#### Estrutura de Dados I

Estrutura de dados: Lista

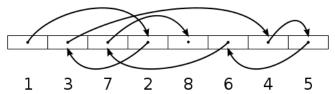
Prof. Rodrigo Minetto

Universidade Tecnológica Federal do Paraná

#### Sumário

- Introdução
- 2 Estrutura
- 3 Função: criar
- 4 Função: inserir
- 5 Função: inserir (funcionamento)
- 6 Função: imprimir
- Função: remover
- 8 Função: destruir

Um vetor ocupa um espaço contíguo na memória e nos permite acessar qualquer um de seus elementos a partir do ponteiro para o primeiro elemento. Dizemos que o vetor é uma estrutura que possibilita o acesso aleatório aos elementos.



Problemas: o vetor é uma estrutura não flexível. 1) O que acontece se o número de elementos que precisamos armazenar exceder a dimensão do vetor? 2) O que acontece se o número de elementos for muito inferior a sua dimensão? Não existe maneira barata (computacionalmente) para alterar a dimensão do vetor em tempo de execução.

**Solução:** utilizar estruturas de dados que cresçam conforme precisamos armazenar novos elementos (e diminuam conforme precisamos retirar elementos armazenados anteriormente). Estas estruturas são chamadas dinâmicas e armazenam cada um dos elementos por alocação dinâmica.

No entanto, em uma **lista encadeada**, a cada inserção alocamos um espaço de memória para armazená-lo mas **não** há garantia que eles ocuparão um **espaço de memória contíguo**. **Desvantagem**: não temos acesso aleatório aos elementos da lista (somente sequêncial):



#### Sumário

- Introdução
- 2 Estrutura
- 3 Função: criar
- 4 Função: inserir
- 5 Função: inserir (funcionamento)
- 6 Função: imprimir
- Função: remover
- 8 Função: destruir

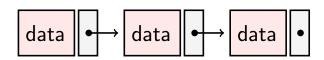
Cada elemento da lista é em geral chamado de **nó** (do inglês **node**). Um nó é representado por uma estrutura que contém a informação armazenada e um ponteiro.

```
struct node {
  int data;
  struct node* next;
};
```

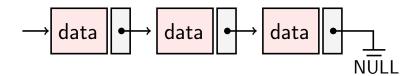
Embora não seja essencial, é uma boa prática de programação definir o tipo da estrutura. Assim List \* $\ell$  passa a representar struct node \* $\ell$ 

```
typedef struct node {
   int data;
   struct node* next;
} List;
```

Para percorrer todos os elementos da lista devemos explicitamente guardar o seu **encadea-mento**, que é realizado ao atribuir ao ponteiro de cada nó o endereço do próximo elemento.



A lista é representada por um ponteiro para o primeiro elemento (ou nó). O último elemento tem como próximo um ponteiro para **NULL**, e sinaliza, que não existe um próximo elemento.



### Tipo abstrato de dados

Tipo abstrato de dados **lista** (**interface**) create: inicializa uma estrutura de dados lista insert: adiciona um nó na lista (qual posição?) remove: remove um nó da lista size: retorna o número de nós na lista print: imprime todos os nós da lista destroy: remove todos os nós da lista

#### Sumário

- Introdução
- Estrutura
- 3 Função: criar
- 4 Função: inserir
- 5 Função: inserir (funcionamento)
- 6 Função: imprimir
- Função: removei
- 8 Função: destruir

## Listas encadeadas (criar)

O endereço de uma lista encadeada é definido como ponteiro para o primeiro elemento. Na variante lista sem cabeça, uma lista vazia é definida como um ponteiro para **NULL** 

```
1: List* create (void)
```

2: **return** NULL;

3:

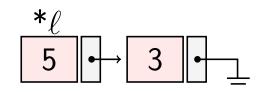
#### Sumário

- Introdução
- Estrutura
- 3 Função: criar
- 4 Função: inserir
- 5 Função: inserir (funcionamento)
- 6 Função: imprimir
- Função: remover
- 8 Função: destruir

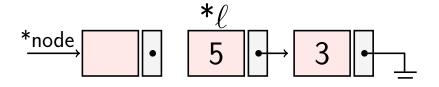
A função de inserção mais simples insere um novo elemento no **início** da lista. No entanto, a nova cabeça da lista passa a apontar para o novo primeiro elemento.

```
1: List* insert (List *\ell, int elem)
```

- List \*node = (List\*)malloc(sizeof(List));
- 3: node $\rightarrow$ data = elem;
- 4:  $\mathsf{node} \rightarrow \mathsf{next} = \ell;$
- 5: return node;

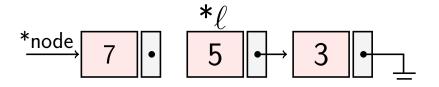


- 1: **List\* insert** (List  $*\ell$ , int elem)
- List \*node = (List\*)malloc(sizeof(List));
- 3: node $\rightarrow$ data = elem;
- 4:  $\mathsf{node} \rightarrow \mathsf{next} = \ell;$
- 5: return node;

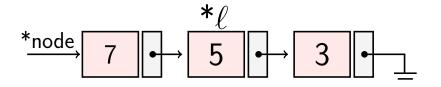


```
1: List* insert (List *ℓ, int elem)
```

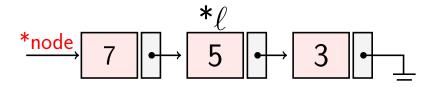
- List \*node = (List\*)malloc(sizeof(List));
- 3: node $\rightarrow$ data = elem;
- 4:  $\mathsf{node} \rightarrow \mathsf{next} = \ell;$
- 5: return node;



- 1: **List\* insert** (List  $*\ell$ , int elem)
- List \*node = (List\*)malloc(sizeof(List));
- 3: node $\rightarrow$ data = elem;
- 4:  $\mathsf{node} \rightarrow \mathsf{next} = \ell;$
- 5: return node;



- 1: **List\* insert** (List  $*\ell$ , int elem)
- List \*node = (List\*)malloc(sizeof(List));
- 3: node $\rightarrow$ data = elem;
- 4:  $node \rightarrow next = \ell$ ;
- 5: return node;



- 1: **List\* insert** (List  $*\ell$ , int elem)
- List \*node = (List\*)malloc(sizeof(List));
- 3: node $\rightarrow$ data = elem;
- 4:  $\mathsf{node} \rightarrow \mathsf{next} = \ell;$
- 5: return node;

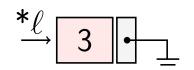
#### Sumário de la sumario de la su

- Introdução
- Estrutura
- 3 Função: criar
- 4 Função: inserir
- 5 Função: inserir (funcionamento)
- 6 Função: imprimir
- Função: remover
- 8 Função: destruir

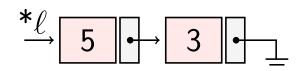
- List \* $\ell$  = create ();
- 3:  $\ell = \text{insert } (\ell, 3);$
- 4:  $\ell = \text{insert } (\ell, 5);$ 5:  $\ell = \text{insert } (\ell, 7);$ 
  - : return 0



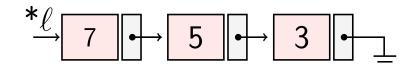
- 2: List  $*\ell = \text{create ()};$ 
  - 3:  $\ell = \text{insert } (\ell, 3);$
  - 4:  $\ell = \text{insert } (\ell, 5);$ 5:  $\ell = \text{insert } (\ell, 7);$ 
    - s: return 0



- 2: List  $*\ell = \text{create ()};$
- $\ell = \text{insert } (\ell, 3);$
- 4:  $\ell = \text{insert } (\ell, 5);$
- 5:  $\ell = \text{insert } (\ell, 7);$
- s: return 0



- 2: List \* $\ell$  = create ();
- $\ell = \text{insert } (\ell, 3);$
- 4:  $\ell = \text{insert } (\ell, 5);$
- 5:  $\ell = \text{insert } (\ell, 7);$ 
  - s: return 0



- List \* $\ell$  = create ();
- 3:  $\ell = \text{insert } (\ell, 3);$
- 4:  $\ell = \text{insert } (\ell, 5);$
- $\ell = \mathsf{insert}(\ell, 7);$
- s: return 0

#### Sumário

- Introdução
- Estrutura
- 3 Função: criar
- 4 Função: inserir
- 5 Função: inserir (funcionamento)
- 6 Função: imprimir
- Função: removei
- 8 Função: destruir

A função para **imprimir** usa o operador diferente  $(\neq)$  para determinar o final da lista. Observe a importância que o último ponteiro seja **NULL** (fonte comum de erros).

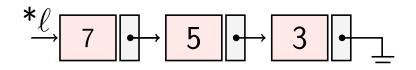
```
1: void print (List *\ell)

2: List *t;

3: for (t = \ell; t \neq NULL; t = t\rightarrownext) {

4: printf("Data: %d", t\rightarrowdata);

5: }
```



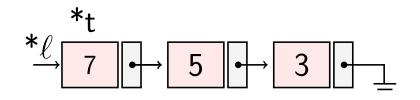
```
1: void print (List *\ell)

2: List *t;

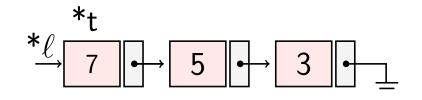
3: for (t = \ell; t \neq NULL; t = t\rightarrownext) {

4: printf("Data: %d", t\rightarrowdata);

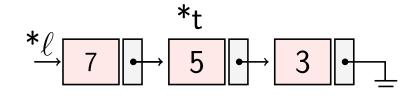
5: }
```



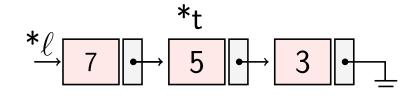
```
    void print (List *ℓ)
    List *t;
    for (t = ℓ; t ≠ NULL; t = t→next) {
    printf("Data: %d", t→data);
    }
```



```
    void print (List *ℓ)
    List *t;
    for (t = ℓ; t ≠ NULL; t = t→next) {
    printf("Data: %d", t→data);
    }
```



```
    void print (List *ℓ)
    List *t;
    for (t = ℓ; t ≠ NULL; t = t→next) {
    printf("Data: %d", t→data);
    }
```



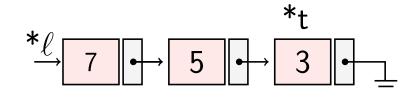
```
1: void print (List *\ell)

2: List *t;

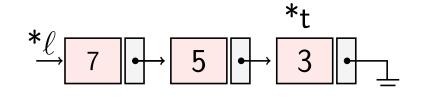
3: for (t = \ell; t \neq NULL; t = t \rightarrow next) {

4: printf("Data: %d", t \rightarrow data);

5: }
```

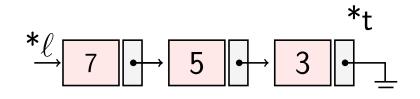


```
    void print (List *ℓ)
    List *t;
    for (t = ℓ; t ≠ NULL; t = t→next) {
    printf("Data: %d", t→data);
    }
```



```
    void print (List *ℓ)
    List *t;
    for (t = ℓ; t ≠ NULL; t = t→next) {
    printf("Data: %d", t→data);
    }
```

# Listas encadeadas (imprimir)



```
    void print (List *ℓ)
    List *t;
    for (t = ℓ; t ≠ NULL; t = t→next) {
    printf("Data: %d", t→data);
    }
```

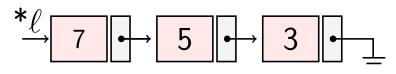
#### Sumário

- Introdução
- Estrutura
- 3 Função: criar
- 4 Função: inserir
- 5 Função: inserir (funcionamento)
- 6 Função: imprimir
- Função: remover
- 8 Função: destruir

A função para **remover** um nó da lista precisa buscar o elemento e então descobrir a posição na lista (que influencia na reconexão da lista)

```
1: List* remove (List *\ell, int elem)
        List *t = \ell, *p = NULL;
        while ((t \neq NULL) and (t \rightarrow data \neq elem))
3:
            p = t;
4:
           t = t \rightarrow next:
5:
6: if (t = NULL) { return \ell; }
        if (p = NULL) \{ \ell = t \rightarrow next; \}
7:
        else { p \rightarrow next = t \rightarrow next; }
8:
        free (t);
9:
   return \ell:
```

Remover: 5



```
    List* remove (List *ℓ, int elem)
    List *t = ℓ, *p = NULL;
```

3: **while** 
$$((t \neq NULL)$$
 **and**  $(t \rightarrow data \neq elem))$ 

4: 
$$p = t$$
;

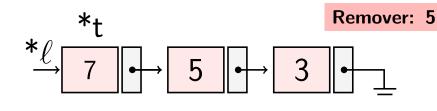
$$\mathsf{t} = \mathsf{t} {
ightarrow} \mathsf{next};$$

6: **if** 
$$(t = NULL)$$
 { return  $\ell$ ; }

7: **if** 
$$(p = NULL) \{ \ell = t \rightarrow next; \}$$

8: **else** { 
$$p\rightarrow next = t\rightarrow next;$$
 }  
9: **free** (t);

10: return 
$$\ell$$
;



```
1: List* remove (List *\ell, int elem)
```

```
2: List *t = \ell, *p = NULL;
3: while ((t \neq NULL) and (t\rightarrowdata \neq elem))
```

4: 
$$p = t$$
;

5:

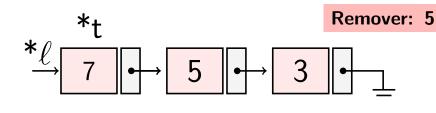
$$\mathsf{t} = \mathsf{t} {\rightarrow} \mathsf{next};$$

6: **if** 
$$(t = NULL)$$
 { return  $\ell$ ; }

7: **if** 
$$(p = NULL) \{ \ell = t \rightarrow next; \}$$

8: **else** { 
$$p\rightarrow next = t\rightarrow next$$
; }  
9: **free** (t);

10: return  $\ell$ ;



```
List* remove (List *\ell, int elem)
       List *t = \ell, *p = NULL;
2:
```

while 
$$((t \neq NULL)$$
 and  $(t\rightarrow data \neq elem))$ 

$$p = t;$$

$$t = t \rightarrow next$$
;

6: **if** 
$$(t = NULL)$$
 { return  $\ell$ ; }

7: **if** 
$$(p = NULL) \{ \ell = t \rightarrow next; \}$$

8: **else** { 
$$p\rightarrow next = t\rightarrow next$$
; }  
9: **free** (t);

return  $\ell$ : 10:

4:

```
*t,*p

*\ell 7 \longrightarrow 5 \longrightarrow 3
```

```
    List* remove (List *ℓ, int elem)
    List *t = ℓ, *p = NULL;
```

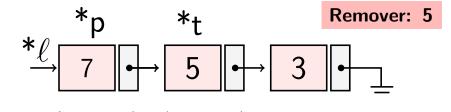
```
3: while ((t \neq NULL) and (t\rightarrow data \neq elem))
4: p = t;
```

5: 
$$t = t \rightarrow \text{next};$$

6: **if** 
$$(t = NULL)$$
 { return  $\ell$ ; }  
7: **if**  $(p = NULL)$  {  $\ell = t \rightarrow next$ ; }

8: **else** { 
$$p \rightarrow next = t \rightarrow next$$
; }  
9: **free** (t);

return  $\ell$ ;



```
1: List* remove (List *\ell, int elem)
2: List *t = \ell, *p = NULL;
```

2: List \*t = 
$$\ell$$
, \*p = NULL;  
3: **while** ((t  $\neq$  NULL) **and**

while 
$$((t \neq NULL)$$
 and  $(t \rightarrow data \neq elem))$ 

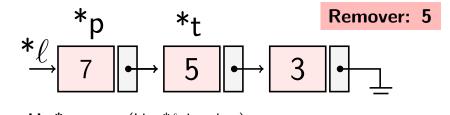
4: 
$$p = t$$
;  
5:  $t = t \rightarrow next$ :

6: **if** 
$$(t = NULL)$$
 { return  $\ell$ ; }

7: **if** (p = NULL) { 
$$\ell = t \rightarrow next;$$
 }

8: **else** { 
$$p \rightarrow next = t \rightarrow next$$
; }  
9: **free** (t);

10: return  $\ell$ ;



while  $((t \neq NULL)$  and  $(t\rightarrow data \neq elem))$ 

```
List* remove (List *\ell, int elem)
       List *t = \ell, *p = NULL;
2:
```

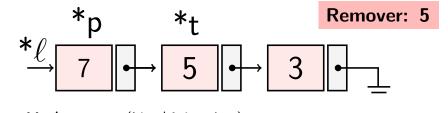
p = t; 4: 5:  $t = t \rightarrow next$ :

**if**  $(t = NULL) \{ return \ell; \}$ 6:

**if** (p = NULL) {  $\ell = t \rightarrow next$ ; } 7: **else** {  $p \rightarrow next = t \rightarrow next; }$ 8:

free (t); 10: return  $\ell$ :

3:



```
1: List* remove (List *\ell, int elem)
2: List *t = \ell, *p = NULL;
3: while ((t \neq NULL) and (t \rightarrow data \neq elem))
```

p = t;

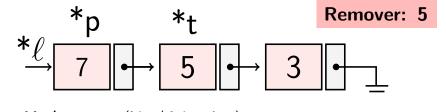
 $t = t \rightarrow next;$ **if**  $(t = NULL) \{ return \ell; \}$ 

7: **if**  $(p = NULL) \{ \ell = t \rightarrow next; \}$ 

8: **else** {  $p \rightarrow next = t \rightarrow next$ ; } 9: **free** (t);

10: return  $\ell$ ;

4: 5:



```
1: List* remove (List *\ell, int elem)

2: List *t = \ell, *p = NULL;

3: while ((t \neq NULL) and (t \rightarrow data \neq elem))

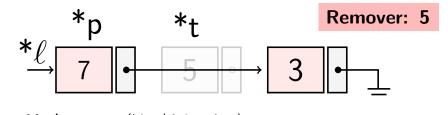
4: p = t;
```

6: **if** (t = NULL) { return  $\ell$ ; } 7: **if** (p = NULL) {  $\ell = t \rightarrow ne$ 

 $t = t \rightarrow next$ :

7: **if**  $(p = NULL) \{ \ell = t \rightarrow next; \}$ 8: **else**  $\{ p \rightarrow next = t \rightarrow next; \}$ 9: **free** (t);

10: return  $\ell$ ;



```
    List* remove (List *ℓ, int elem)
    List *t = ℓ, *p = NULL;
```

3: **while** 
$$((t \neq NULL)$$
 **and**  $(t\rightarrow data \neq elem))$ 

4: 
$$p = t$$
;

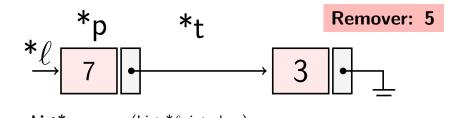
5: 
$$t = t \rightarrow next;$$
  
6: **if**  $(t = NULL)$  { return  $\ell$ ; }

7: **if** 
$$(p = NULL) \{ \ell = t \rightarrow next; \}$$

7: If 
$$(p = \text{NOLL}) \{ \ell = t \rightarrow \text{next}; \}$$

8: **else** { 
$$p \rightarrow next = t \rightarrow next$$
; }  
9: **free** (t);

10: return  $\ell$ ;



```
    List* remove (List *ℓ, int elem)
    List *t = ℓ, *p = NULL;
```

3: while 
$$((t \neq NULL)$$
 and  $(t \rightarrow data \neq elem))$ 

$$p = t;$$

4:

10:

5: 
$$t = t \rightarrow next$$
;  
6: **if**  $(t = NULL)$  { return  $\ell$ ; }

7: **if** (p = NULL) { 
$$\ell = t \rightarrow next;$$
 }

8: **else** { 
$$p\rightarrow next = t\rightarrow next$$
; }  
9: **free** (t);

return  $\ell$ ;

Remover: 5



```
1: List* remove (List *\ell, int elem)
```

2: List \*t = 
$$\ell$$
, \*p = NULL;  
3: **while** ((t  $\neq$  NULL) **and** (t $\rightarrow$ data  $\neq$  elem))

4: 
$$p = t$$
;

5: 
$$t = t \rightarrow \text{next}$$
;

6: **if** 
$$(t = NULL) \{ return \ell; \}$$

7: **if** 
$$(p = NULL) \{ \ell = t \rightarrow next; \}$$

8: **else** { 
$$p\rightarrow next = t\rightarrow next$$
; }  
9: **free** (t);

10: **return**  $\ell$ ;

Remover: 3



```
1: List* remove (List *\ell, int elem)
```

2: List \*t = 
$$\ell$$
, \*p = NULL;  
3: **while** ((t  $\neq$  NULL) **and** (t $\rightarrow$ data  $\neq$  elem))

$$p = t$$
:

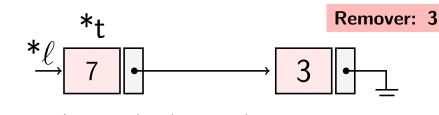
$$p = \iota$$
,

$$\mathsf{t} = \mathsf{t} {
ightarrow} \mathsf{next};$$
**if**  $(\mathsf{t} = \mathsf{NULL}) \ \{ \mathsf{return} \ \ell; \ \}$ 

7: **if** 
$$(p = NULL) \{ \ell = t \rightarrow next; \}$$

8: **else** { 
$$p \rightarrow \text{next} = t \rightarrow \text{next}$$
; }

4: 5:



```
    List* remove (List *ℓ, int elem)
    List *t = ℓ, *p = NULL;
```

3: **while** 
$$((t \neq NULL)$$
 **and**  $(t \rightarrow data \neq elem))$ 

4: 
$$p = t$$
;

5:

$$\mathsf{t} = \mathsf{t} {
ightarrow} \mathsf{next}; \\ \mathbf{if} \ (\mathsf{t} = \mathsf{NULL}) \ \{ \ \mathsf{return} \ \ell; \ \}$$

6: **if** (t = NULL) { return 
$$\ell$$
; }  
7: **if** (p = NULL) {  $\ell$  = t $\rightarrow$ next; }

8: **else** 
$$\{ p \rightarrow \text{next} = t \rightarrow \text{next}; \}$$
  
9: **free**  $\{t\};$ 

10: return  $\ell$ ;



```
    List* remove (List *ℓ, int elem)
    List *t = ℓ, *p = NULL;
```

3: **while** 
$$((t \neq NULL)$$
 **and**  $(t\rightarrow data \neq elem))$ 

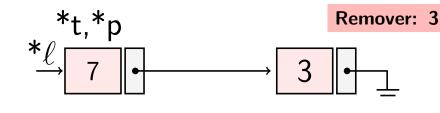
4: 
$$p = t$$
;  
5:  $t = t \rightarrow next$ ;

return  $\ell$ :

6: **if** 
$$(t = NULL)$$
 { return  $\ell$ ; }

7: **if** 
$$(p = NULL) \{ \ell = t \rightarrow next; \}$$

8: **else** { 
$$p \rightarrow next = t \rightarrow next$$
; }  
9: **free** (t);



```
    List* remove (List *ℓ, int elem)
    List *t = ℓ, *p = NULL;
```

3: **while** 
$$((t \neq NULL)$$
 **and**  $(t \rightarrow data \neq elem))$ 

$$p = t$$
;

4:

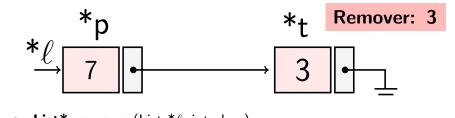
5:

$$t = t \rightarrow next;$$

6: **if** (t = NULL) { return 
$$\ell$$
; }  
7: **if** (p = NULL) {  $\ell$  = t $\rightarrow$ next; }

7: **if** 
$$(p = NULL) \{ \ell = t \rightarrow next; \}$$
  
8: **else**  $\{ p \rightarrow next = t \rightarrow next; \}$   
9: **free**  $(t);$ 

return 
$$\ell$$
;



```
List* remove (List *ℓ, int elem)
      List *t = \ell, *p = NULL;
2:
```

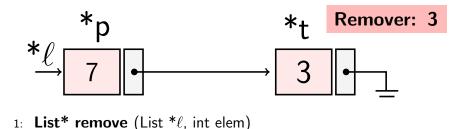
3: **while** 
$$((t \neq NULL)$$
 **and**  $(t \rightarrow data \neq elem))$ 

$$p = t;$$

5: 
$$t = t \rightarrow next$$
;  
6: **if**  $(t = NULL)$  { return  $\ell$ ; }

7: **if** 
$$(p = NULL)$$
 {  $\ell = t \rightarrow next$ ; }  
8: **else** {  $p \rightarrow next = t \rightarrow next$ ; }

4:



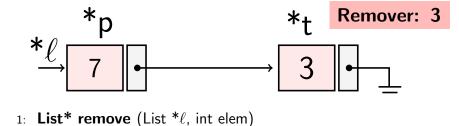
```
2: List *t = \ell, *p = NULL;
```

3: **while** 
$$((t \neq NULL)$$
 **and**  $(t\rightarrow data \neq elem))$   
4:  $p = t$ ;

5: 
$$t = t \rightarrow next;$$

6: **if** 
$$(t = NULL)$$
 { return  $\ell$ ; }  
7: **if**  $(p = NULL)$  {  $\ell = t \rightarrow next$ ; }

7: If 
$$(p = NULL)$$
 {  $\ell = t \rightarrow next$ ; }  
8: **else** {  $p \rightarrow next = t \rightarrow next$ ; }  
9: **free** (t);  
10: **return**  $\ell$ :



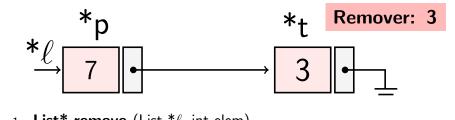
```
2: List *t = \ell, *p = NULL;
```

3: **while**  $((t \neq NULL)$  **and**  $(t\rightarrow data \neq elem))$ 4: p = t;

$$t = t \rightarrow next;$$

6: **if** 
$$(t = NULL)$$
 { return  $\ell$ ; }  
7: **if**  $(p = NULL)$  {  $\ell = t \rightarrow next$ ; }

8: **else** {  $p \rightarrow next = t \rightarrow next$ ; } 9: **free** (t); 10: **return**  $\ell$ :



```
    List* remove (List *ℓ, int elem)
    List *t = ℓ, *p = NULL;
```

3: **while** 
$$((t \neq NULL)$$
 **and**  $(t \rightarrow data \neq elem))$ 

$$p = t$$
;

4:

5:

10:

return  $\ell$ :

$$t = t \rightarrow next;$$
  
**if**  $(t = NULL) \{ return \ell; \}$ 

6: **if** 
$$(t = NULL) \{ return \ell; \}$$
  
7: **if**  $(p = NULL) \{ \ell = t \rightarrow next; \}$ 

7: If 
$$(p = NULL)$$
 {  $\ell = t \rightarrow next$ ; }  
8: else {  $p \rightarrow next = t \rightarrow next$ ; }  
9: free  $(t)$ ;

```
*\ell 7 • Remover: 3
```

```
    List* remove (List *ℓ, int elem)
    List *t = ℓ, *p = NULL;
```

3: **while** 
$$((t \neq NULL)$$
 **and**  $(t\rightarrow data \neq elem))$   
4:  $p = t$ ;

5: 
$$t = t \rightarrow next;$$

6: **if** 
$$(t = NULL)$$
 { return  $\ell$ ; }

7: **if** 
$$(p = NULL) \{ \ell = t \rightarrow next; \}$$
  
8: **else**  $\{ p \rightarrow next = t \rightarrow next; \}$ 

10: return 
$$\ell$$
;

9:

free (t);

```
*\ell 7 • Remover: 3
```

```
    List* remove (List *ℓ, int elem)
    List *t = ℓ, *p = NULL;
```

```
3: while ((t \neq NULL) and (t\rightarrow data \neq elem))
4: p = t;
```

5: 
$$t = t \rightarrow next;$$
  
6: **if**  $(t = NULL) \{ return \ell; \}$ 

7: **if** 
$$(p = NULL)$$
 {  $\ell = t \rightarrow next$ ; }  
8: **else** {  $p \rightarrow next = t \rightarrow next$ ; }

10: return  $\ell$ ;

9:

free (t);

Remover: 3

```
List* remove (List *ℓ, int elem)
        List *t = \ell, *p = NULL;
        while ((t \neq NULL) and (t \rightarrow data \neq elem))
3:
4:
            p = t;
5:
            t = t \rightarrow next:
     if (t = NULL) { return \ell; }
6:
        if (p = NULL) \{ \ell = t \rightarrow next; \}
7:
        else { p \rightarrow next = t \rightarrow next; }
8:
        free (t);
9:
    return \ell:
```

Remover:

```
List* remove (List *\ell, int elem)
```

- List \*t =  $\ell$ , \*p = NULL; 2: while  $((t \neq NULL)$  and  $(t\rightarrow data \neq elem))$ 3:
- p = t; 4:
- $t = t \rightarrow next$ : 5:
- **if** (t = NULL) { return  $\ell$ ; } 6:
- **if**  $(p = NULL) \{ \ell = t \rightarrow next; \}$ 7: **else** {  $p \rightarrow next = t \rightarrow next; }$ 8:
- free (t); 9: return  $\ell$ : 10:

```
1: List* remove (List *\ell, int elem)
2: List *t = \ell, *p = NULL;
3: while ((t \neq NULL) and (t \rightarrow data \neq elem))
```

 $\mathsf{p} = \mathsf{t}; \ \mathsf{t} = \mathsf{t} {
ightarrow} \mathsf{next};$ 

6: **if**  $(t = NULL) \{ return \ell; \}$ 

7: **if**  $(p = NULL) \{ \ell = t \rightarrow next; \}$ 8: **else**  $\{ p \rightarrow next = t \rightarrow next; \}$ 9: **free** (t);

return  $\ell$ ;

4:

5:

```
1: List* remove (List *\ell, int elem)
2: List *t = \ell, *p = NULL;
3: while ((t \neq NULL) and (t \rightarrow data \neq elem))
```

5:  $t = t \rightarrow next;$ 6: **if** (t = NULL) { return  $\ell$ ; }

p = t;

7: **if** (p = NULL) {  $\ell = t \rightarrow next$ ; } 8: **else** {  $p \rightarrow next = t \rightarrow next$ ; } 9: **free** (t);

10: return  $\ell$ ;

```
    List* remove (List *ℓ, int elem)
    List *t = ℓ, *p = NULL;
```

```
3: while ((t \neq NULL) and (t\rightarrow data \neq elem))
4: p = t;
```

5: 
$$t = t \rightarrow next$$
;  
6: **if**  $(t = NULL)$  { return  $\ell$ ; }

7: **if** (p = NULL) { 
$$\ell$$
 = t $\rightarrow$ next; }  
8: **else** { p $\rightarrow$ next = t $\rightarrow$ next; }

```
*\ell \xrightarrow{7} 7
Remover: 7
```

```
    List* remove (List *ℓ, int elem)
    List *t = ℓ, *p = NULL;
```

```
3: while ((t \neq NULL) and (t\rightarrow data \neq elem))
4: p = t;
```

$$t = t \rightarrow next;$$
**if**  $(t = NULL) \{ return \ \ell; \}$ 

6: **if** (t = NULL) { return 
$$\ell$$
; }  
7: **if** (p = NULL) {  $\ell$  = t $\rightarrow$ next; }  
8: **else** { p $\rightarrow$ next = t $\rightarrow$ next; }

free (t);

return  $\ell$ :

```
*t
*

Remover: 7
```

```
1: List* remove (List *\ell, int elem)

2: List *t = \ell, *p = NULL;

3: while ((t \neq NULL) and (t \rightarrow data \neq elem))

4: p = t;

5: t = t \rightarrow next;

6: if (t = NULL) { return \ell; }

7: if (p = NULL) { \ell = t \rightarrow next; }

8: else { p \rightarrow next = t \rightarrow next; }
```

```
Remover: 7
   List* remove (List *\ell, int elem)
        List *t = \ell, *p = NULL;
        while ((t \neq NULL) and (t \rightarrow data \neq elem))
3:
4:
            p = t;
            t = t \rightarrow next:
5:
       if (t = NULL) { return \ell; }
6:
        if (p = NULL) \{ \ell = t \rightarrow next; \}
7:
        else { p \rightarrow next = t \rightarrow next; }
8:
        free (t);
9:
    return \ell:
```

Remover: 7

```
List* remove (List *\ell, int elem)
        List *t = \ell, *p = NULL;
        while ((t \neq NULL) and (t \rightarrow data \neq elem))
3:
4:
            p = t;
            t = t \rightarrow next:
5:
        if (t = NULL) { return \ell; }
6:
        if (p = NULL) \{ \ell = t \rightarrow next; \}
7:
        else { p \rightarrow next = t \rightarrow next; }
8:
        free (t);
9:
    return \ell:
```

#### Sumário de la sumario de la su

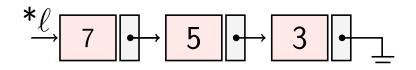
- Introdução
- Estrutura
- 3 Função: criar
- 4 Função: inseria
- 5 Função: inserir (funcionamento)
- 6 Função: imprimir
- Função: remover
- 8 Função: destruir

### Listas encadeadas (destruir)

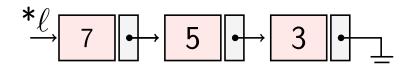
A função para **destruir** (**ou desalocar**) uma lista precisa ter o cuidado de não liberar um nó antes de guardar o ponteiro do próximo elemento.

```
1: void destroy (List *\ell)
2: while (\ell \neq \text{NULL})
3: List *t = \ell \rightarrow \text{next};
4: free (\ell);
5: \ell = t;
6: }
```

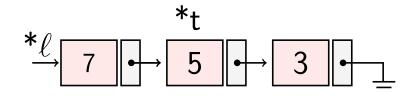
#### Listas encadeadas (destruir)



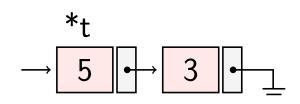
```
1: void destroy (List *\ell)
2: while (\ell \neq \text{NULL})
3: List *t = \ell \rightarrow \text{next};
4: free (\ell);
5: \ell = t;
```



```
1: void destroy (List *\ell)
2: while (\ell \neq NULL)
3: List *t = \ell \rightarrow next;
4: free (\ell);
5: \ell = t;
6: }
```

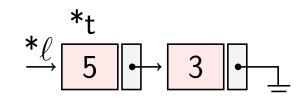


```
1: void destroy (List *\ell)
2: while (\ell \neq \text{NULL})
3: List *t = \ell \rightarrow \text{next};
4: free (\ell);
5: \ell = t;
6: }
```

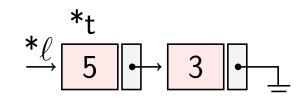


```
1: void destroy (List *\ell)
2: while (\ell \neq \text{NULL})
3: List *t = \ell \rightarrow \text{next};
4: free (\ell);
5: \ell = t;
```

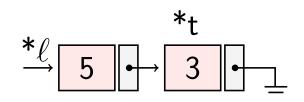
6:



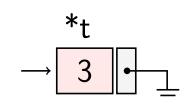
```
1: void destroy (List *\ell)
2: while (\ell \neq \text{NULL})
3: List *t = \ell \rightarrow \text{next};
4: free (\ell);
5: \ell = t;
6: }
```



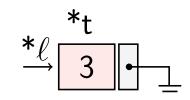
```
1: void destroy (List *\ell)
2: while (\ell \neq NULL)
3: List *t = \ell \rightarrow next;
4: free (\ell);
5: \ell = t;
6: }
```



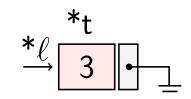
```
1: void destroy (List *\ell)
2: while (\ell \neq \text{NULL})
3: List *t = \ell \rightarrow \text{next};
4: free (\ell);
5: \ell = t;
6: }
```



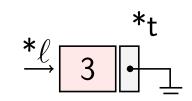
```
1: void destroy (List *\ell)
2: while (\ell \neq \text{NULL})
3: List *t = \ell \rightarrow \text{next};
4: free (\ell);
5: \ell = t;
6: }
```



```
1: void destroy (List *\ell)
2: while (\ell \neq \text{NULL})
3: List *t = \ell \rightarrow \text{next};
4: free (\ell);
5: \ell = t;
6: }
```



```
1: void destroy (List *\ell)
2: while (\ell \neq NULL)
3: List *t = \ell \rightarrow next;
4: free (\ell);
5: \ell = t;
6: }
```



```
1: void destroy (List *\ell)
2: while (\ell \neq \text{NULL})
3: List *t = \ell \rightarrow \text{next};
4: free (\ell);
5: \ell = t;
6: }
```

```
*t
```

```
1: void destroy (List *\ell)
2: while (\ell \neq \text{NULL})
3: List *t = \ell \rightarrow \text{next};
4: free (\ell);
5: \ell = \text{t};
6: }
```

```
*t
*ℓ
<u></u>___
```

```
1: void destroy (List *\ell)
2: while (\ell \neq \text{NULL})
3: List *t = \ell \rightarrow \text{next};
4: free (\ell);
5: \ell = t;
6: }
```

```
*t
*ℓ
<u></u>___
```

```
1: void destroy (List *\ell)
2: while (\ell \neq \text{NULL})
3: List *t = \ell \rightarrow \text{next};
4: free (\ell);
5: \ell = t;
6: }
```