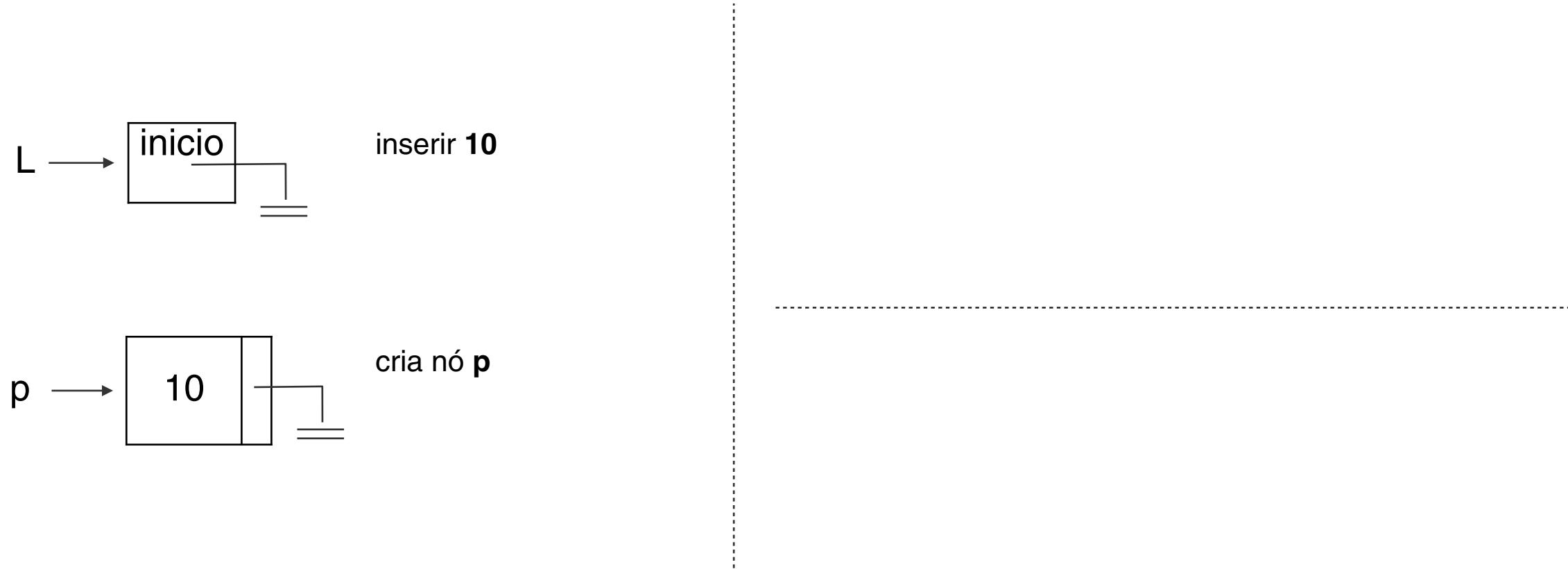
Inserção no início (cabeça) da lista

- **Caso 1:** Lista está vazia



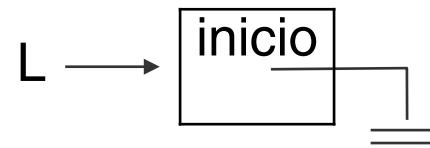
Inserção no início (cabeça) da lista

- **Caso 1:** Lista está vazia

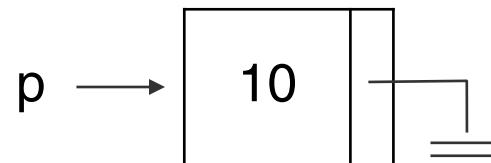


Inserção no início (cabeça) da lista

- **Caso 1:** Lista está vazia

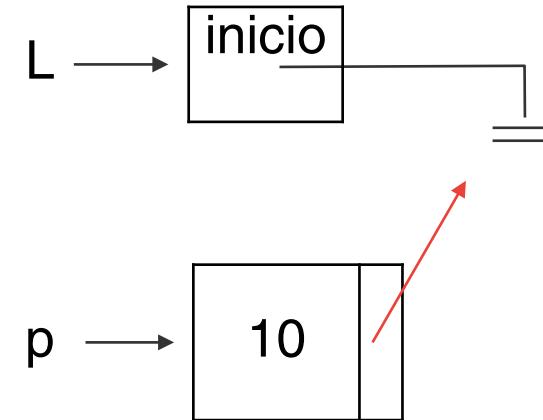


inserir 10



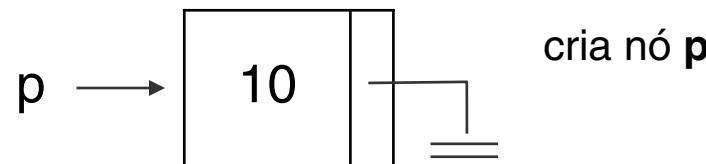
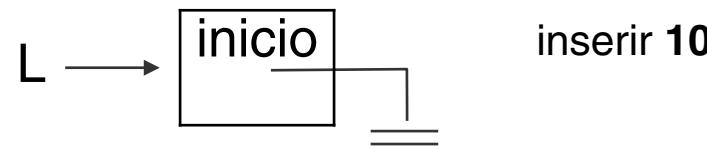
cria nó p

o **próximo** de p aponta para onde o **inicio** de L aponta

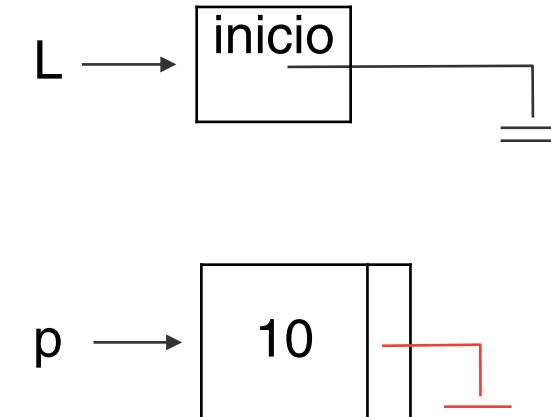


Inserção no início (cabeça) da lista

- **Caso 1:** Lista está vazia

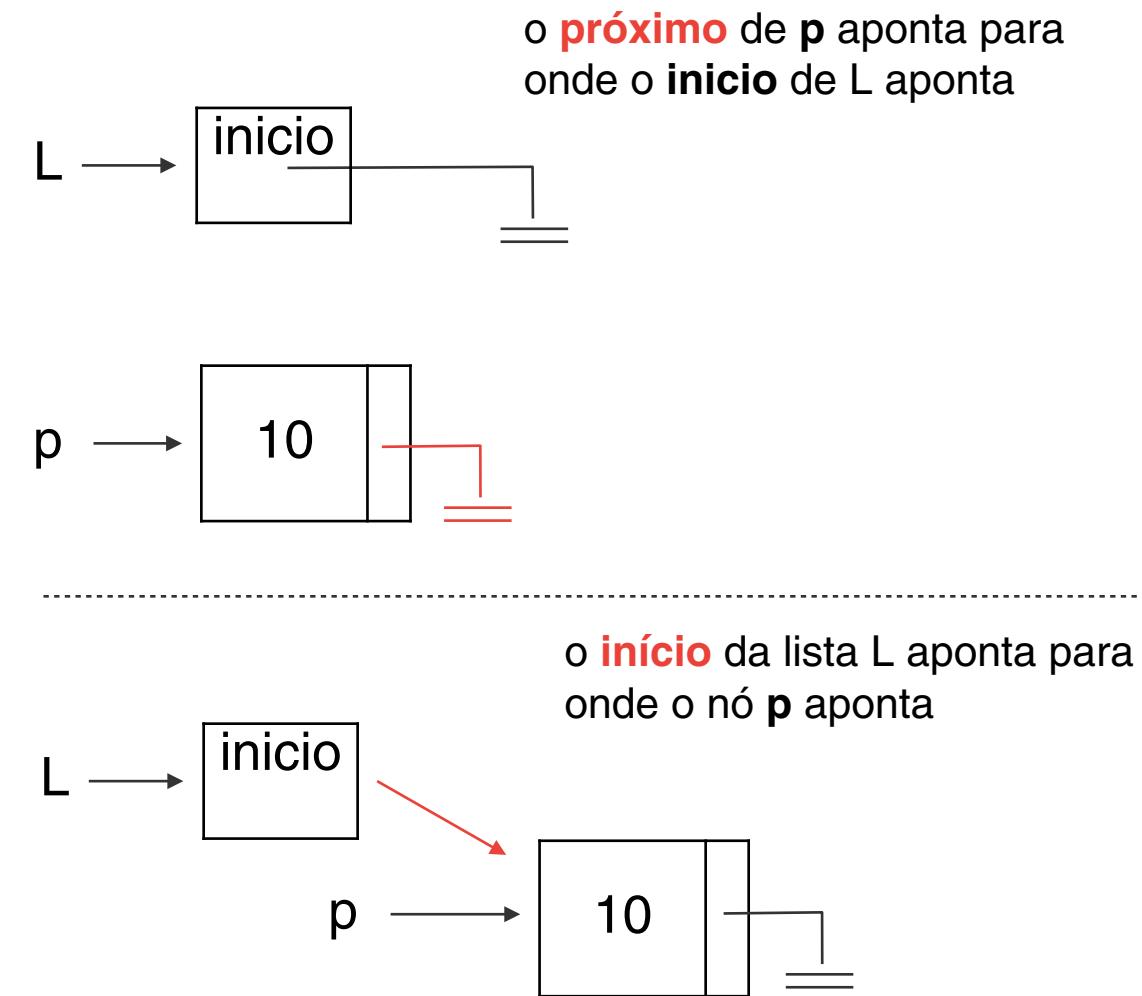
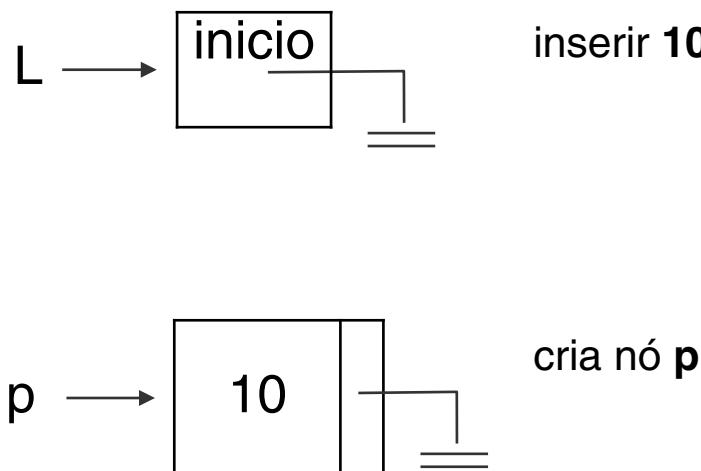


o **próximo** de **p** aponta para onde o **inicio** de **L** aponta



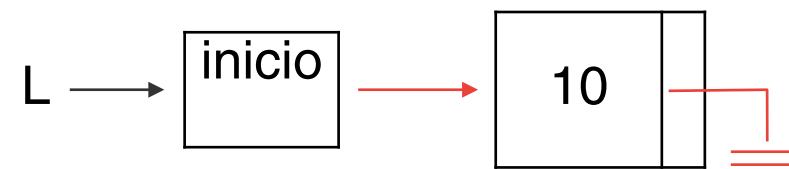
Inserção no início (cabeça) da lista

- Caso 1: Lista está vazia



Inserção no início (cabeça) da lista

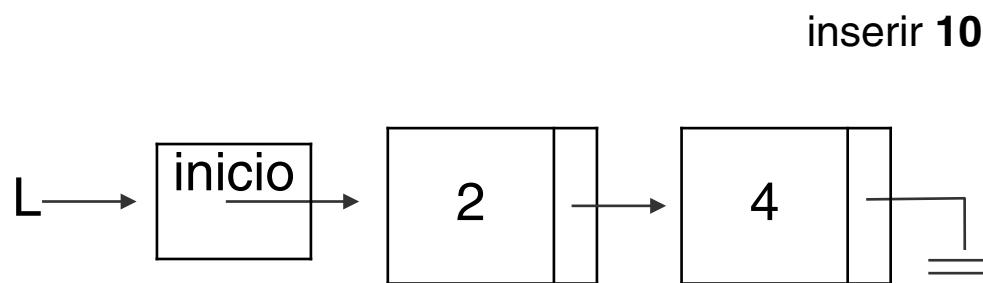
- **Caso 1:** Lista está vazia



configuração final da lista
após a inserção

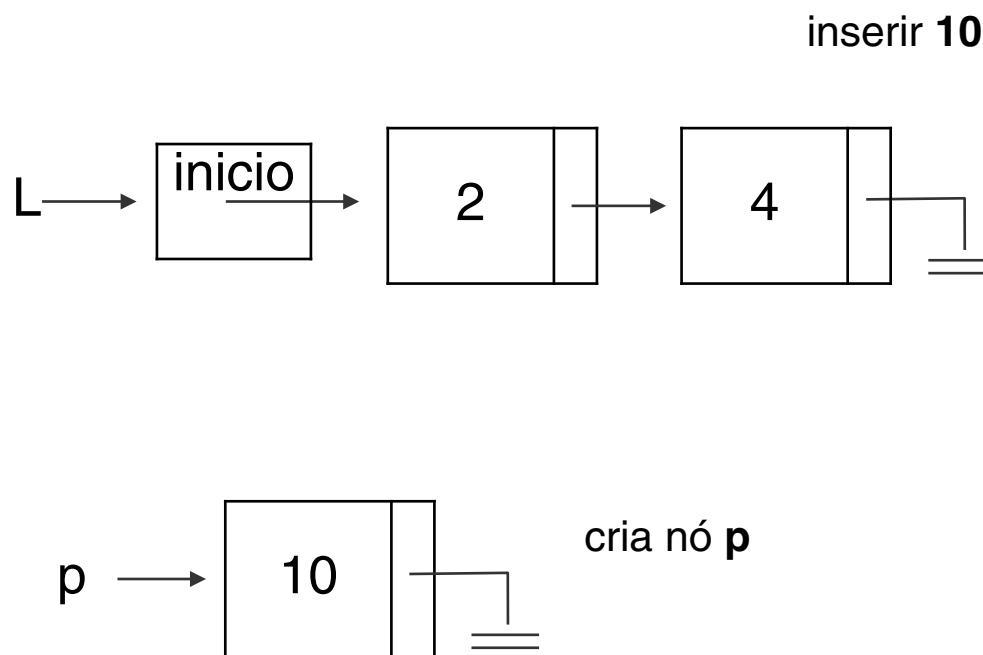
Inserção no início (cabeça) da lista

- **Caso 2:** Lista possui elementos



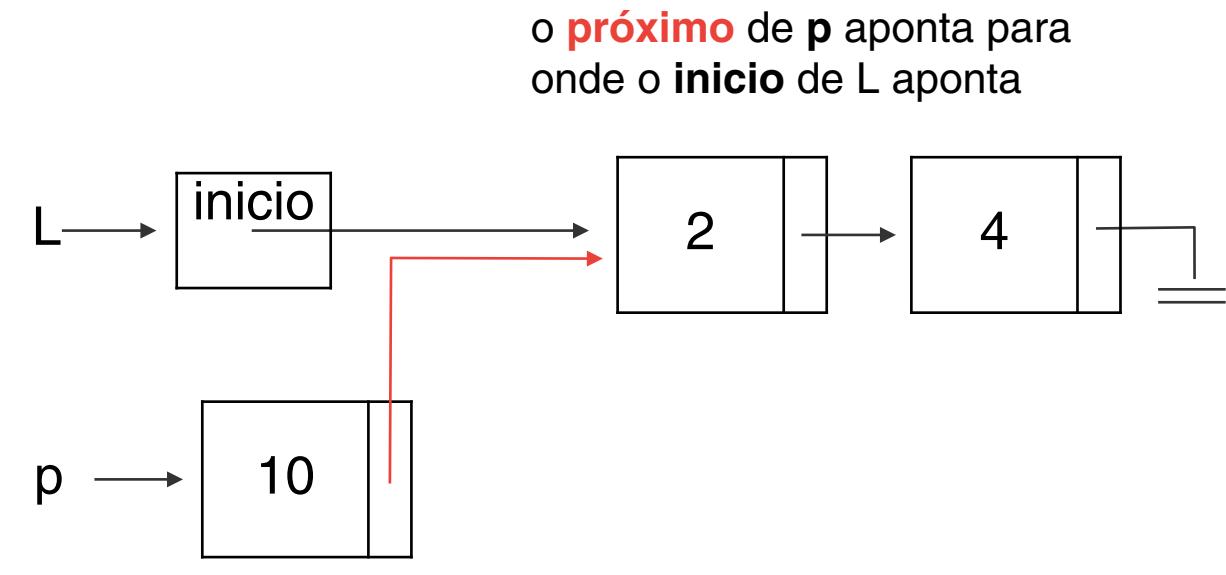
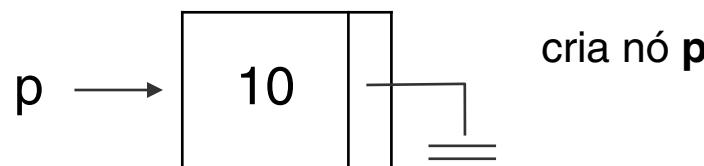
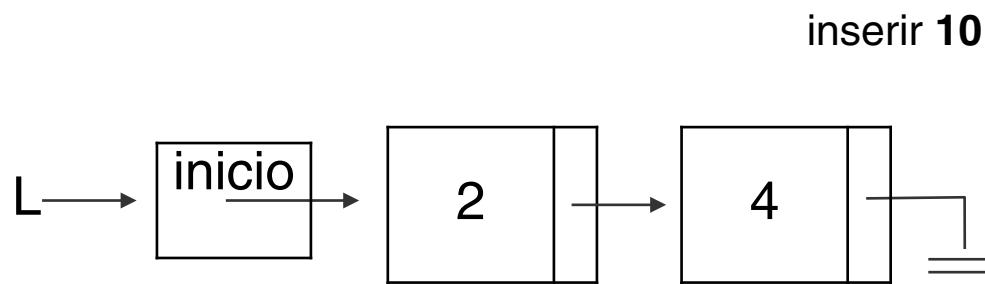
Inserção no início (cabeça) da lista

- **Caso 2:** Lista possui elementos



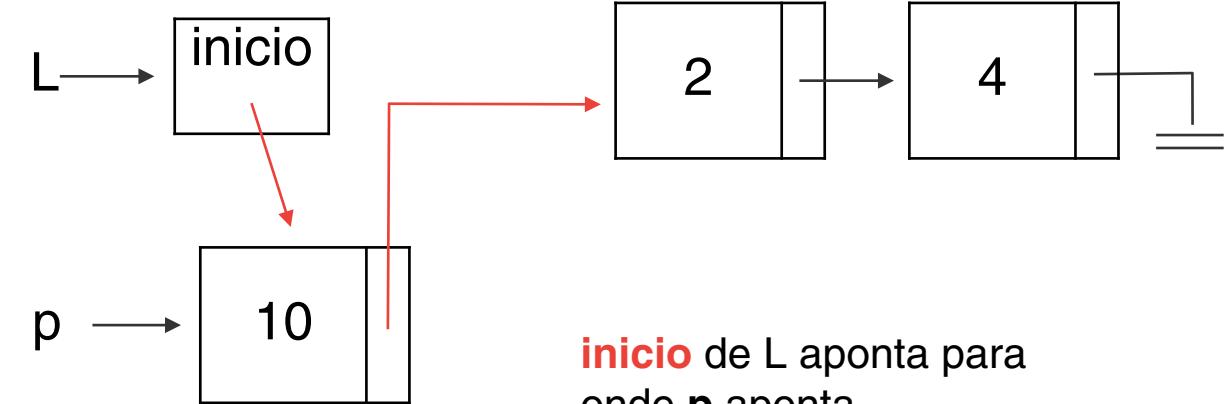
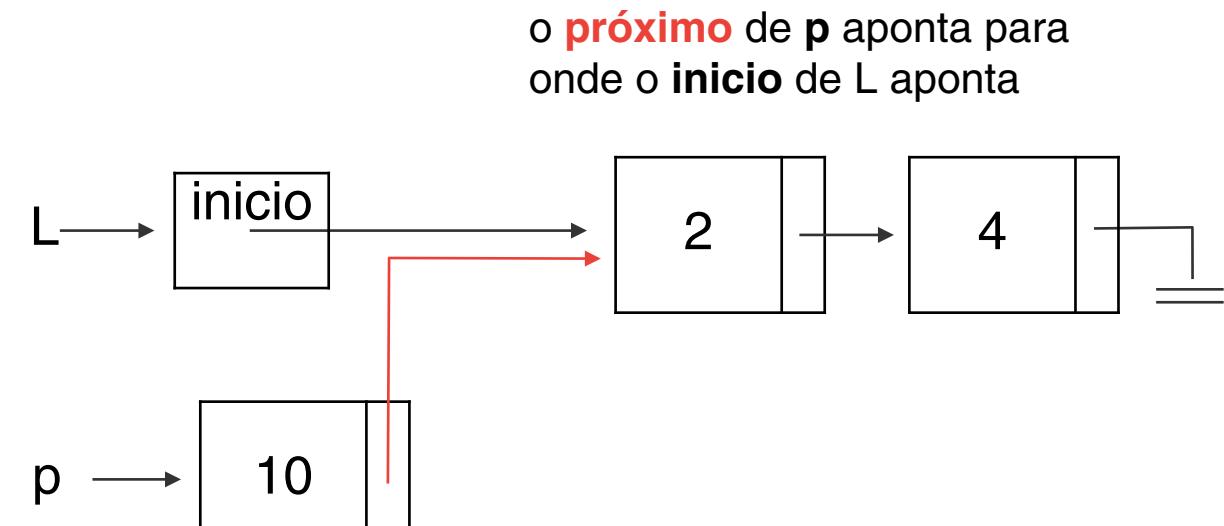
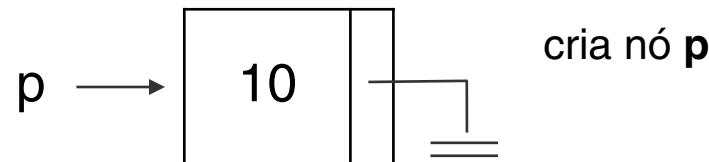
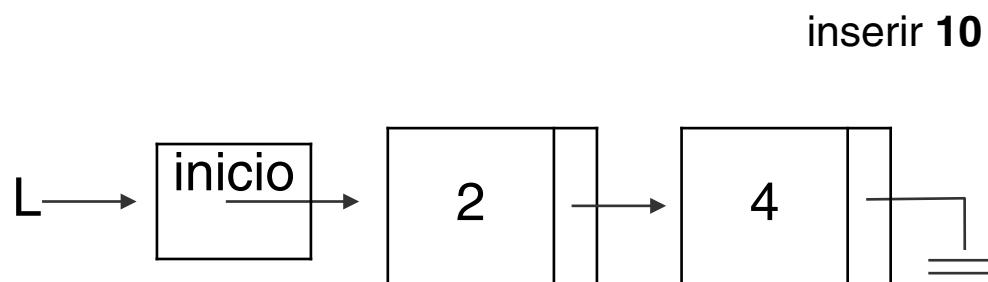
Inserção no início (cabeça) da lista

- **Caso 2:** Lista possui elementos



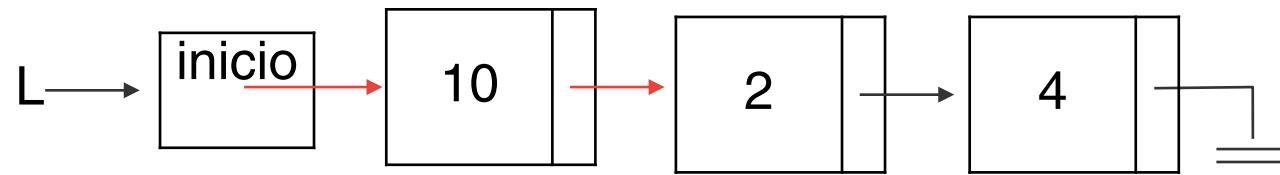
Inserção no início (cabeça) da lista

- **Caso 2:** Lista possui elementos



Inserção no início (cabeça) da lista

- **Caso 2:** Lista possui elementos



configuração final da lista
após a inserção

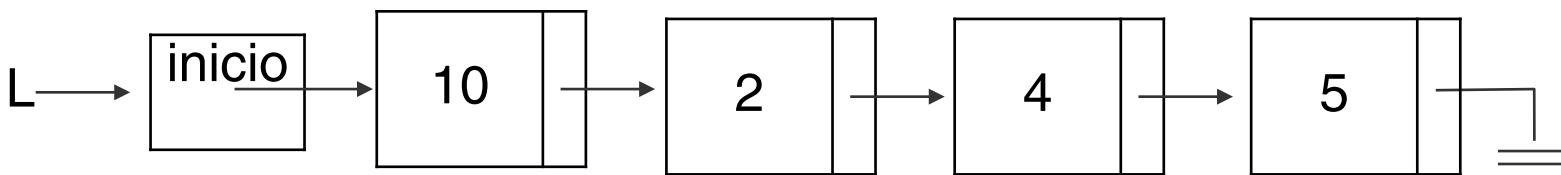
Listas Encadeadas Simples

- Diversos tipos de **operações**:
 - Inserção na cabeça (início) da lista;
 - Impressão dos Elementos da Lista
 - Inserção na cauda (fim) da lista;
 - Remover elementos da lista;
 - Contar o número de elementos da Lista;
 - Verificar se a lista está vazia e retornar verdadeiro/falso
 - Concatenar de duas listas;
 - Copiar uma lista;
 - Ordenação de uma lista;
 - Buscar um dado elemento na lista e retornar seu ponteiro;
 - Inserção ordenada;

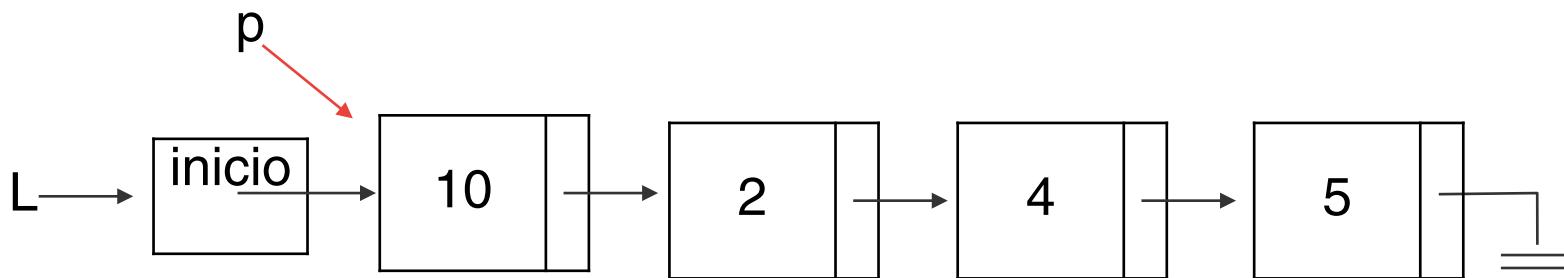
Lista Encadeada Simples

- Diversos tipos de **operações**:
 - Inserção na cabeça (íncio) da lista;
 - **Impressão dos Elementos da Lista**
 - Inserção na cauda (fim) da lista;
 - Remover elementos da lista;
 - Contar o número de elementos da Lista;
 - Verificar se a lista está vazia e retornar verdadeiro/falso
 - Concatenar de duas listas;
 - Copiar uma lista;
 - Ordenação de uma lista;
 - Buscar um dado elemento na lista e retornar seu ponteiro;
 - Inserção ordenada;

Impressão dos Elementos da Lista

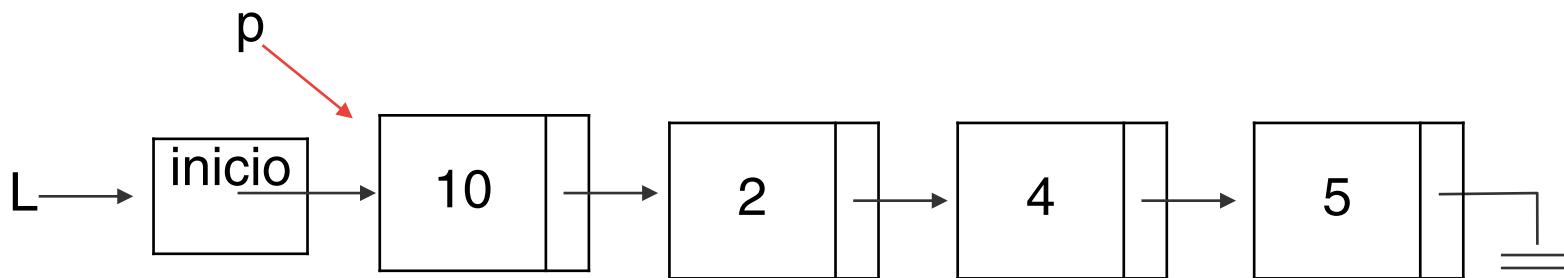


Impressão dos Elementos da Lista



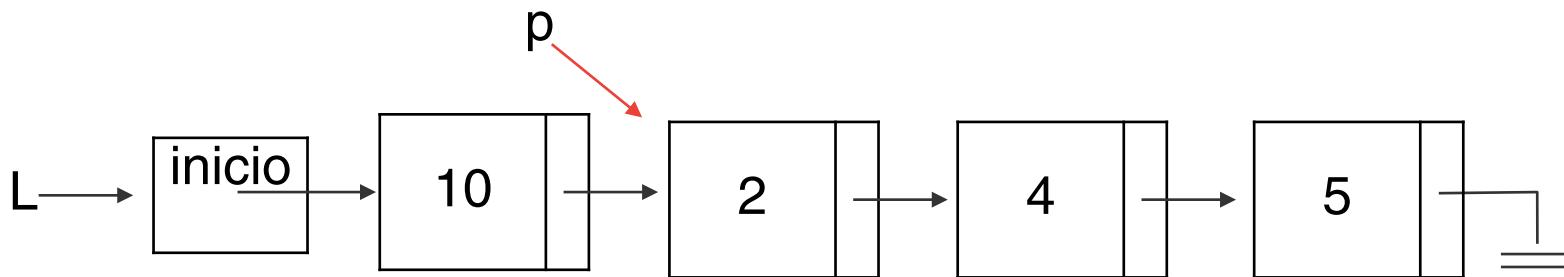
crie um ponteiro **p** e faça-o
apontar para o **início da lista**

Impressão dos Elementos da Lista



enquanto a lista **não chegou ao fim** (NULL),
imprima os dados apontado por **p** e
mova **p para o próximo elemento** da lista
(faça **p** apontar para seu próximo elemento)
 $p = p->prox;$

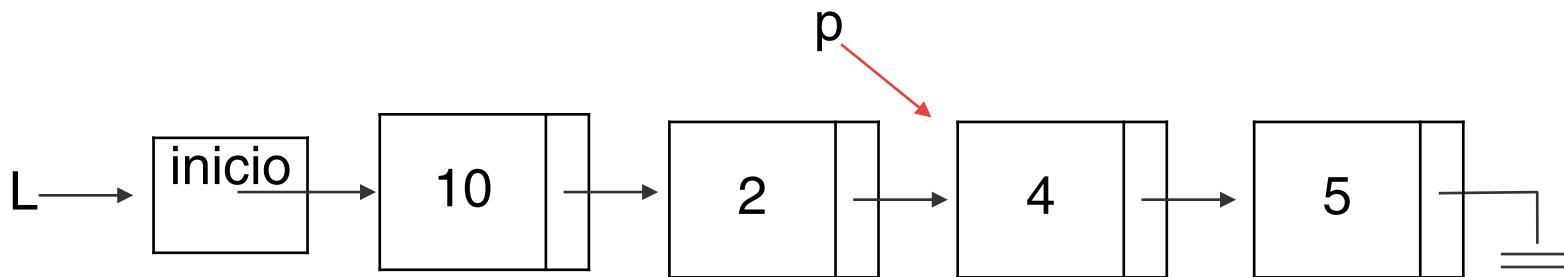
Impressão dos Elementos da Lista



enquanto a lista **não chegou ao fim** (NULL),
imprima os dados apontado por **p** e
mova **p para o próximo elemento** da lista
(faça **p** apontar para seu próximo elemento)

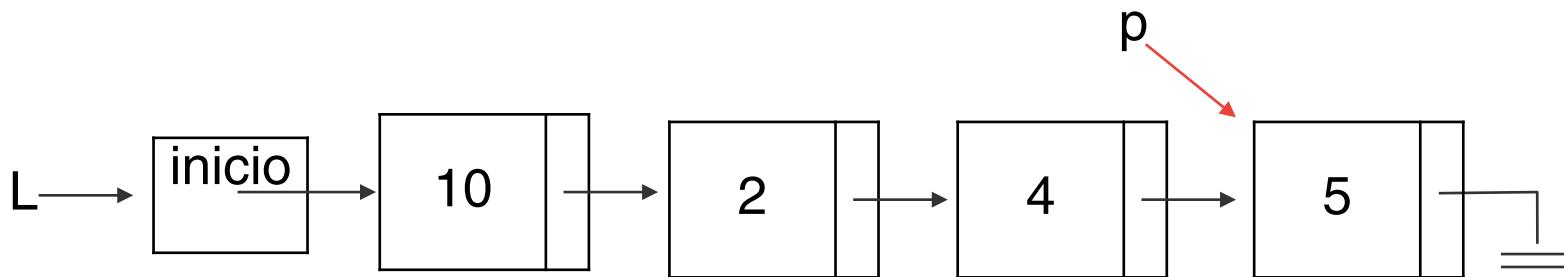
$p = p->prox;$

Impressão dos Elementos da Lista



enquanto a lista **não chegou ao fim** (NULL),
imprima os dados apontado por **p** e
mova **p para o próximo elemento** da lista
(faça **p** apontar para seu próximo elemento)
 $p = p->prox;$

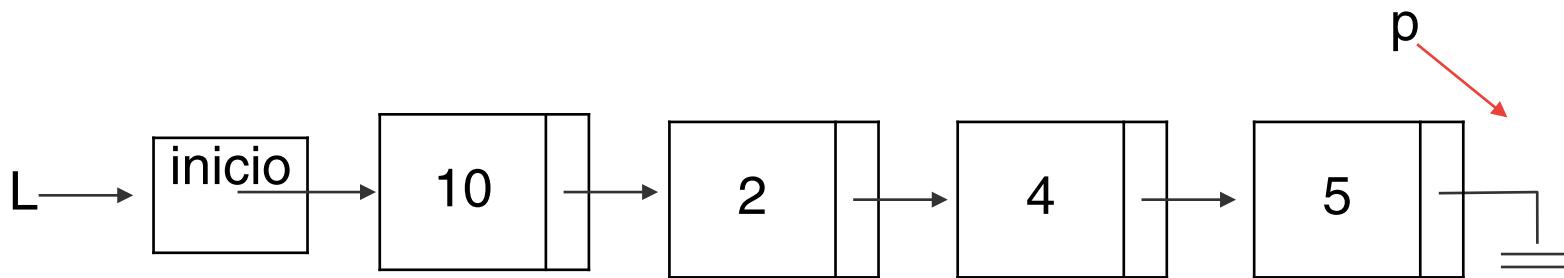
Impressão dos Elementos da Lista



enquanto a lista **não chegou ao fim** (NULL),
imprima os dados apontado por **p** e
mova **p para o próximo elemento** da lista
(faça **p** apontar para seu próximo elemento)

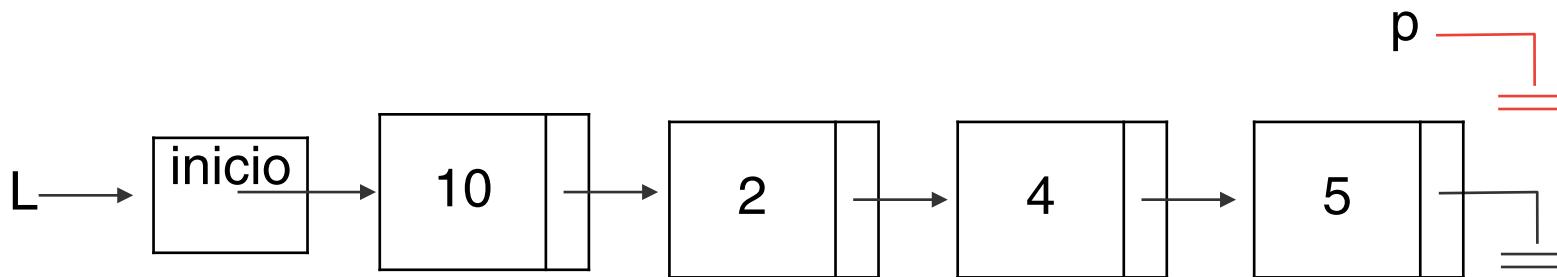
$p = p->prox;$

Impressão dos Elementos da Lista



enquanto a lista **não chegou ao fim** (NULL),
imprima os dados apontado por **p** e
mova **p para o próximo elemento** da lista
(faça **p** apontar para seu próximo elemento)
$$p = p->prox;$$

Impressão dos Elementos da Lista



enquanto a lista **não chegou ao fim** (NULL),
imprima os dados apontado por p e
mova p para o **próximo elemento** da lista
(faça p apontar para seu próximo elemento)

$p = p->prox;$

Lista Encadeada Simples

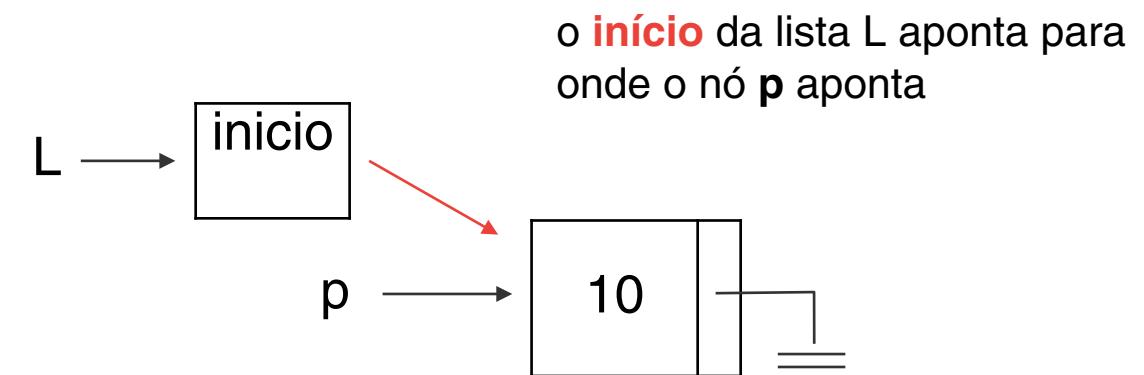
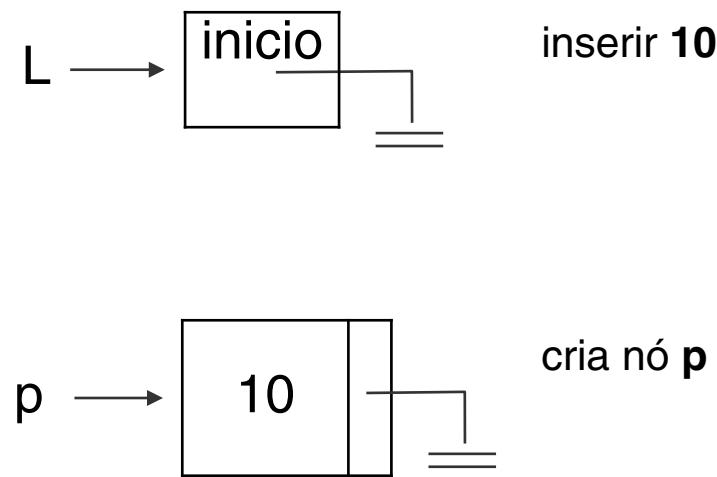
- Diversos tipos de **operações**:
 - Inserção na cabeça (íncio) da lista;
 - **Impressão dos Elementos da Lista**
 - Inserção na cauda (fim) da lista;
 - Remover elementos da lista;
 - Contar o número de elementos da Lista;
 - Verificar se a lista está vazia e retornar verdadeiro/falso
 - Concatenar de duas listas;
 - Copiar uma lista;
 - Ordenação de uma lista;
 - Buscar um dado elemento na lista e retornar seu ponteiro;
 - Inserção ordenada;

Listas Encadeadas Simples

- Diversos tipos de **operações**:
 - Inserção na cabeça (índice 0) da lista;
 - Impressão dos Elementos da Lista
 - **Inserção na cauda (fim) da lista;**
 - Remover elementos da lista;
 - Contar o número de elementos da Lista;
 - Verificar se a lista está vazia e retornar verdadeiro/falso
 - Concatenar de duas listas;
 - Copiar uma lista;
 - Ordenação de uma lista;
 - Buscar um dado elemento na lista e retornar seu ponteiro;
 - Inserção ordenada;

Inserção no fim (cauda) da lista

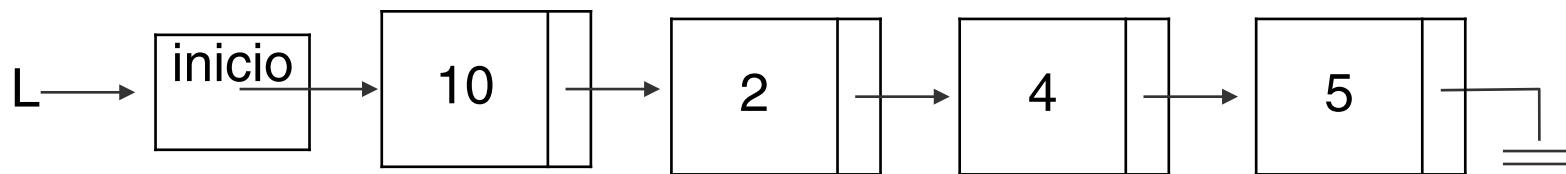
- Caso 1: Lista está vazia



Inserção no fim (cauda) da lista

- **Caso 2:** Lista possui elementos

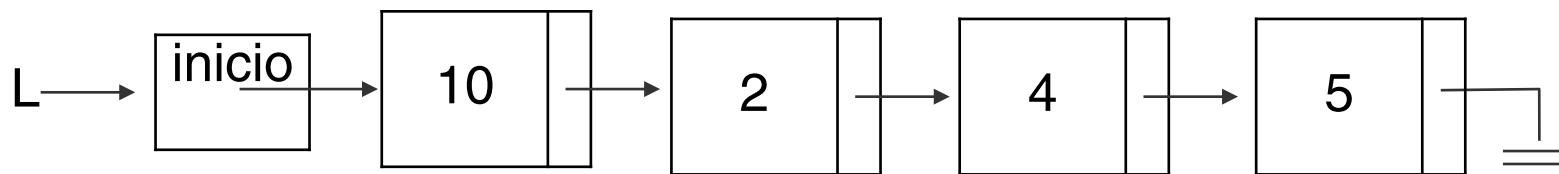
inserir 7



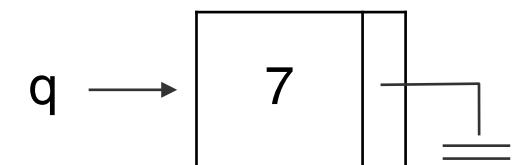
Inserção no fim (cauda) da lista

- **Caso 2:** Lista possui elementos

inserir 7



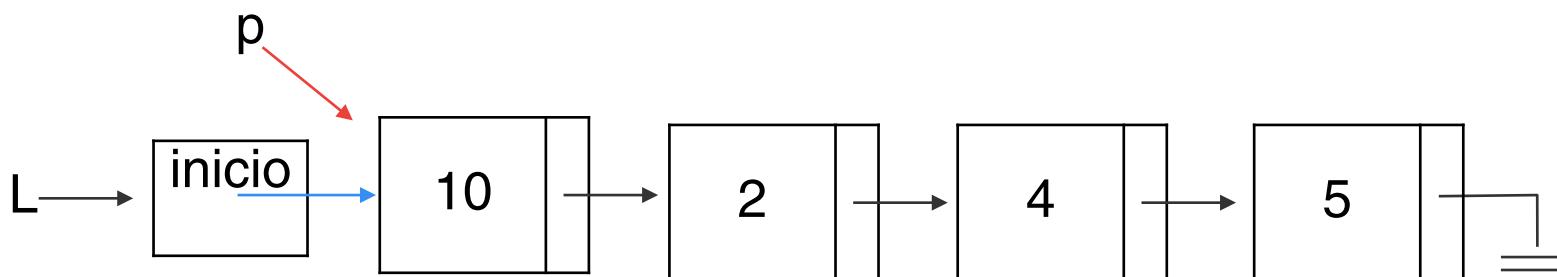
cria nó q



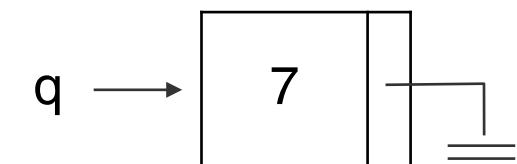
Inserção no fim (cauda) da lista

- **Caso 2:** Lista possui elementos

inserir 7



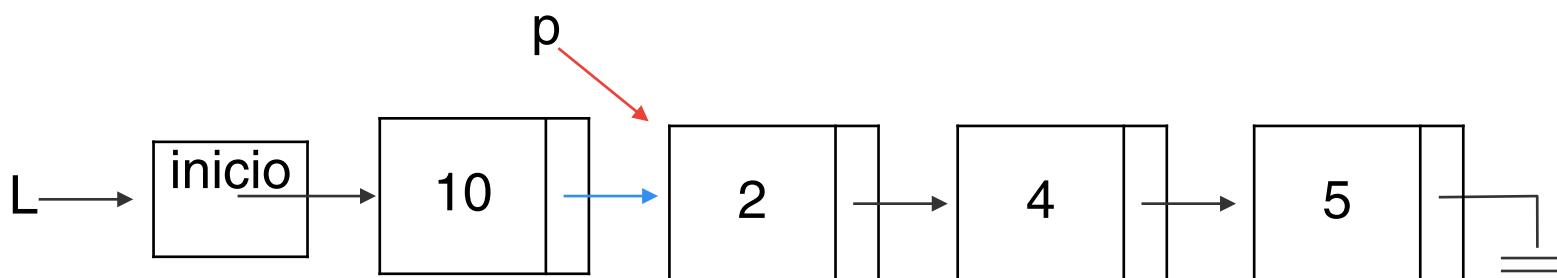
O ponteiro **p** aponta, inicialmente, para o **índio nó da lista**.
Queremos posicionar **p** no último nó para então inserir
o novo nó depois (em $p \rightarrow \text{prox}$)



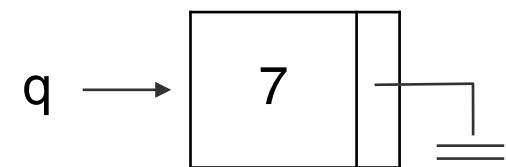
Inserção no fim (cauda) da lista

- **Caso 2:** Lista possui elementos

inserir 7



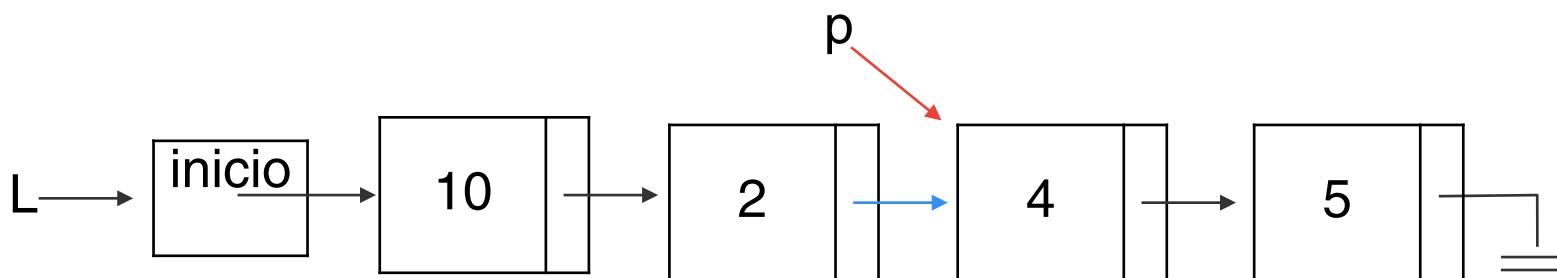
Enquanto o próximo nó de **p** não é NULL,
ou seja, enquanto **p** não aponta para o **último nó da lista**,
mova **p** para o **próximo nó** da lista



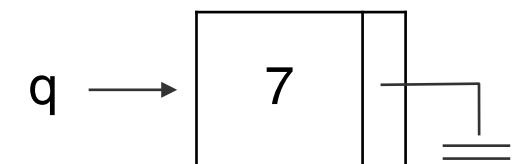
Inserção no fim (cauda) da lista

- **Caso 2:** Lista possui elementos

inserir 7



Enquanto o próximo nó de **p** não é NULL,
ou seja, enquanto **p** não aponta para o **último nó da lista**,
mova **p** para o **próximo nó** da lista

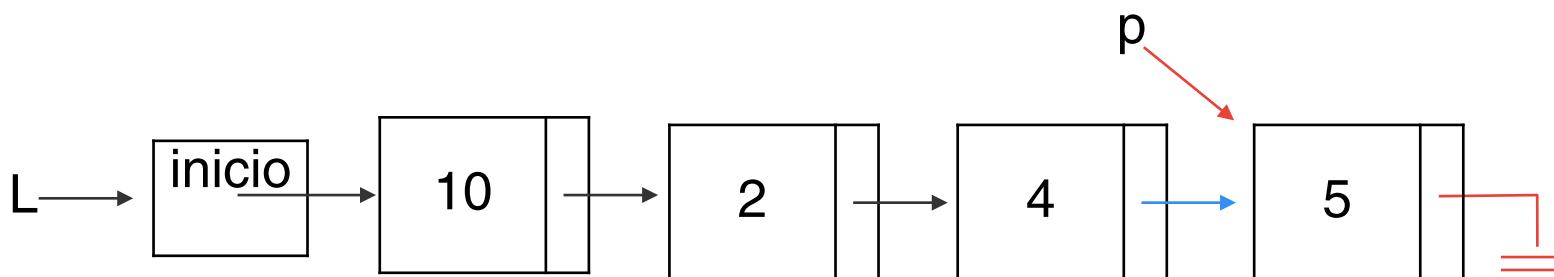


Inserção no fim (cauda) da lista

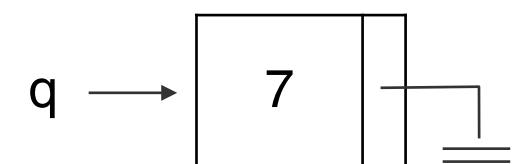
- **Caso 2:** Lista possui elementos

o próximo nó de **p** é **NULL**,
chegamos no nó final

inserir 7



Enquanto o próximo nó de **p** não é **NULL**,
ou seja, enquanto **p** não aponta para o **último nó da lista**,
mova **p** para o **próximo nó** da lista

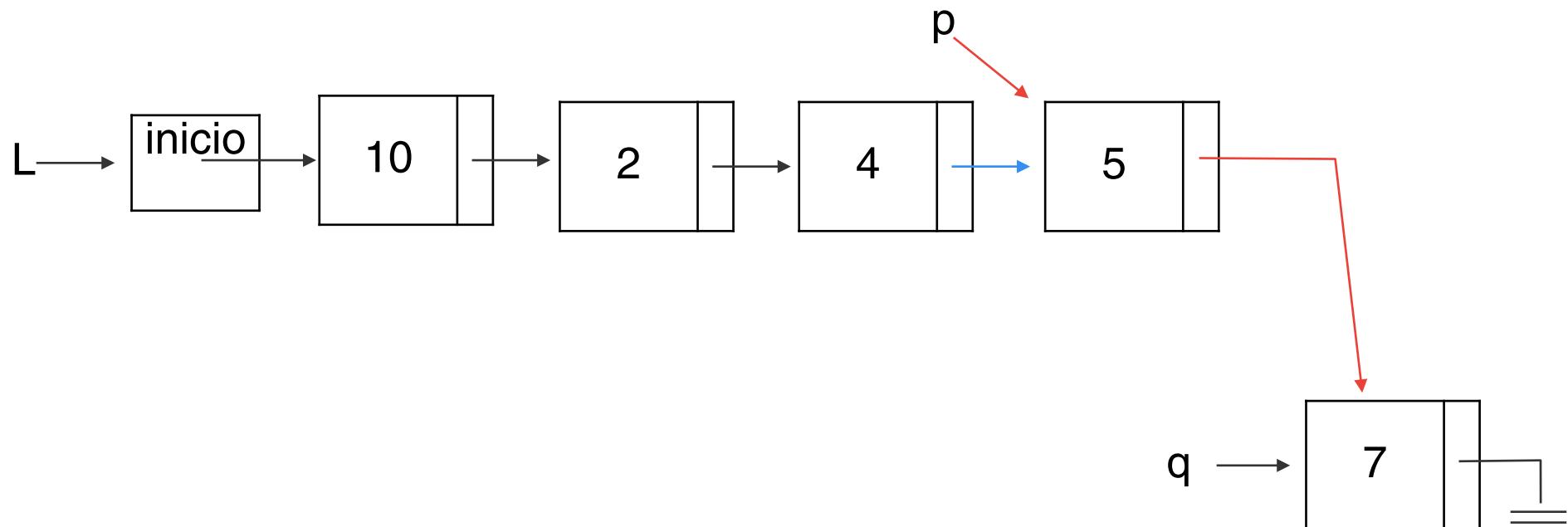


Inserção no fim (cauda) da lista

- **Caso 2:** Lista possui elementos

o próximo nó de **p** é **NULL**,
chegamos no nó final

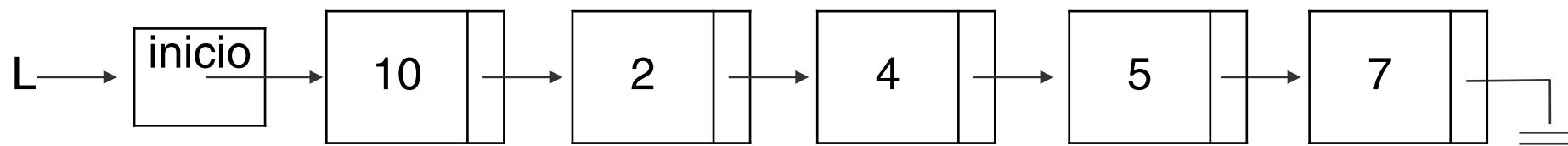
inserir 7



o **próximo** de **p** aponta para **q**

Inserção no fim (cauda) da lista

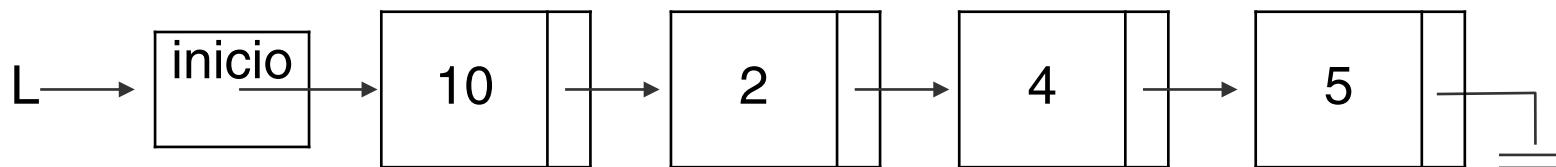
- **Caso 2:** Lista possui elementos



configuração final da lista
após a inserção

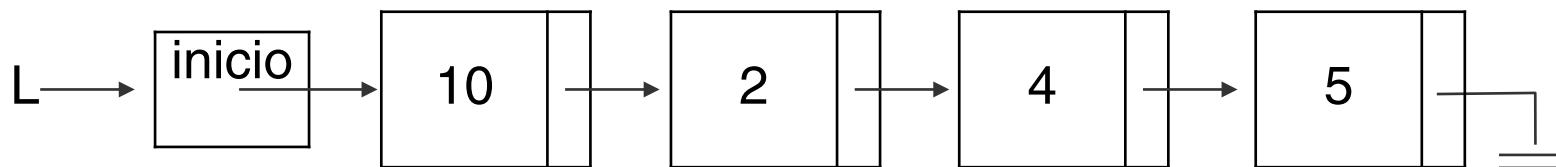
Inserção no fim (cauda) da lista

- **Caso 2:** Lista possui elementos
- Problemas?



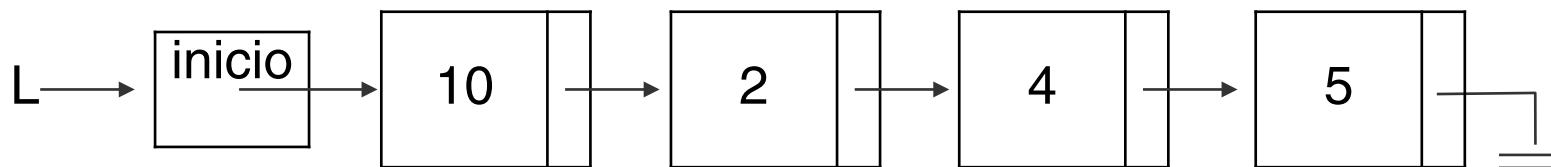
Inserção no fim (cauda) da lista

- **Caso 2:** Lista possui elementos
- **Problemas?**
 - Precisamos percorrer todos os nós da lista até chegar no último nó —> **ineficiente**



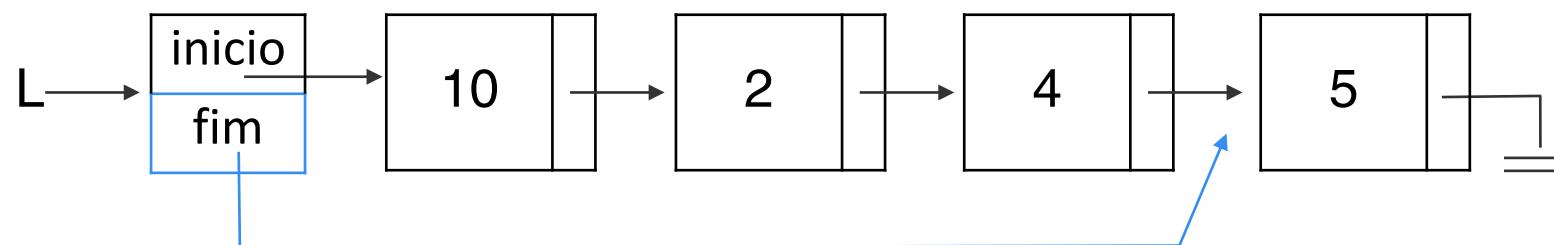
Inserção no fim (cauda) da lista

- **Caso 2:** Lista possui elementos
- **Problemas?**
 - Precisamos percorrer todos os nós da lista até chegar no último nó —> **ineficiente**
- **Solução?**



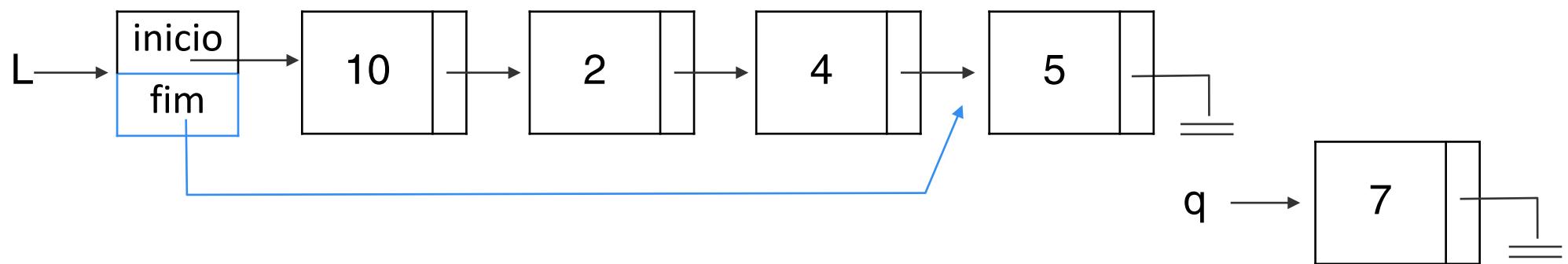
Inserção no fim (cauda) da lista

- **Caso 2:** Lista possui elementos
- **Problemas?**
 - Precisamos percorrer todos os nós da lista até chegar no último nó —> **ineficiente**
- **Solução?**
 - Adicionamos, na struct de Lista, um ponteiro **fim** para o **nó final da lista**;



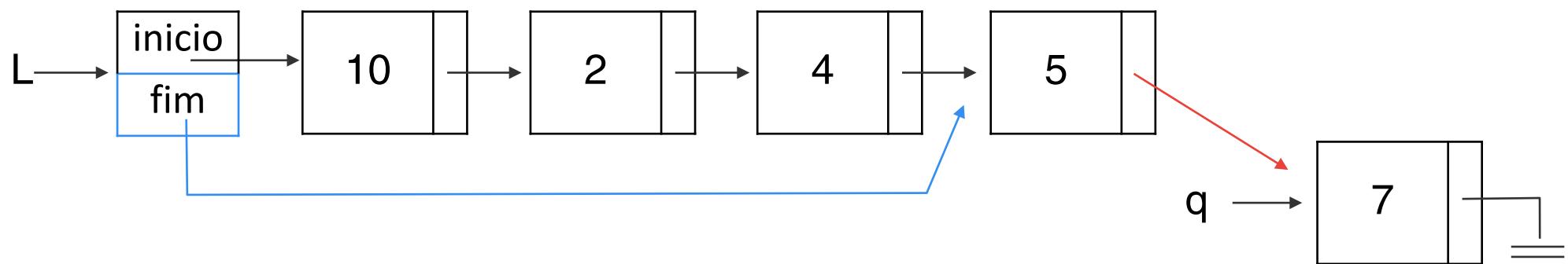
Inserção no fim (cauda) da lista

- **Caso 2:** Lista possui elementos
- **Problemas?**
 - Precisamos percorrer todos os nós da lista até chegar no último nó —> **ineficiente**
- **Solução?**
 - Adicionamos, na struct de Lista, um ponteiro **fim** para o **nó final da lista**;
 - Assim, conseguimos adicionar o novo nó diretamente no final da lista;



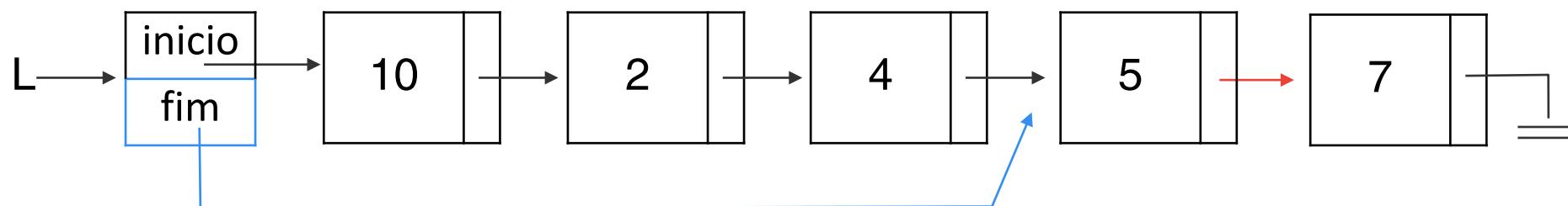
Inserção no fim (cauda) da lista

- **Caso 2:** Lista possui elementos
- **Problemas?**
 - Precisamos percorrer todos os nós da lista até chegar no último nó —> **ineficiente**
- **Solução?**
 - Adicionamos, na struct de Lista, um ponteiro **fim** para o **nó final da lista**;
 - Assim, conseguimos adicionar o novo nó diretamente no final da lista;



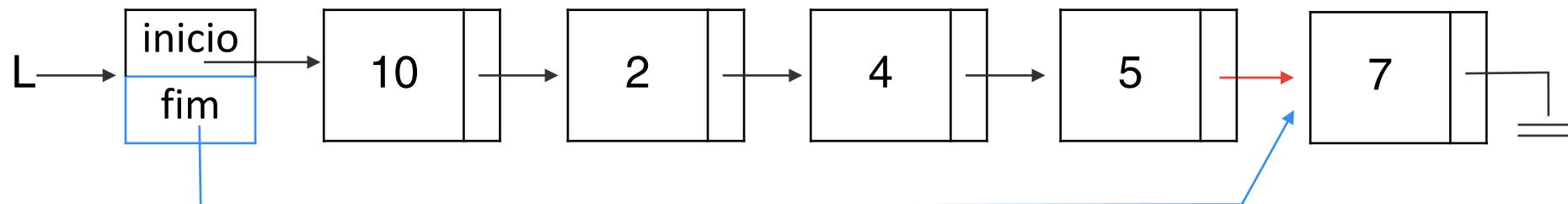
Inserção no fim (cauda) da lista

- **Caso 2:** Lista possui elementos
- **Problemas?**
 - Precisamos percorrer todos os nós da lista até chegar no último nó —> **ineficiente**
- **Solução?**
 - Adicionamos, na struct de Lista, um ponteiro **fim** para o **nó final da lista**;
 - Assim, conseguimos adicionar o novo nó diretamente no final da lista;



Inserção no fim (cauda) da lista

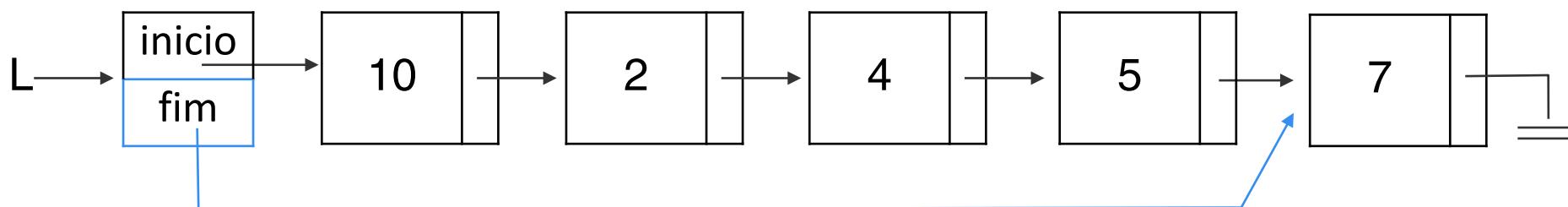
- **Caso 2:** Lista possui elementos
- **Problemas?**
 - Precisamos percorrer todos os nós da lista até chegar no último nó —> **ineficiente**
- **Solução?**
 - Adicionamos, na struct de Lista, um ponteiro **fim** para o **nó final da lista**;
 - Assim, conseguimos adicionar o novo nó diretamente no final da lista;
 - Teremos que adaptar o **caso 1** da função de **inserção no início** da lista;



Inserção no fim (cauda) da lista

```
typedef struct _no {  
    int val;  
    struct _no *prox;  
} No;
```

```
typedef struct _lista {  
    No *inicio;  
    No *fim;  
} Lista;
```



Listas Encadeadas Simples

- Diversos tipos de **operações**:
 - Inserção na cabeça (índice 0) da lista;
 - Impressão dos Elementos da Lista
 - **Inserção na cauda (fim) da lista;**
 - Remover elementos da lista;
 - Contar o número de elementos da Lista;
 - Verificar se a lista está vazia e retornar verdadeiro/falso
 - Concatenar de duas listas;
 - Copiar uma lista;
 - Ordenação de uma lista;
 - Buscar um dado elemento na lista e retornar seu ponteiro;
 - Inserção ordenada;

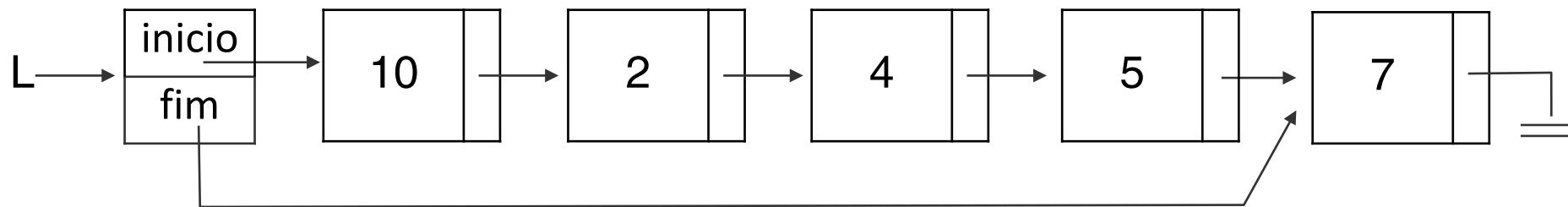
Lista Encadeada Simples

- Diversos tipos de **operações**:
 - Inserção na cabeça (íncio) da lista;
 - Impressão dos Elementos da Lista
 - Inserção na cauda (fim) da lista;
 - **Remover elementos da lista**;
 - Contar o número de elementos da Lista;
 - Verificar se a lista está vazia e retornar verdadeiro/falso
 - Concatenar de duas listas;
 - Copiar uma lista;
 - Ordenação de uma lista;
 - Buscar um dado elemento na lista e retornar seu ponteiro;
 - Inserção ordenada;

Remover um elemento da Lista

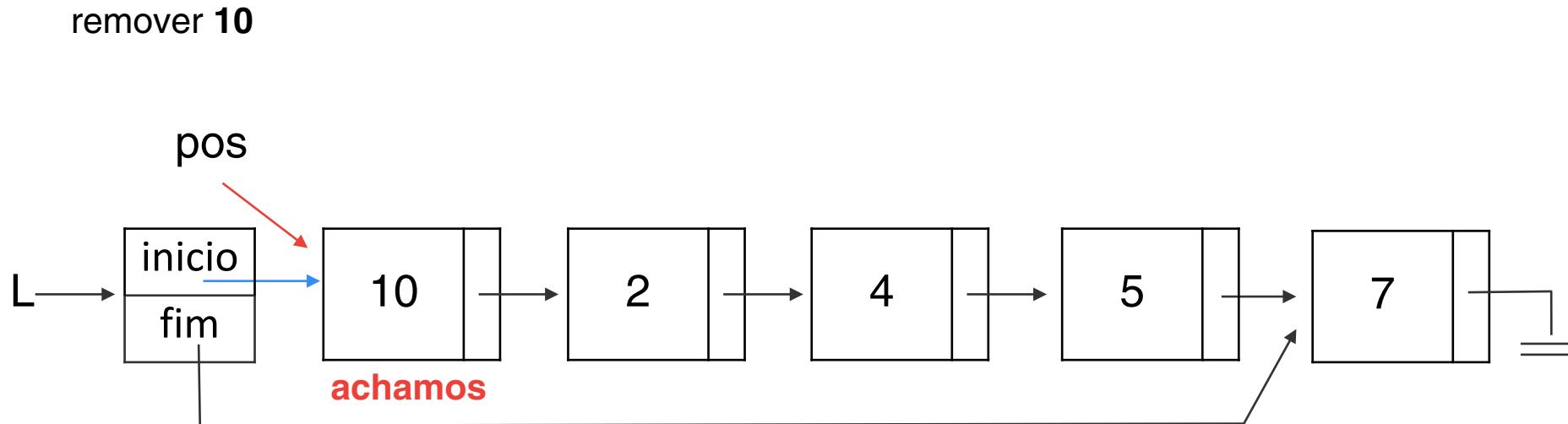
- Caso 1: Elemento está na **cabeça (início)** da lista

remover 10



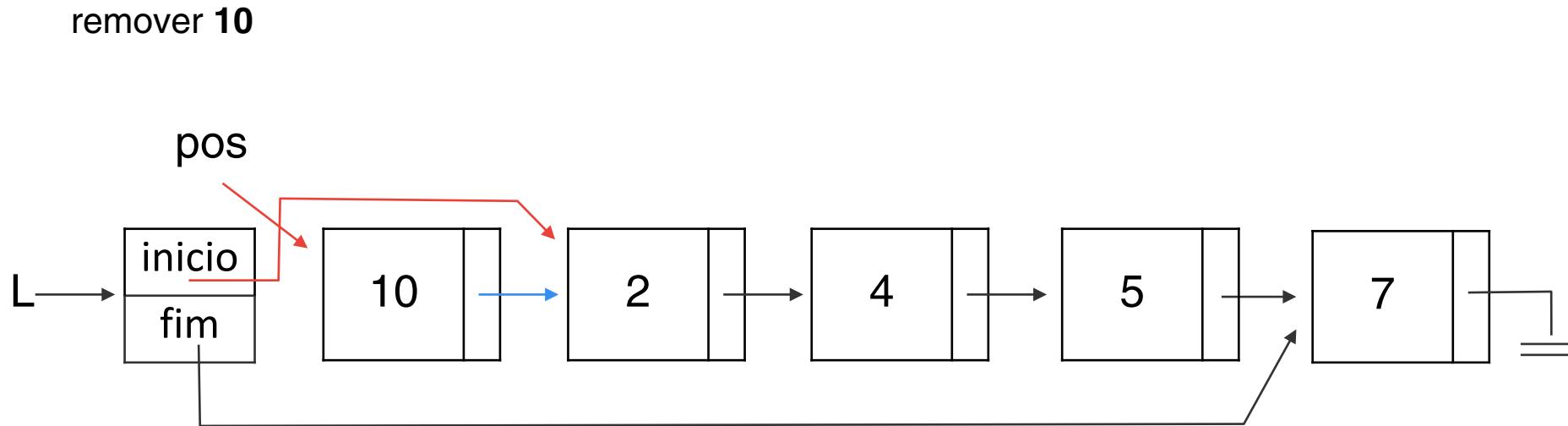
Remover um elemento da Lista

- Caso 1: Elemento está na **cabeça (início)** da lista



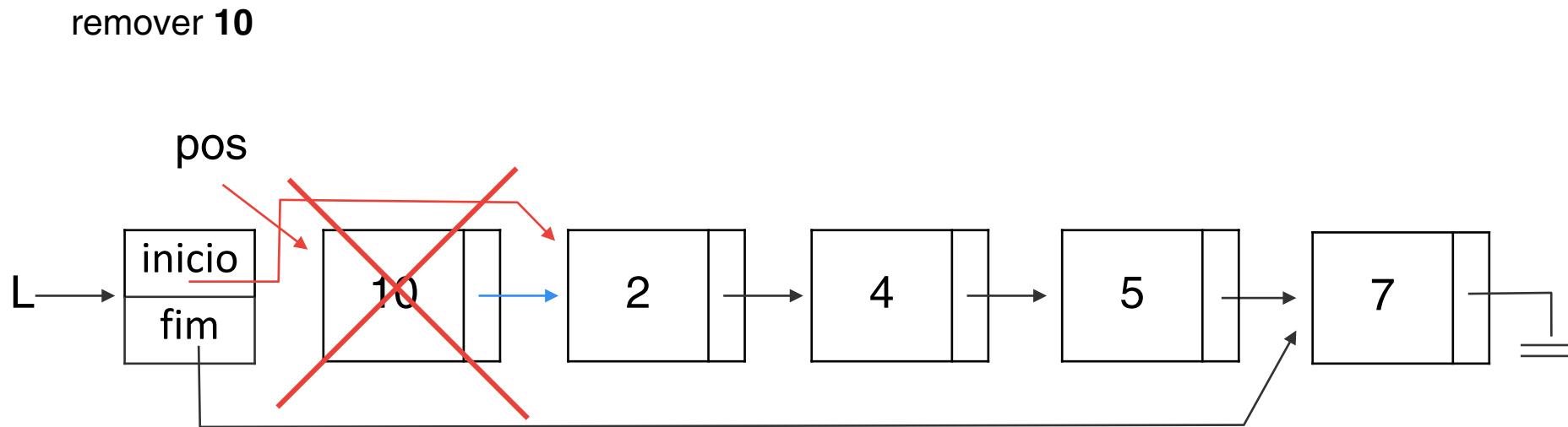
Remover um elemento da Lista

- Caso 1: Elemento está na **cabeça (início)** da lista



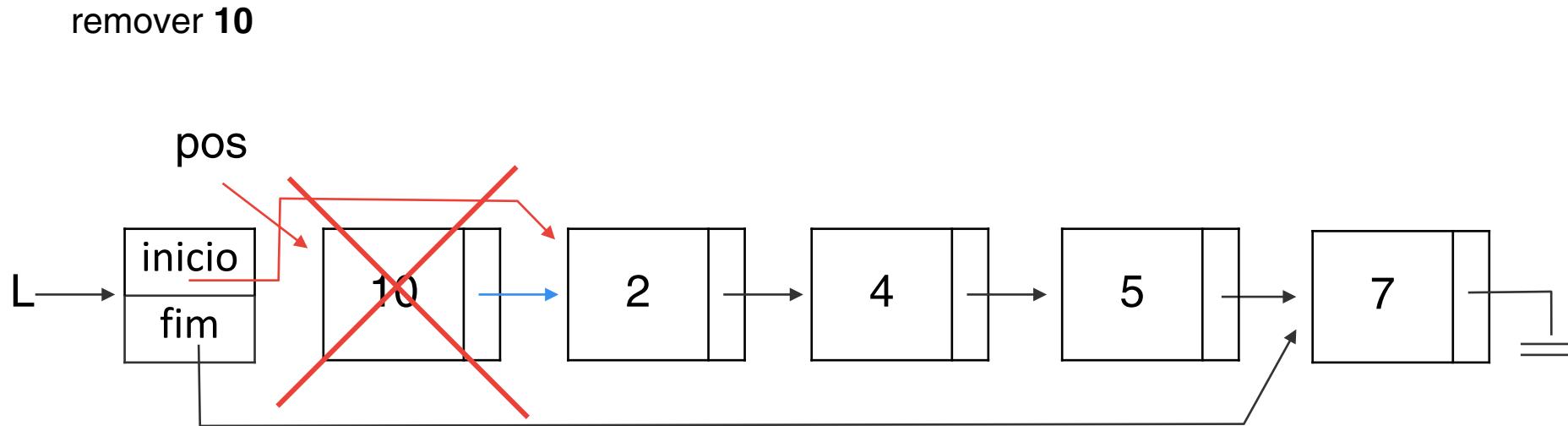
Remover um elemento da Lista

- Caso 1: Elemento está na **cabeça (início)** da lista



Remover um elemento da Lista

- Caso 1: Elemento está na **cabeça (início)** da lista

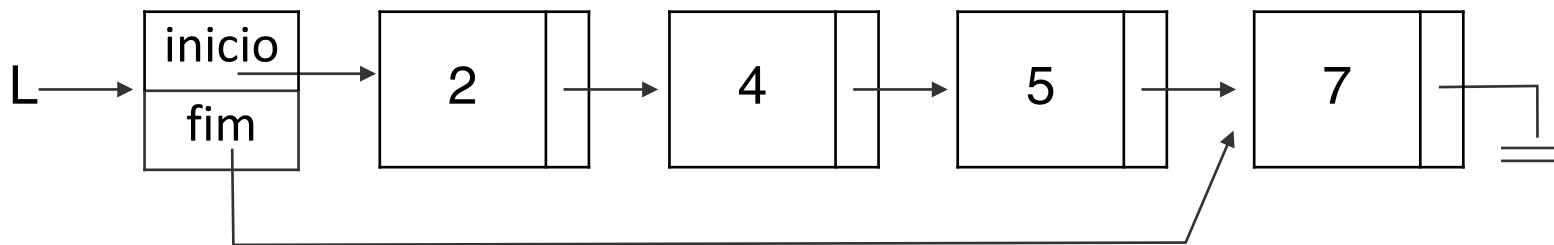


poderíamos continuar percorrendo
a lista para remover outros
possíveis nós 10

Remover um elemento da Lista

- Caso 1: Elemento está na **cabeça (início)** da lista

remover 10

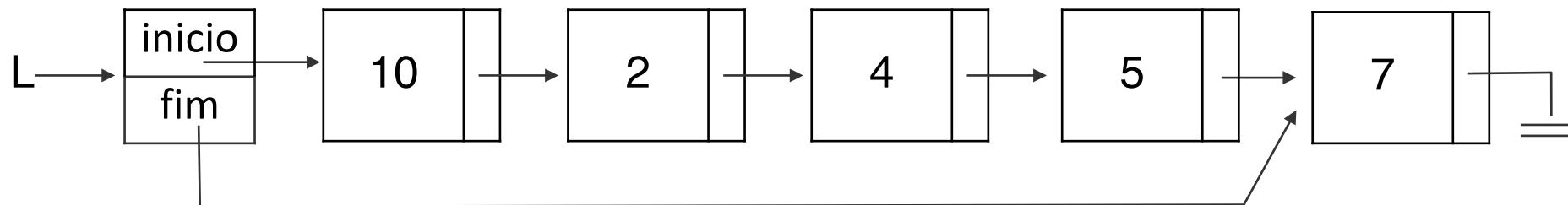


configuração final da lista
após a inserção

Remover um elemento da Lista

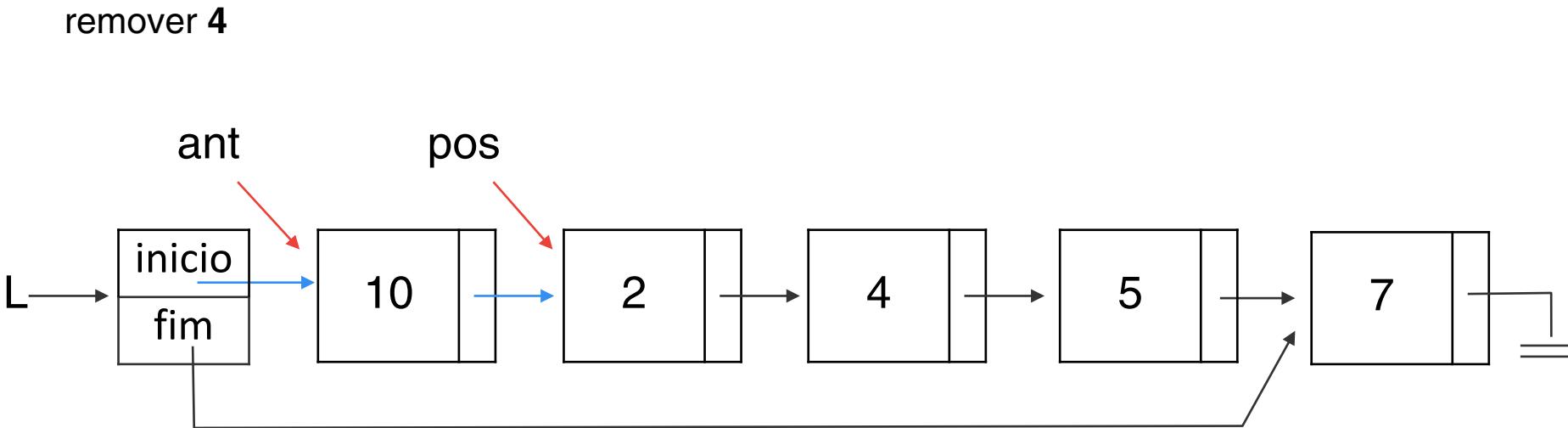
- **Caso 2:** Elemento está no **meio** da lista

remover 4



Remover um elemento da Lista

- Caso 2: Elemento está no **meio** da lista

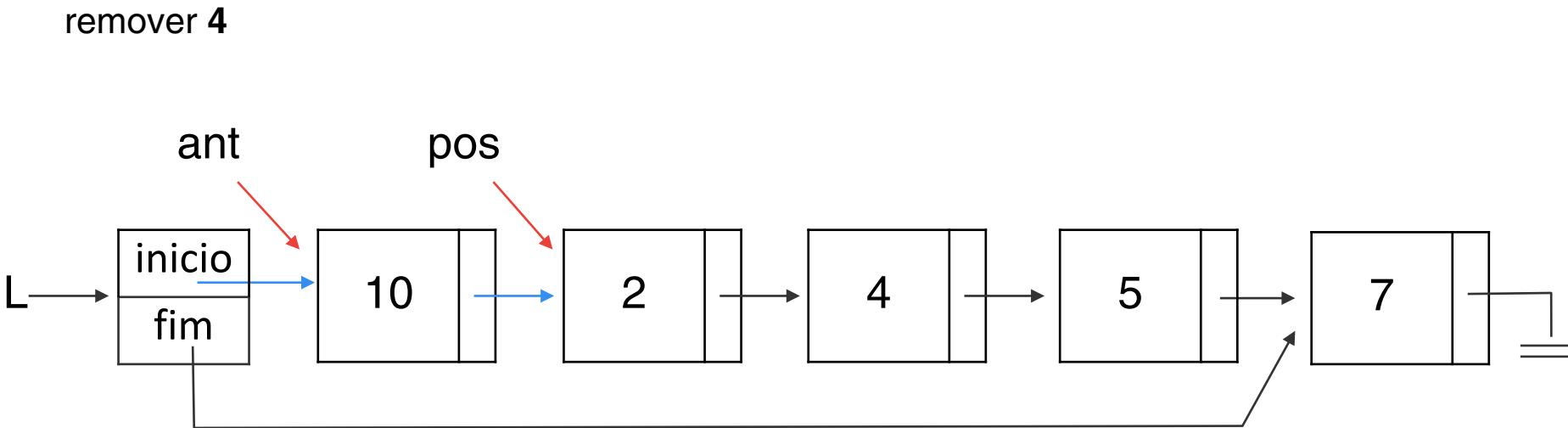


`ant = L->inicio;`
`pos = L->inicio->prox;`

além do ponteiro **pos** que checa o **elemento atual**, precisaremos de um outro ponteiro **ant** apontando para o **nó anterior**

Remover um elemento da Lista

- **Caso 2:** Elemento está no **meio** da lista

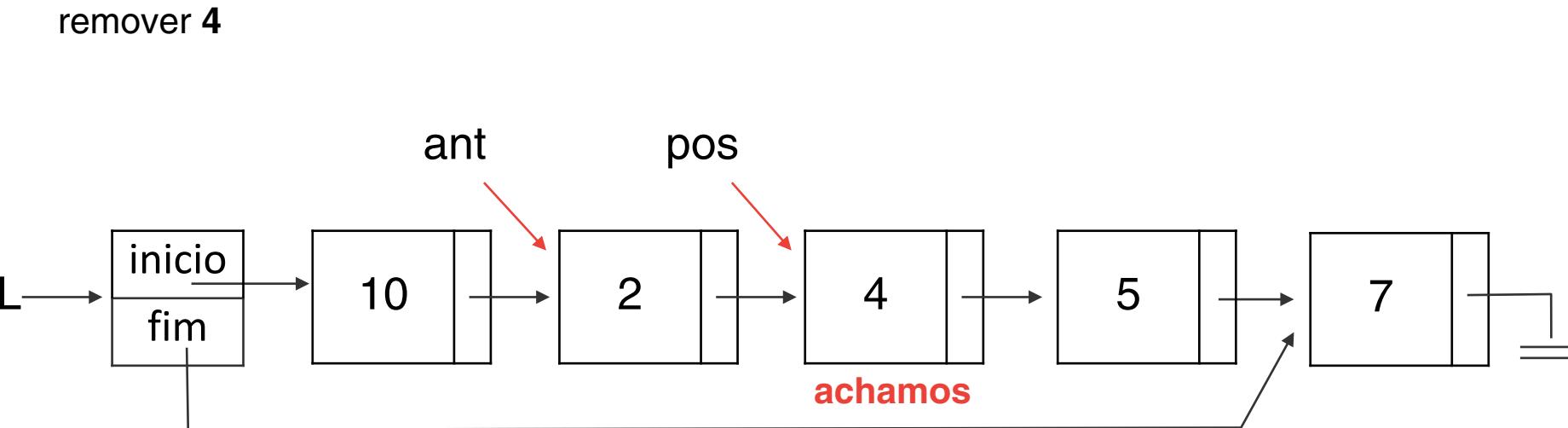


ant = L->inicio;
pos = L->inicio->prox;

enquanto **não achou** o elemento e
a lista **não chegou ao fim** ($pos == \text{NULL}$),
mova os dois ponteiros para os próximos nós

Remover um elemento da Lista

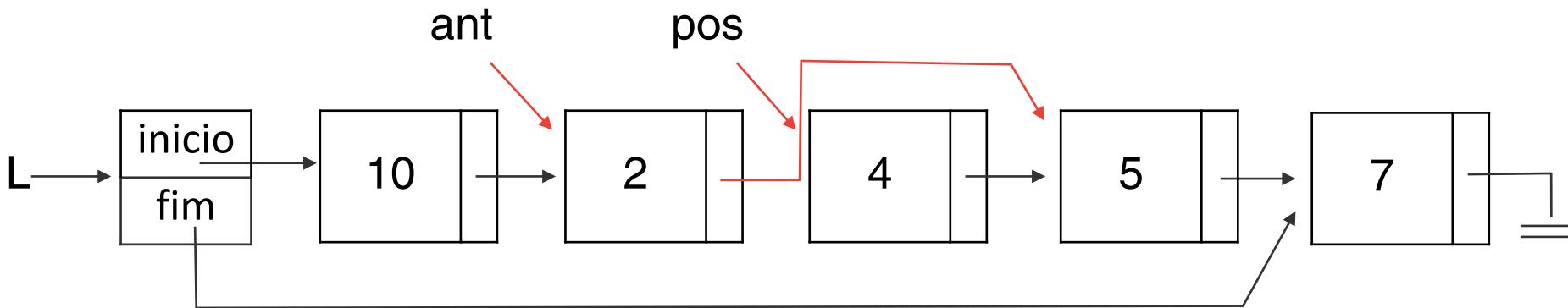
- Caso 2: Elemento está no **meio** da lista



Remover um elemento da Lista

- **Caso 2:** Elemento está no **meio** da lista

remover 4

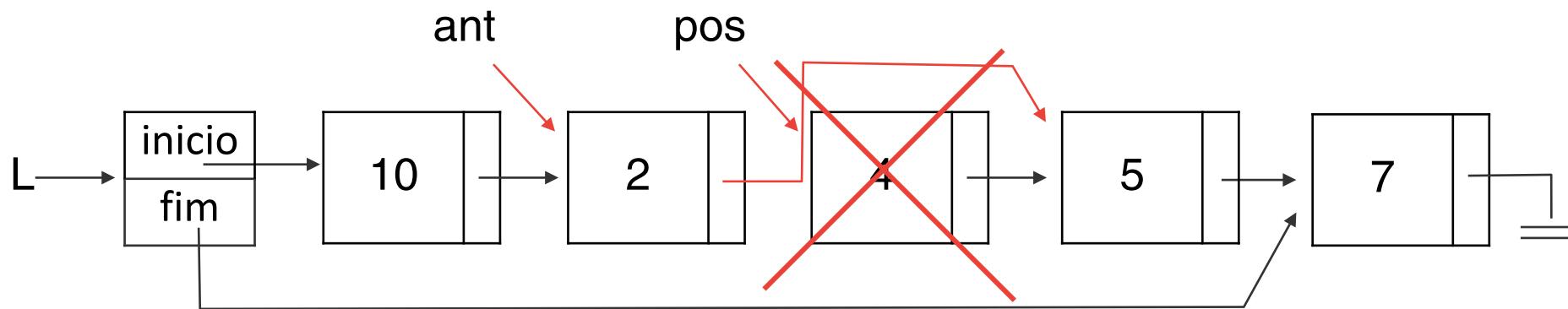


```
ant->prox = pos->prox;  
free(pos);  
pos = ant->prox;
```

Remover um elemento da Lista

- **Caso 2:** Elemento está no **meio** da lista

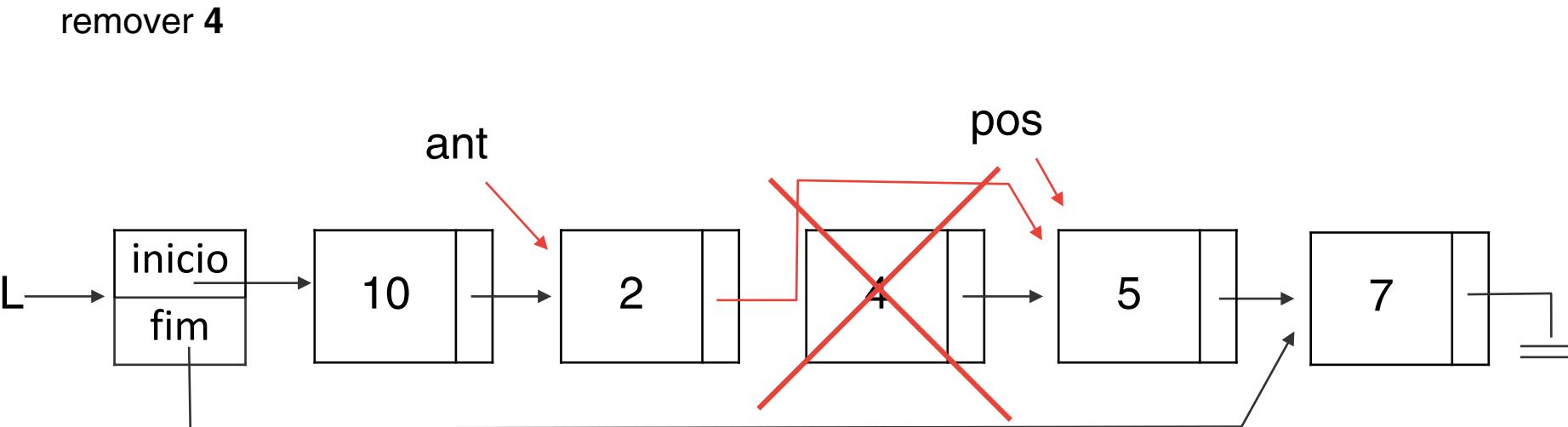
remover 4



```
ant->prox = pos->prox;  
free(pos);  
pos = ant->prox;
```

Remover um elemento da Lista

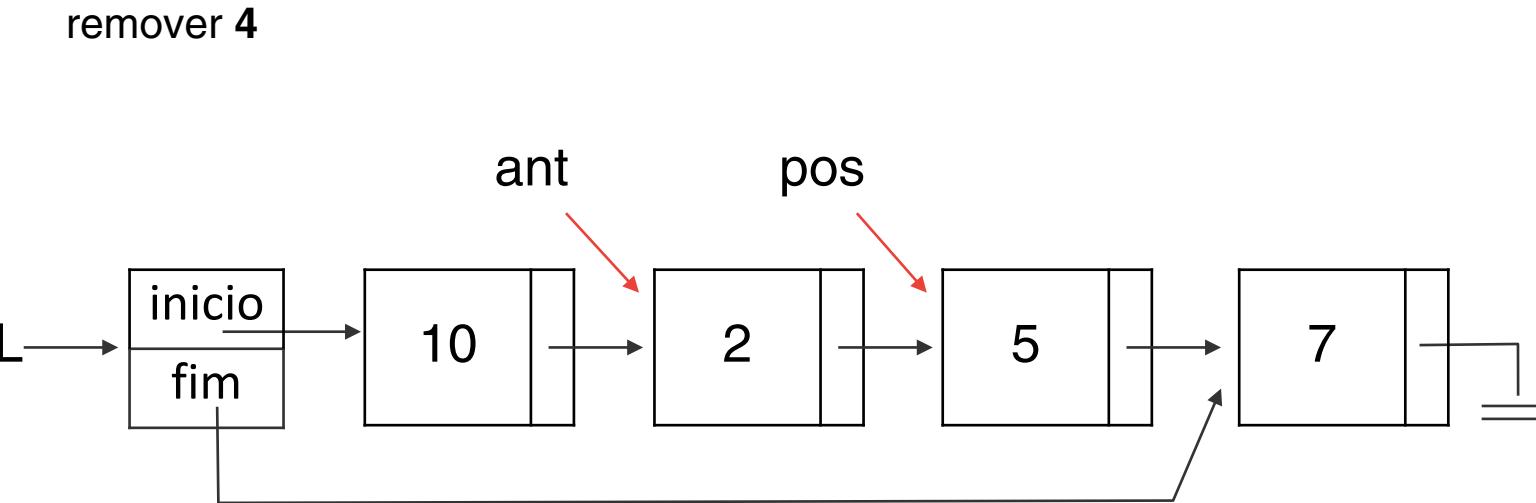
- **Caso 2:** Elemento está no **meio** da lista



```
ant->prox = pos->prox;  
free(pos);  
pos = ant->prox;
```

Remover um elemento da Lista

- Caso 2: Elemento está no **meio** da lista

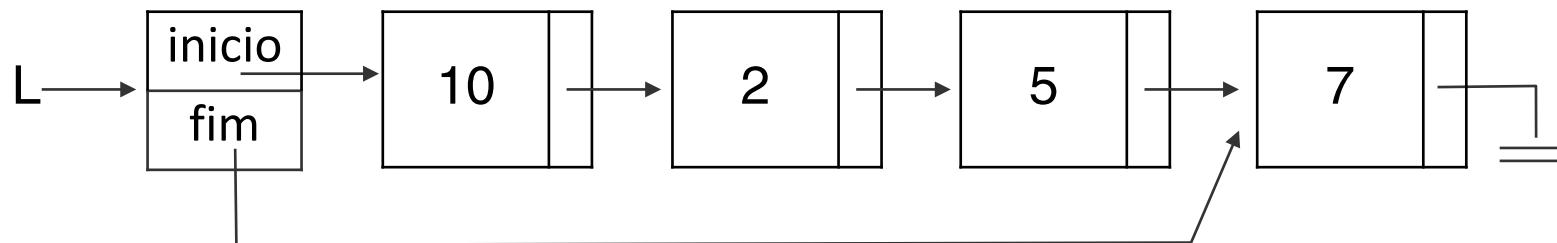


poderíamos continuar percorrendo
a lista para remover outros
possíveis nós 4

Remover um elemento da Lista

- **Caso 2:** Elemento está no **meio** da lista

remover 4

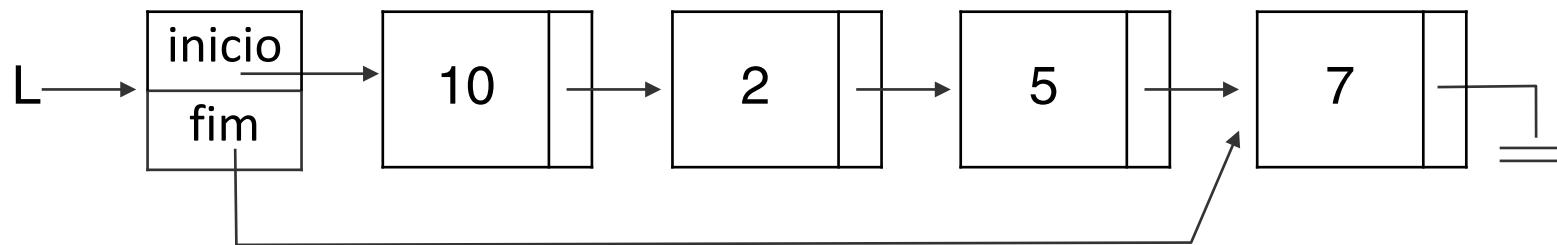


configuração final da lista
após a inserção

Remover um elemento da Lista

- Caso 3: Elemento está no **final (cauda)** da lista

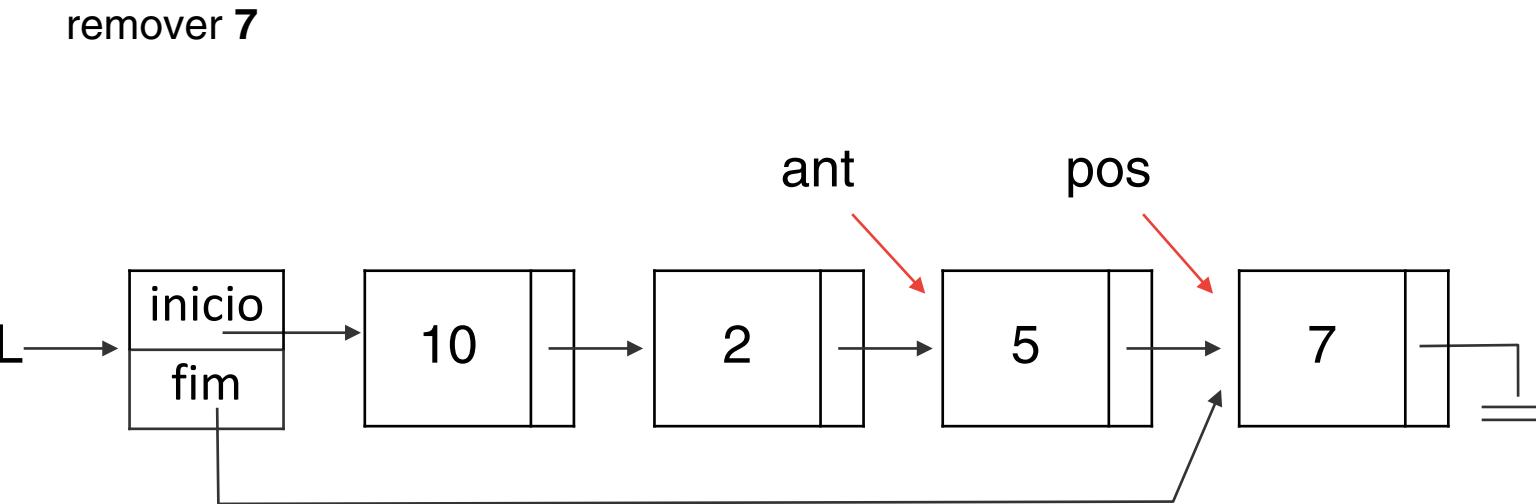
remover 7



configuração final da lista
após a inserção

Remover um elemento da Lista

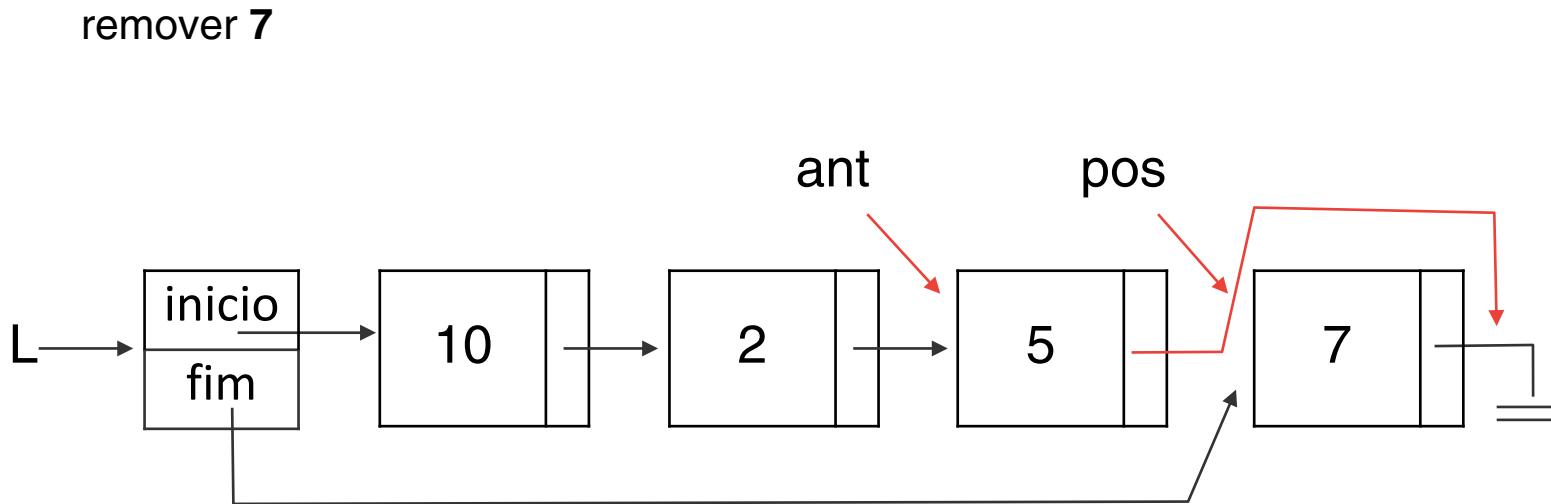
- Caso 3: Elemento está no **final (cauda)** da lista



Suponha que já percorremos
a lista inteira e chegamos no
nó final

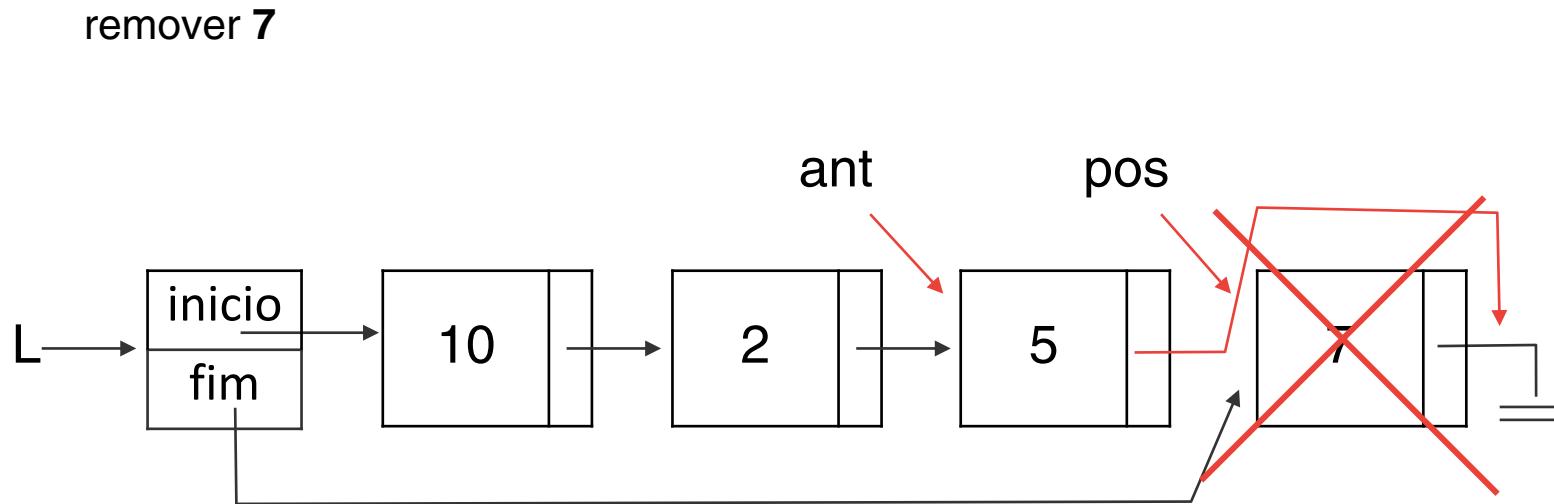
Remover um elemento da Lista

- Caso 3: Elemento está no **final (cauda)** da lista



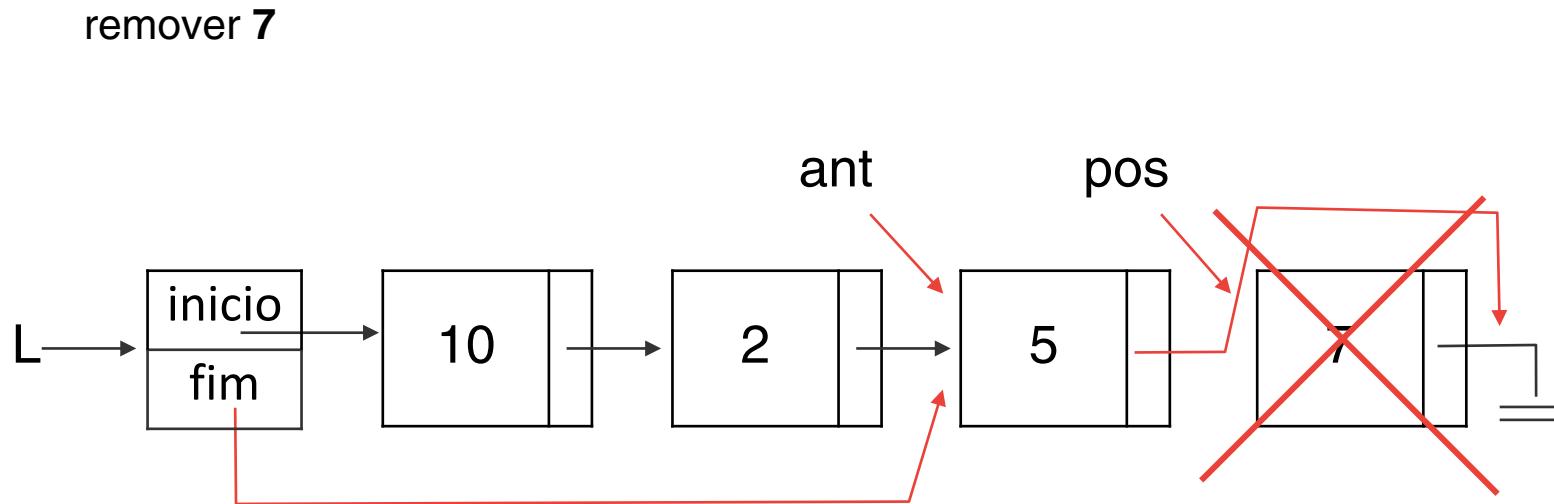
Remover um elemento da Lista

- Caso 3: Elemento está no **final (cauda)** da lista



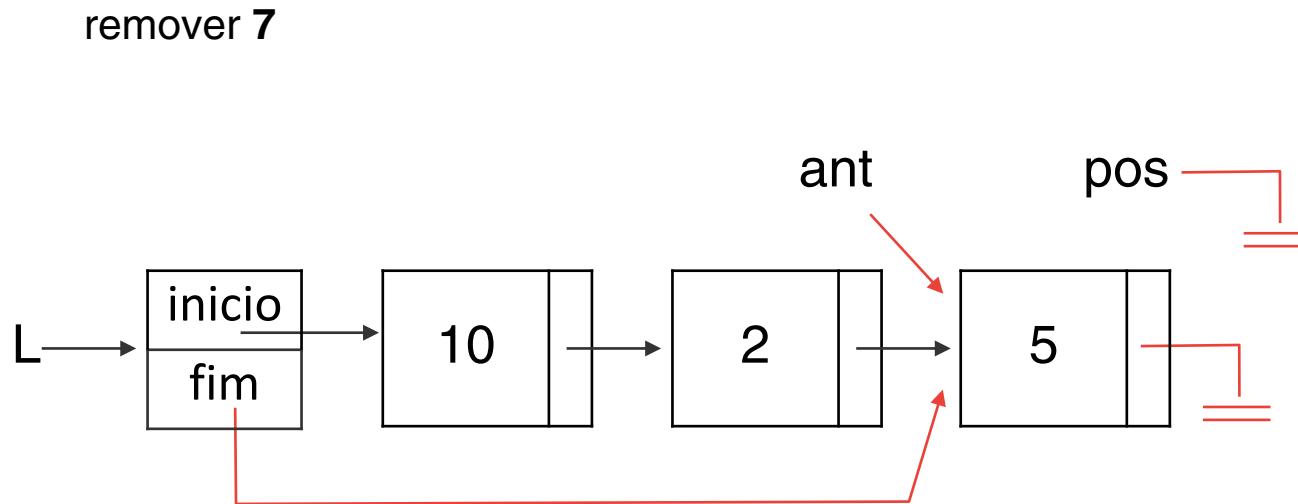
Remover um elemento da Lista

- Caso 3: Elemento está no **final (cauda)** da lista



Remover um elemento da Lista

- Caso 3: Elemento está no **final (cauda)** da lista



Listas Encadeadas Simples

- Diversos tipos de **operações**:
 - Inserção na cabeça (índice 0) da lista;
 - Impressão dos Elementos da Lista
 - Inserção na cauda (índice n-1) da lista;
 - **Remover elementos da lista**;
 - Contar o número de elementos da Lista;
 - Verificar se a lista está vazia e retornar verdadeiro/falso
 - Concatenar de duas listas;
 - Copiar uma lista;
 - Ordenação de uma lista;
 - Buscar um dado elemento na lista e retornar seu ponteiro;
 - Inserção ordenada;

Listas Encadeadas Simples

- Diversos tipos de **operações**:
 - Inserção na cabeça (índice 0) da lista;
 - Impressão dos Elementos da Lista
 - Inserção na cauda (índice n-1) da lista;
 - Remover elementos da lista;
 - Contar o número de elementos da Lista;
 - Verificar se a lista está vazia e retornar verdadeiro/falso
 - Concatenar de duas listas;
 - Copiar uma lista;
 - Ordenação de uma lista;
 - Buscar um dado elemento na lista e retornar seu ponteiro;
 - Inserção ordenada;

Arrays x Lista Encadeada

Arrays

Vantagens

- Acesso direto ($v[i]$): **rápido**
- Dados **contíguos** na memória
- Percorrer os elementos do array é **rápido**

Desvantagens

- Tamanho do array é fixado (não é flexível)

Listas Encadeadas

Vantagens

- Tamanho é **flexível**;

Desvantagens

- Acesso de um elemento é **sequencial**;
 - É necessário comparar elemento por elemento;
- Dados não contíguos na memória;
- Percorrer a lista inteira é **lento**;

Referências

- Feofiloff, Paulo. **Algoritmos em linguagem C**. Elsevier Brasil, 2009.
 - Cap 4