Estrutura de Dados I

Estrutura de dados: lista duplamente encadeada

Prof. Rodrigo Minetto

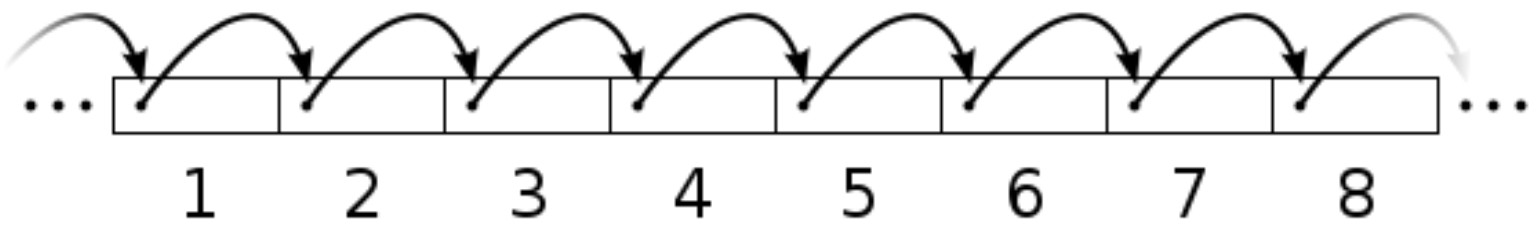
Universidade Tecnológica Federal do Paraná

Sumário

1. Introdução
2. Função: **inserir**
3. Função: **inserir (funcionamento)**
4. Função: **remover**

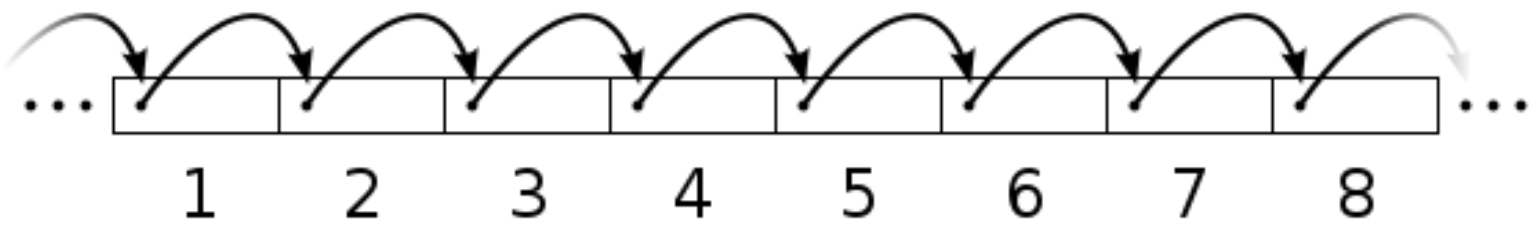
Introdução

O tipo abstrato de dados **lista** visto aula passada caracteriza-se por formar um encadeamento simples entre os elementos — cada nó armazena um ponteiro para o próximo elemento da lista.



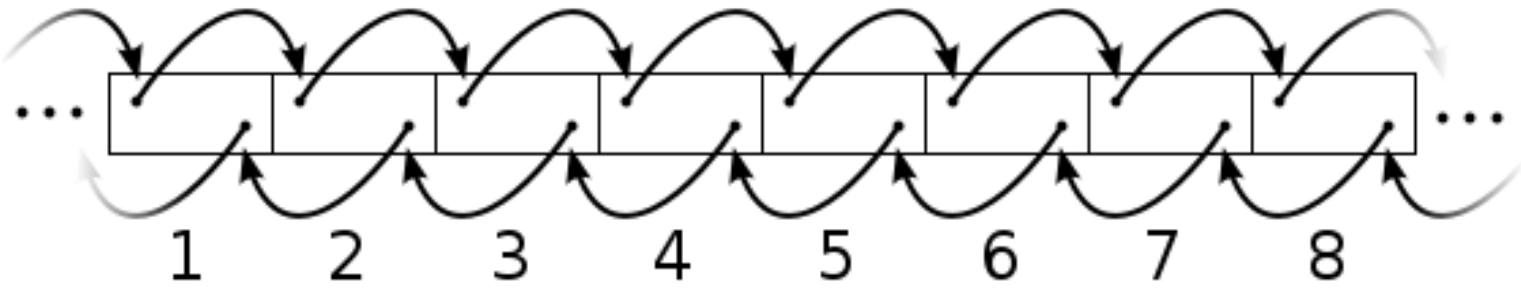
Introdução

Problemas: **1)** não é possível percorrer eficientemente os elementos em ordem inversa (do fim para o início); **2)** dificuldade para remoção, pois dado um determinado nó não temos como acessar diretamente seu antecessor.



Introdução

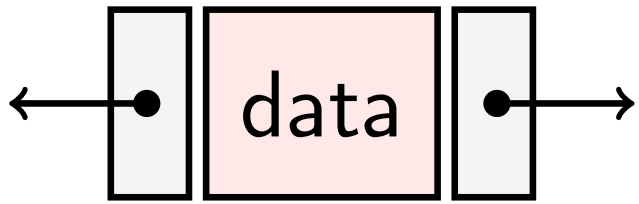
Solução: implementar o que chamamos de **lista duplamente encadeada**. Nela, cada elemento tem um ponteiro para o próximo elemento e um ponteiro para o elemento anterior.



Estrutura

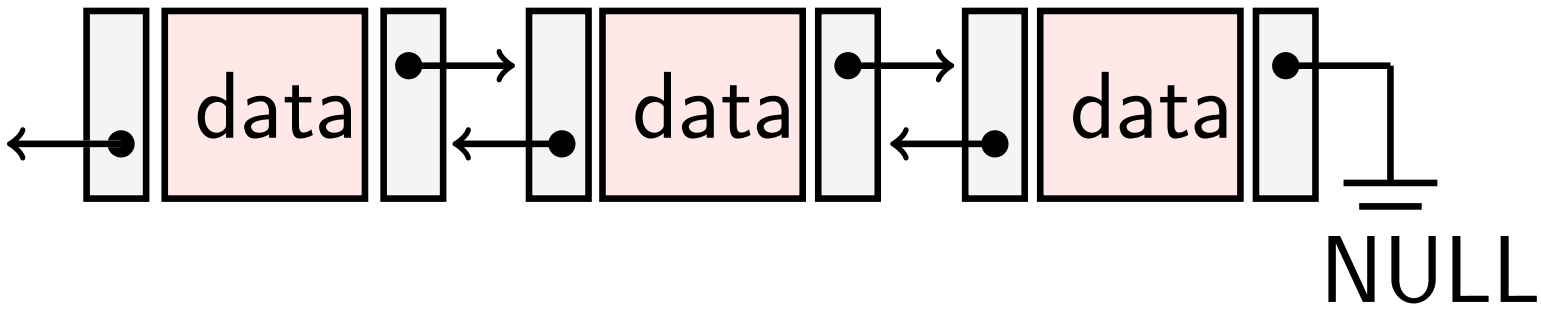
Observe a similaridade entre a estrutura de uma lista simples e uma lista dupla. A única diferença é a adição de mais um ponteiro para o elemento anterior.

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



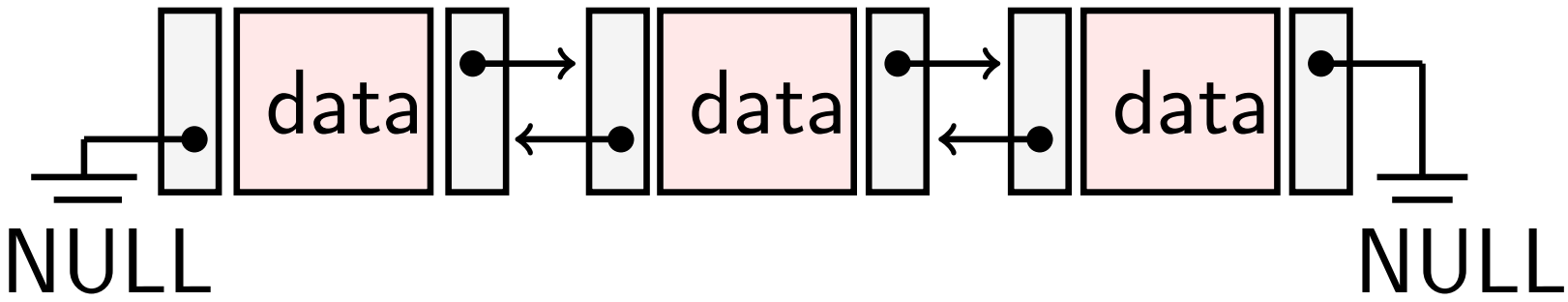
Listas duplamente encadeadas

O encadeamento em duas vias permite acessar ambos os elementos adjacentes: o próximo e o anterior, e desta forma, percorrer a lista em ordem inversa.



Listas duplamente encadeadas

Da mesma forma que o último elemento da lista tem como **next** o valor NULL, o primeiro elemento da lista não tem elemento anterior, e portanto o ponteiro **prev** é NULL.



Tipo abstrato de dados

Lista duplamente encadeada (**interface**)

create: inicializa uma estrutura de dados lista

insert-front: adiciona um nó na cabeça da lista

remove: remove um nó da lista

size: retorna o número de nós na lista

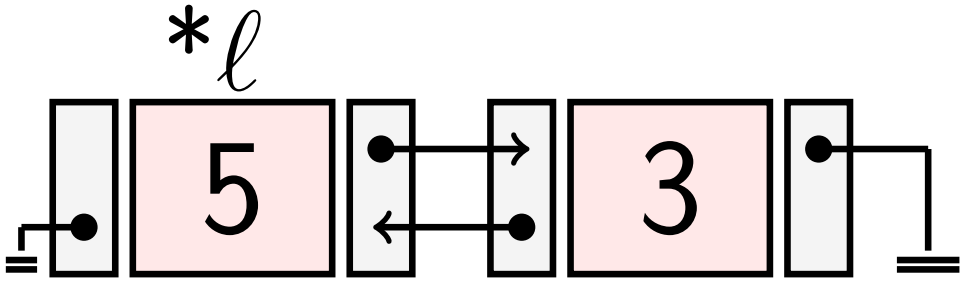
print: imprime todos os nós da lista

destroy: remove todos os nós da lista

Sumário

1. Introdução
2. Função: **inserir**
3. Função: **inserir (funcionamento)**
4. Função: **remover**

Inserir (início)

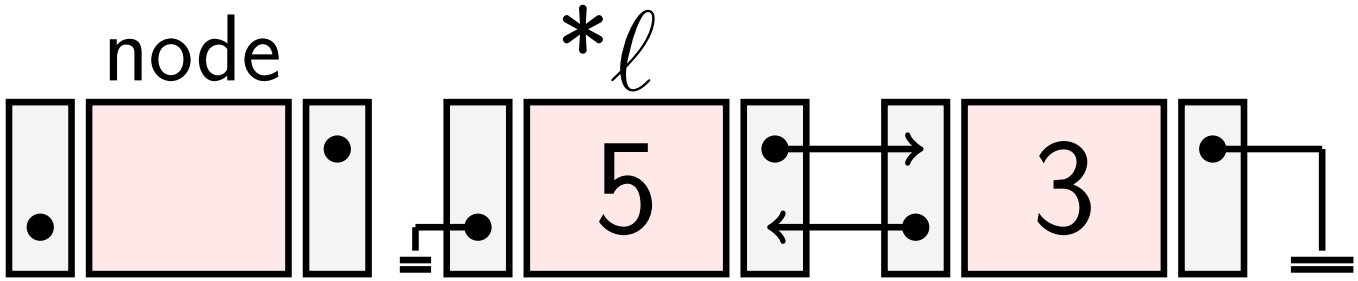


1: **List* insert-front** (List *ℓ, int elem)

2: List* node = (List*)malloc(sizeof(List));
3: node→data = elem;
4: node→next = ℓ;
5: node→prev = NULL;
6: **if** ($\ell \neq$ NULL) { ℓ →prev = node; }
7: **return** node;

Inserir: 7

Inserir (início)

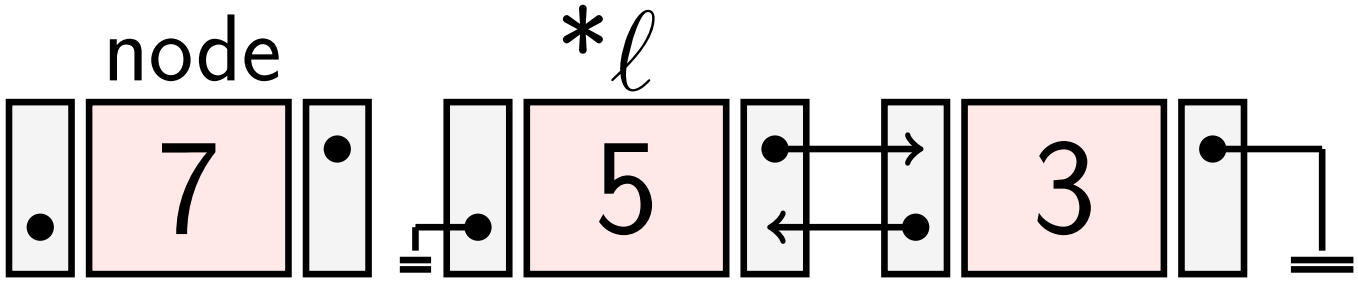


1: **List* insert-front** (List *ℓ, int elem)

2: List* node = (List*)malloc(sizeof(List));
3: node→data = elem;
4: node→next = ℓ;
5: node→prev = NULL;
6: **if** ($\ell \neq$ NULL) { ℓ →prev = node; }
7: **return** node;

Inserir: 7

Inserir (início)

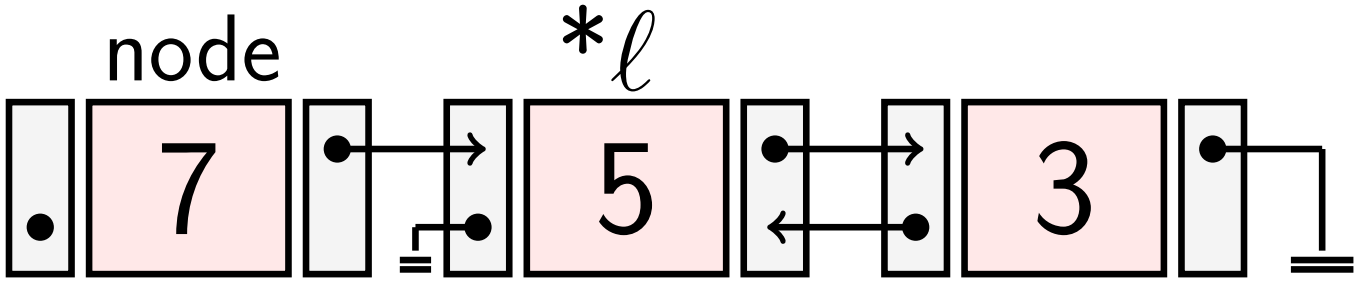


1: **List* insert-front** (List *ℓ, int elem)

2: List* node = (List*)malloc(sizeof(List));
3: node→data = elem;
4: node→next = ℓ;
5: node→prev = NULL;
6: **if** ($\ell \neq$ NULL) { ℓ →prev = node; }
7: **return** node;

Inserir: 7

Inserir (início)

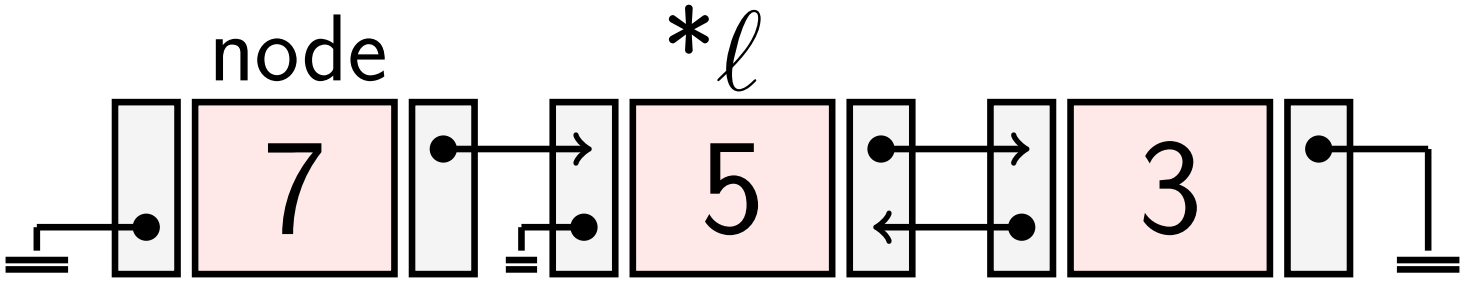


1: **List* insert-front** (List *ℓ, int elem)

2: List* node = (List*)malloc(sizeof(List));
3: node→data = elem;
4: node→next = ℓ;
5: node→prev = NULL;
6: **if** ($\ell \neq$ NULL) { ℓ →prev = node; }
7: **return** node;

Inserir: 7

Inserir (início)

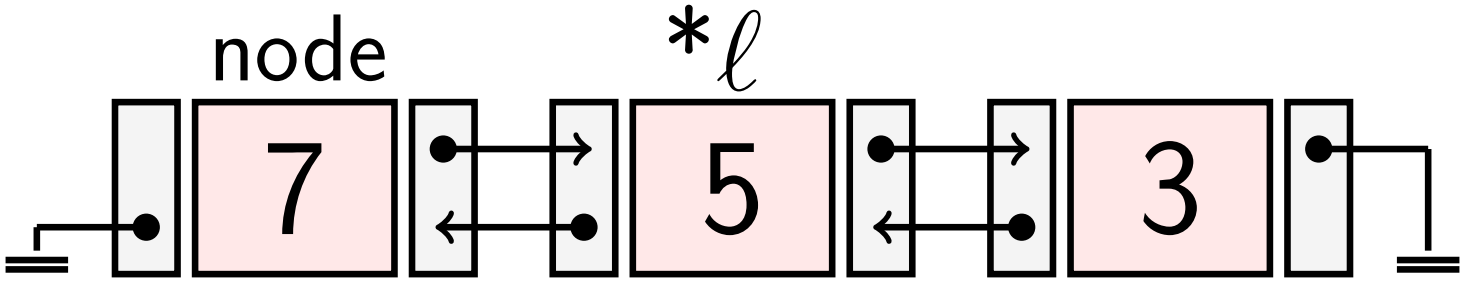


1: **List* insert-front** (List *ℓ, int elem)

2: List* node = (List*)malloc(sizeof(List));
3: node→data = elem;
4: node→next = ℓ;
5: node→prev = NULL;
6: **if** ($\ell \neq$ NULL) { ℓ →prev = node; }
7: **return** node;

Inserir: 7

Inserir (início)

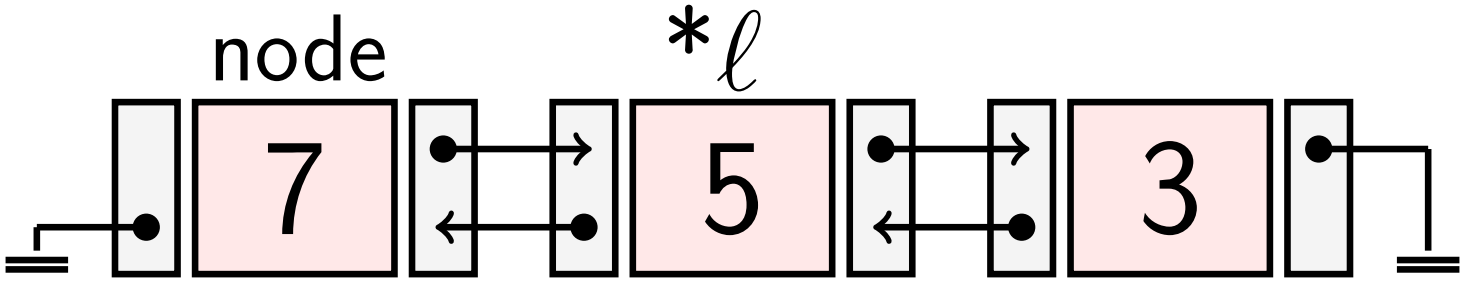


1: **List* insert-front** (List *ℓ, int elem)

2: List* node = (List*)malloc(sizeof(List));
3: node→data = elem;
4: node→next = ℓ;
5: node→prev = NULL;
6: **if** ($\ell \neq$ NULL) { ℓ →prev = node; }
7: **return** node;

Inserir: 7

Inserir (início)



1: **List* insert-front** (List *ℓ, int elem)

2: List* node = (List*)malloc(sizeof(List));
3: node→data = elem;
4: node→next = ℓ;
5: node→prev = NULL;
6: **if** ($\ell \neq$ NULL) { ℓ →prev = node; }
7: **return** node;

Inserir: 7

Sumário

1. Introdução
2. Função: **inserir**
3. Função: **inserir (funcionamento)**
4. Função: **remover**

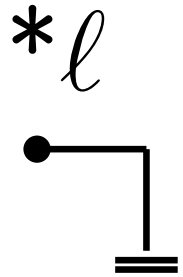
Inserir (início)

Importante: a função **main** sempre (e somente) guarda a **cabeça** da lista.

```
1: int main (void)
---------------------------------------
2:     List *ℓ = create ();
3:     ℓ = insert-front (ℓ, 3);
4:     ℓ = insert-front (ℓ, 5);
5:     ℓ = insert-front (ℓ, 7);
6: return 0
```

Inserir (início)

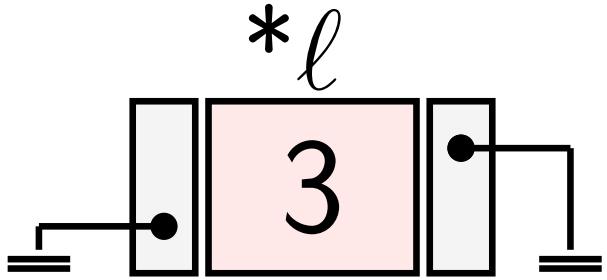
Importante: a função **main** sempre (e somente) guarda a **cabeça** da lista.




```
1: int main (void)
---------------------------------------
2:     List *ℓ = create ();
3:     ℓ = insert-front (ℓ, 3);
4:     ℓ = insert-front (ℓ, 5);
5:     ℓ = insert-front (ℓ, 7);
6: return 0
```

Inserir (início)

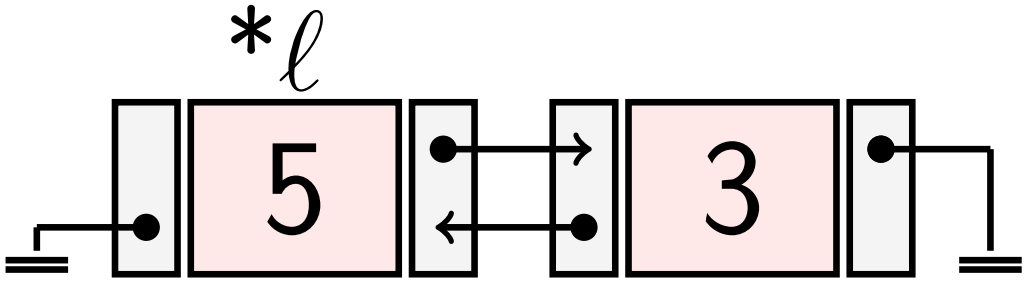
Importante: a função **main** sempre (e somente) guarda a **cabeça** da lista.




```
1: int main (void)
---------------------------------------
2:     List *ℓ = create ();
3:     ℓ = insert-front (ℓ, 3);
4:     ℓ = insert-front (ℓ, 5);
5:     ℓ = insert-front (ℓ, 7);
6: return 0
```

Inserir (início)

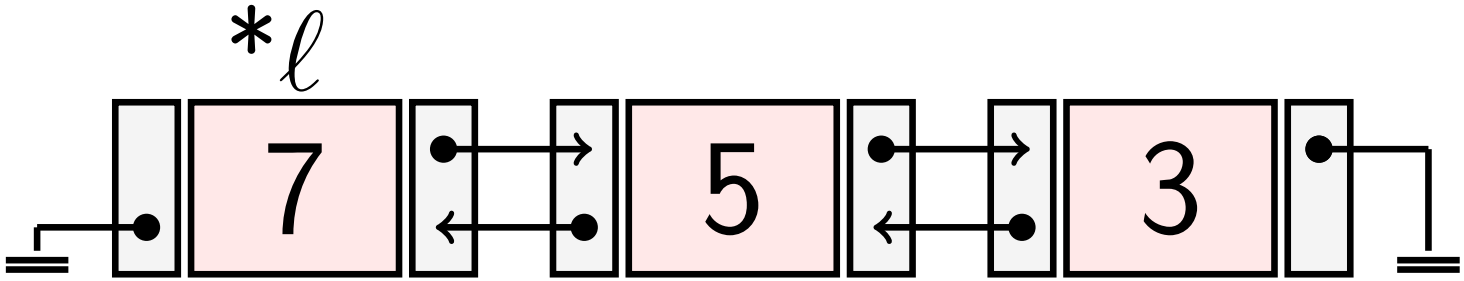
Importante: a função **main** sempre (e somente) guarda a **cabeça** da lista.




```
1: int main (void)
---------------------------------------
2:     List *ℓ = create ();
3:     ℓ = insert-front (ℓ, 3);
4:     ℓ = insert-front (ℓ, 5);
5:     ℓ = insert-front (ℓ, 7);
6: return 0
```

Inserir (início)

Importante: a função **main** sempre (e somente) guarda a **cabeça** da lista.

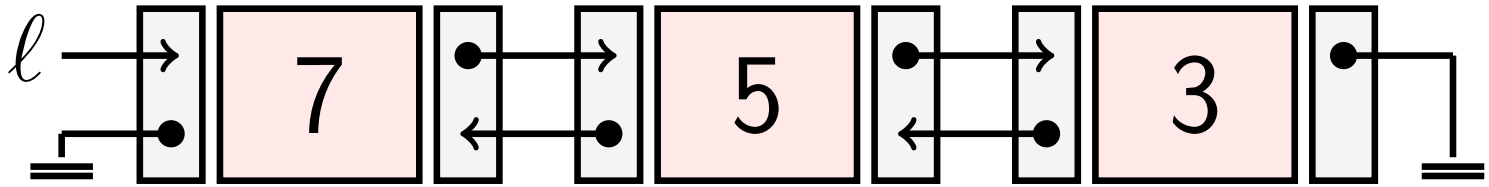



```
1: int main (void)
---------------------------------------
2:     List *ℓ = create ();
3:     ℓ = insert-front (ℓ, 3);
4:     ℓ = insert-front (ℓ, 5);
5:     ℓ = insert-front (ℓ, 7);
6: return 0
```

Sumário

1. Introdução
2. Função: **inserir**
3. Função: **inserir (funcionamento)**
4. Função: **remover**

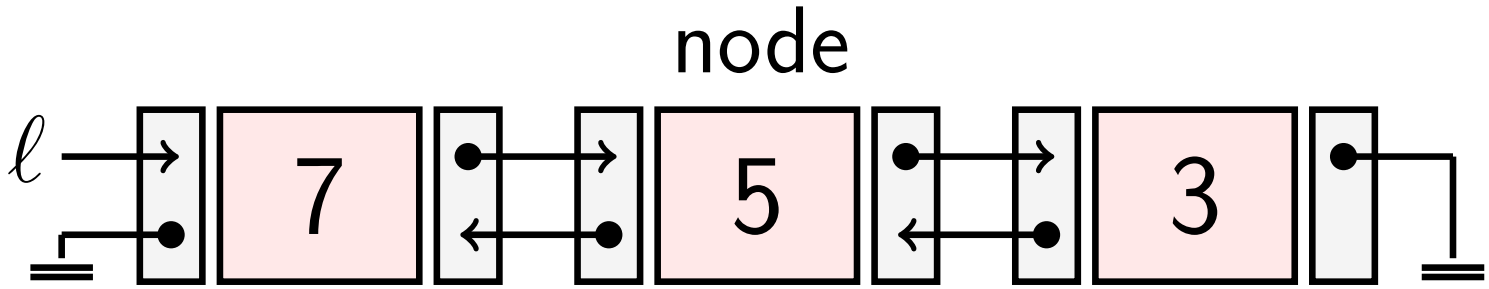
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 5

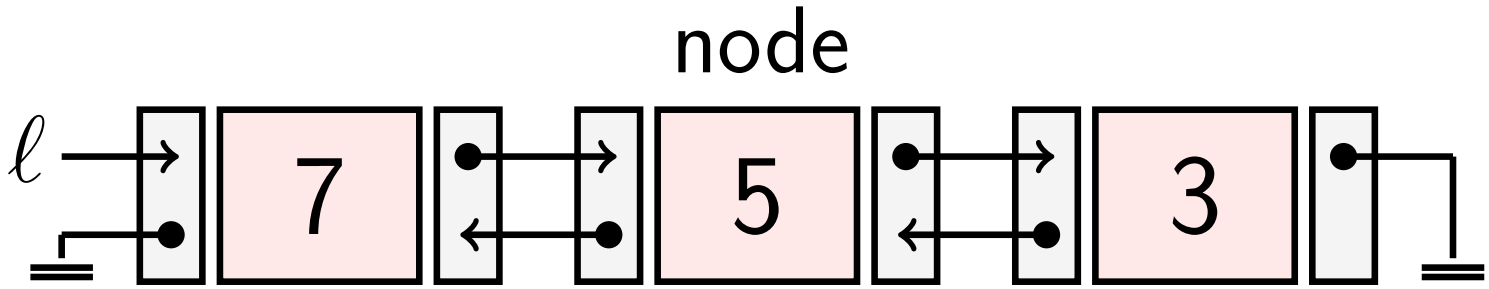
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 5

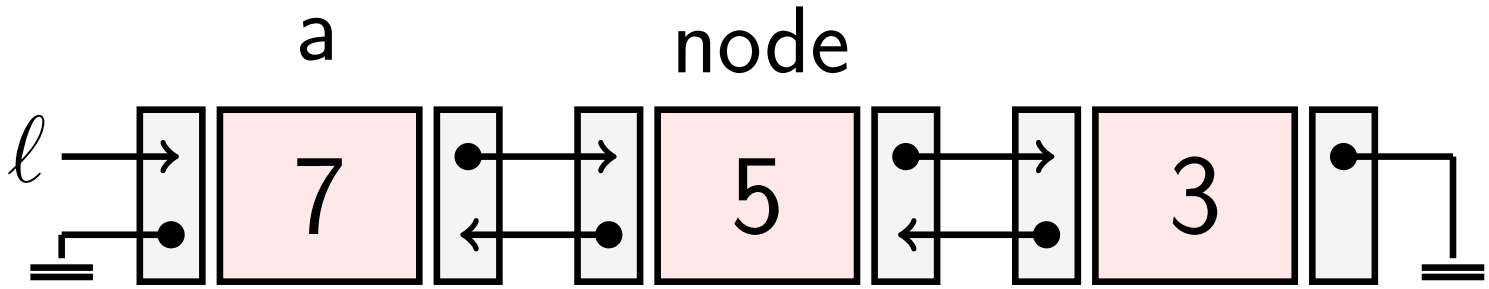
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 5

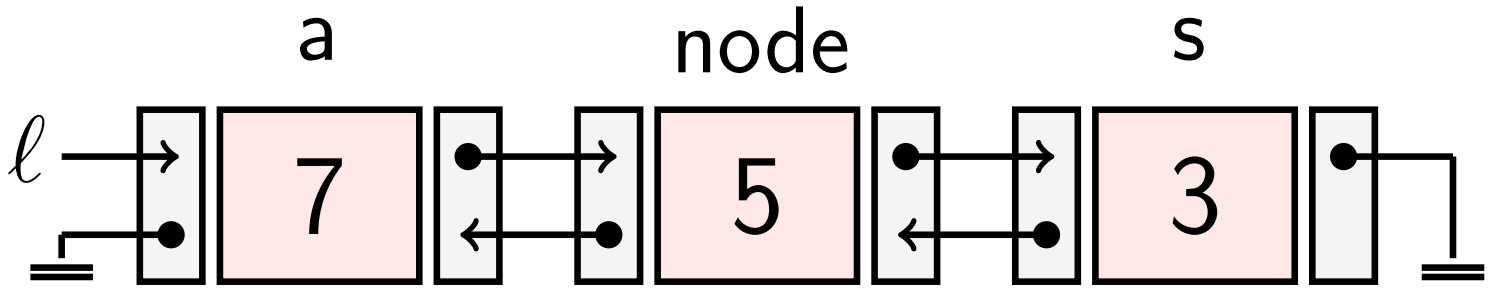
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 5

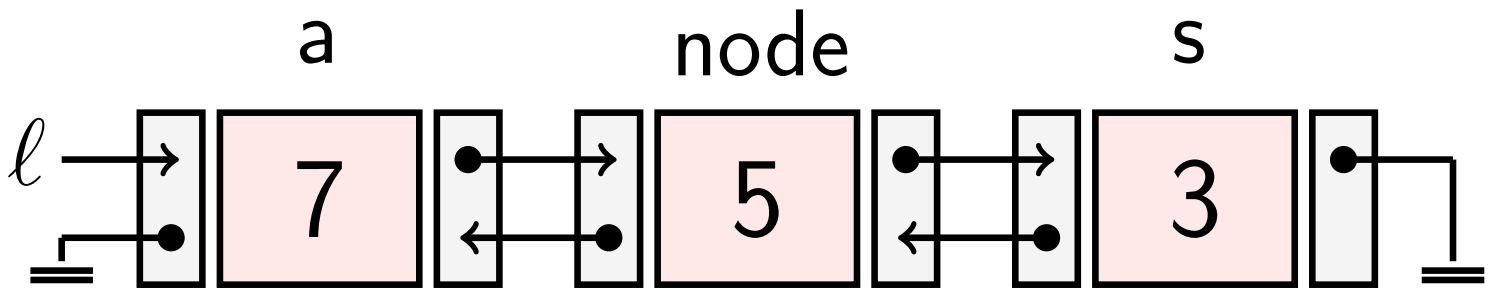
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 5

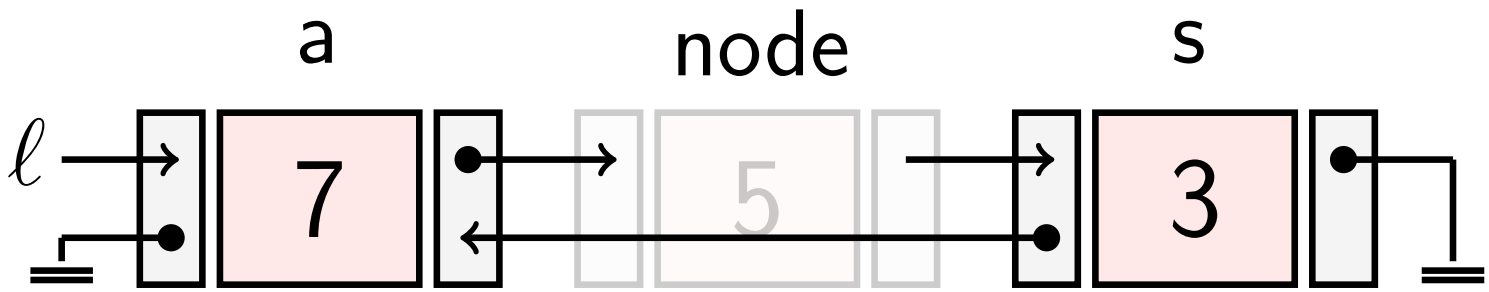
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 5

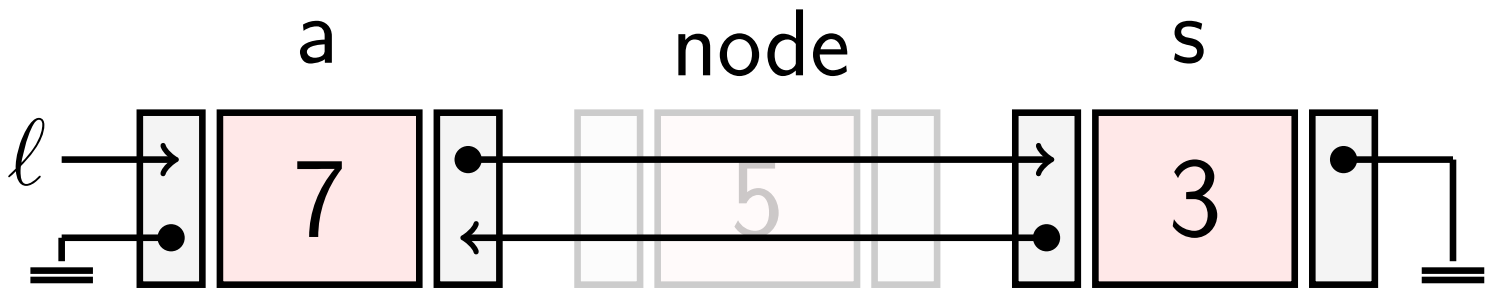
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 5

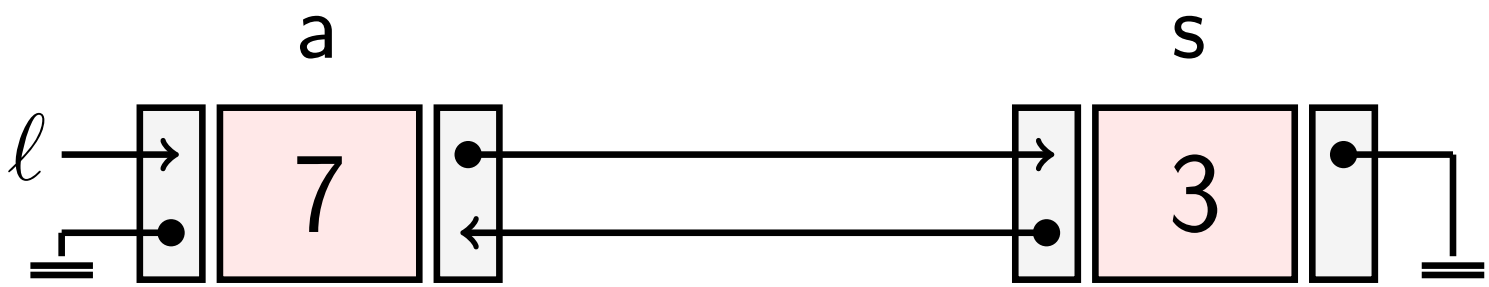
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 5

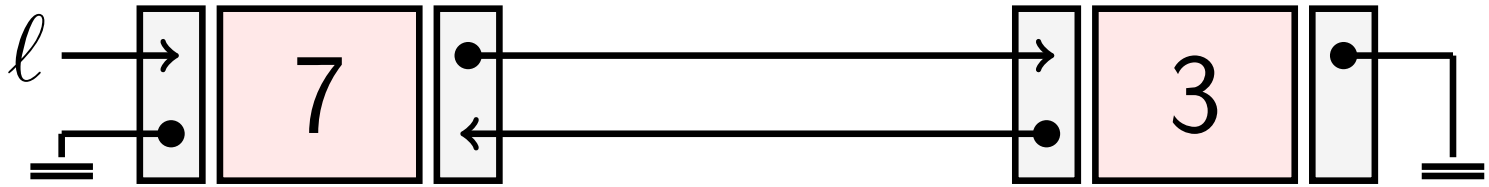
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 5

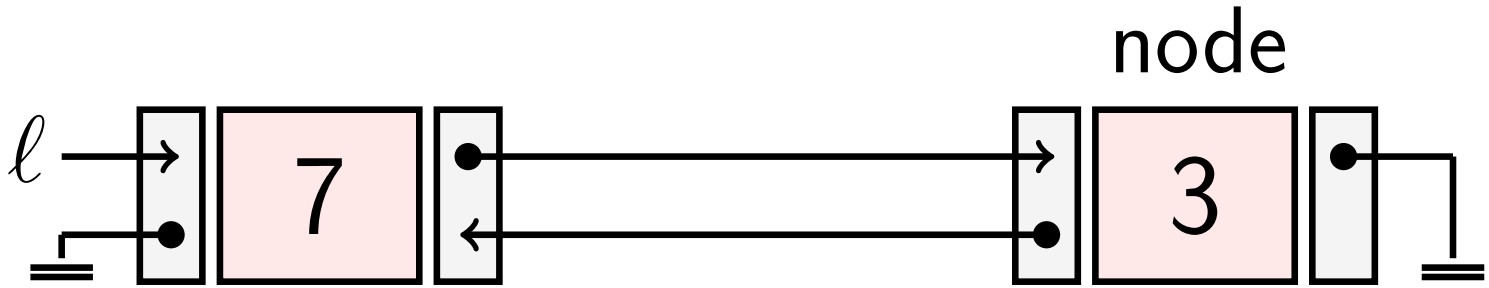
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 3

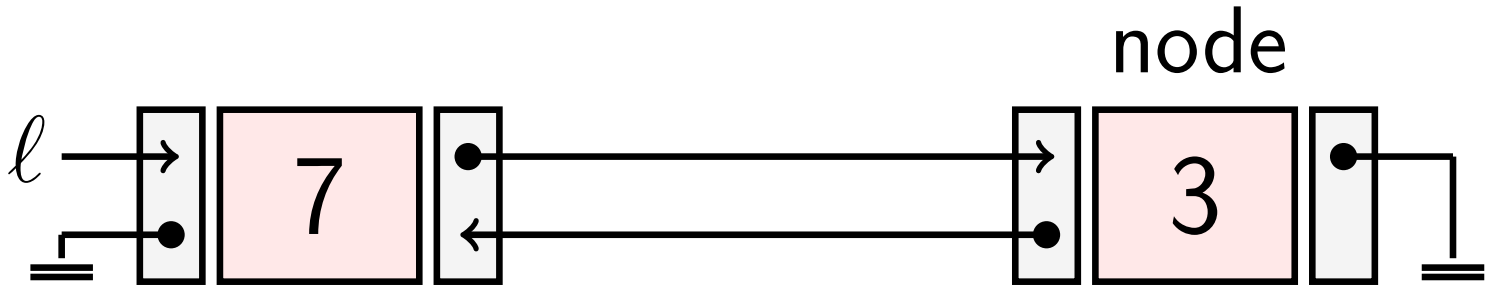
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 3

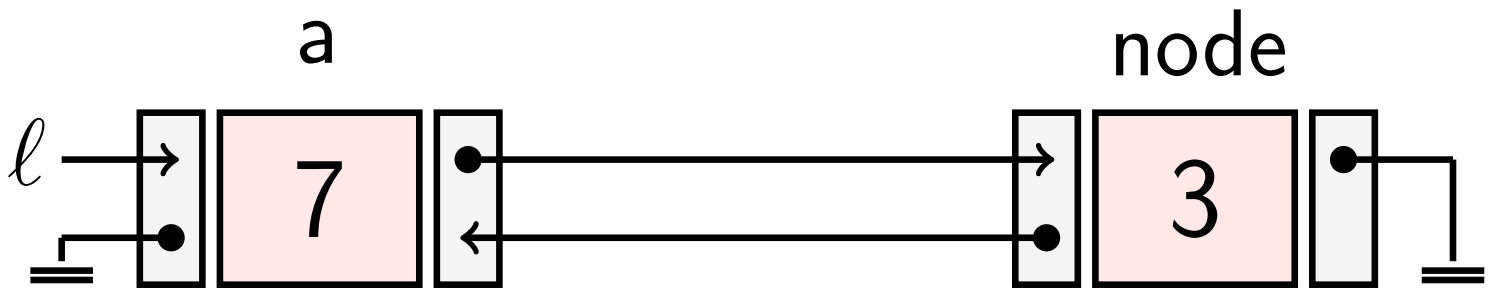
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 3

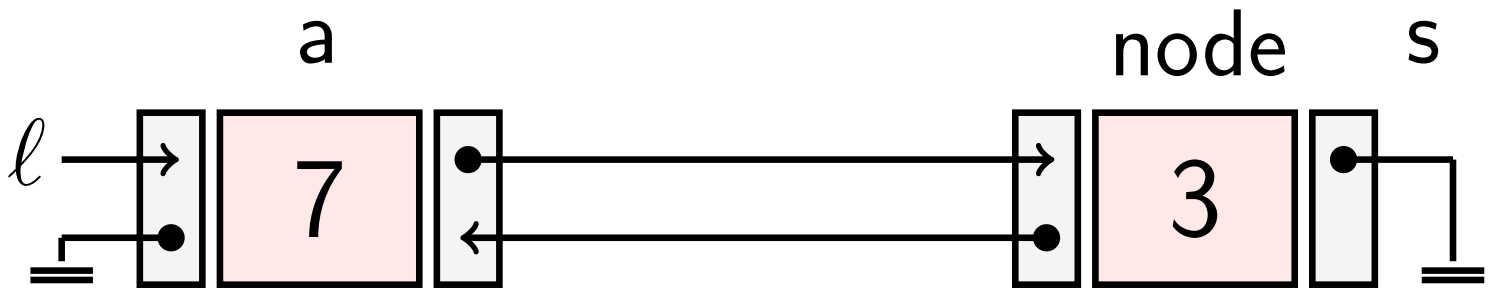
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 3

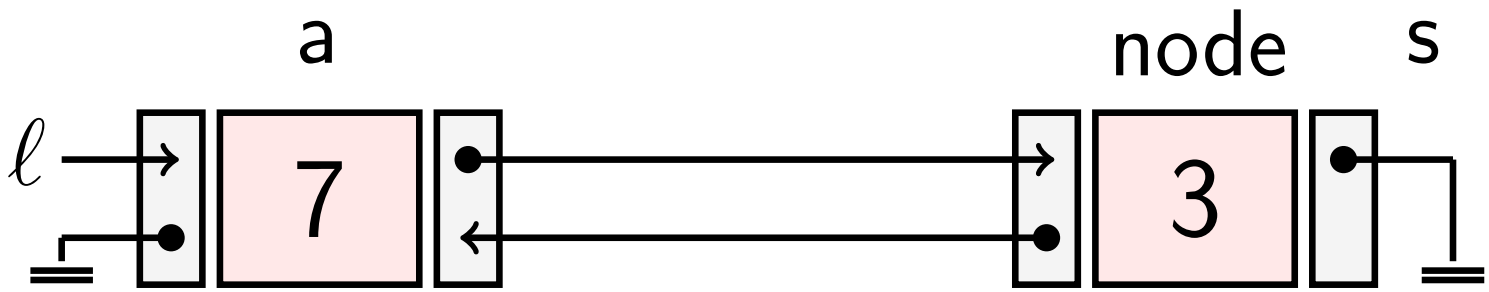
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 3

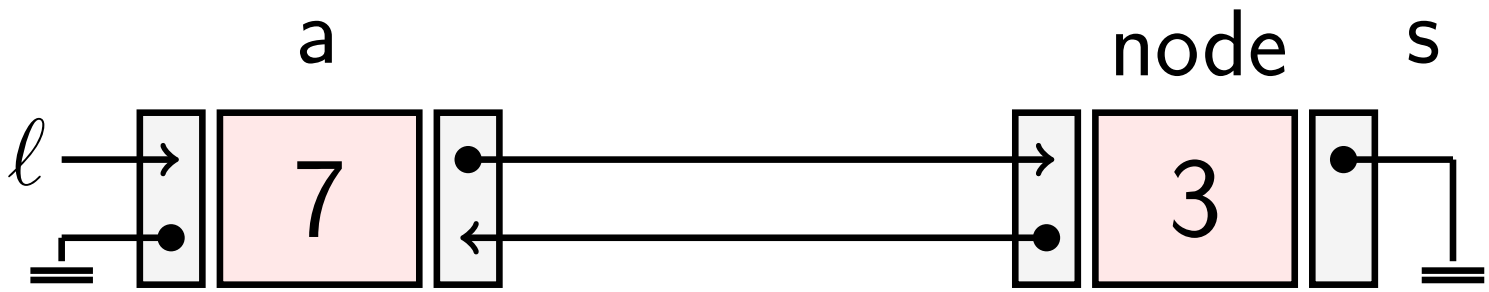
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 3

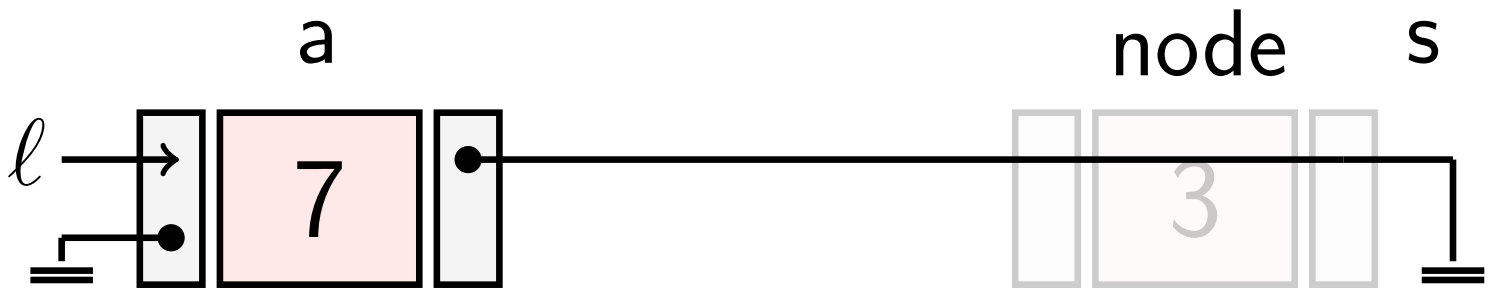
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 3

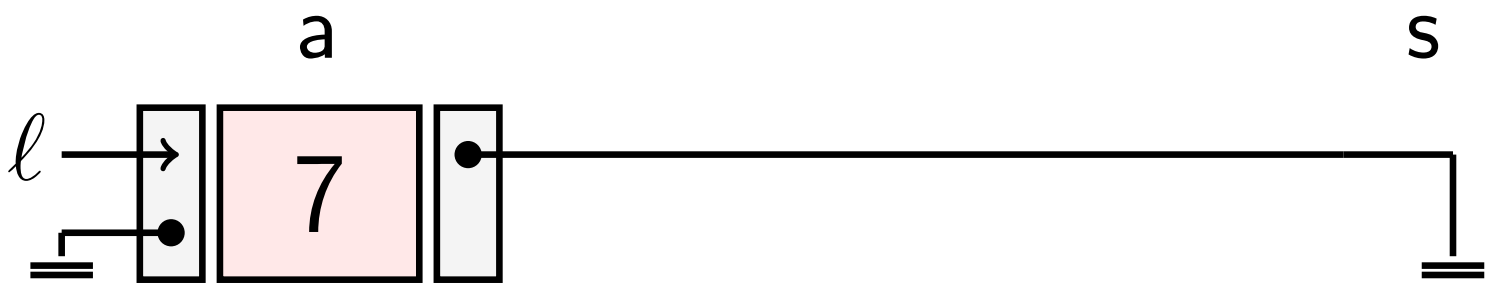
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 3

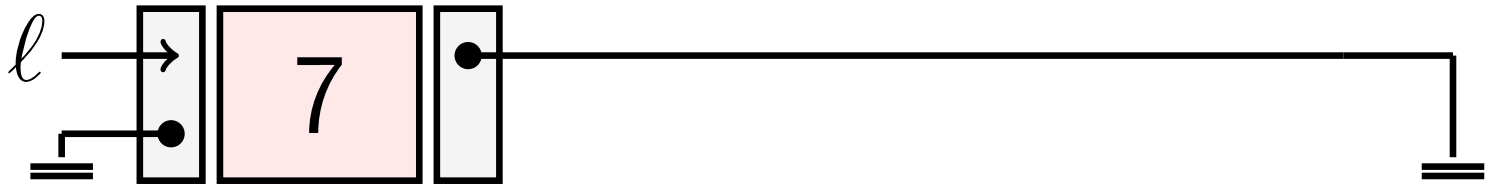
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 3

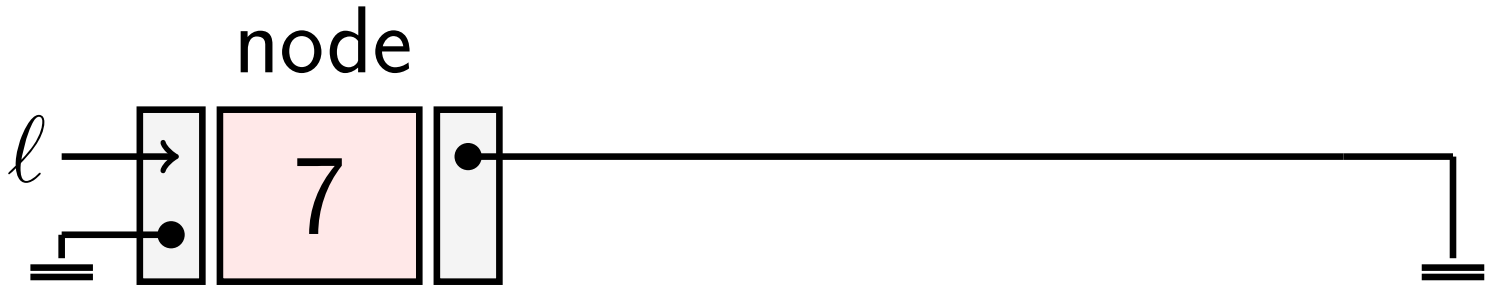
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 7

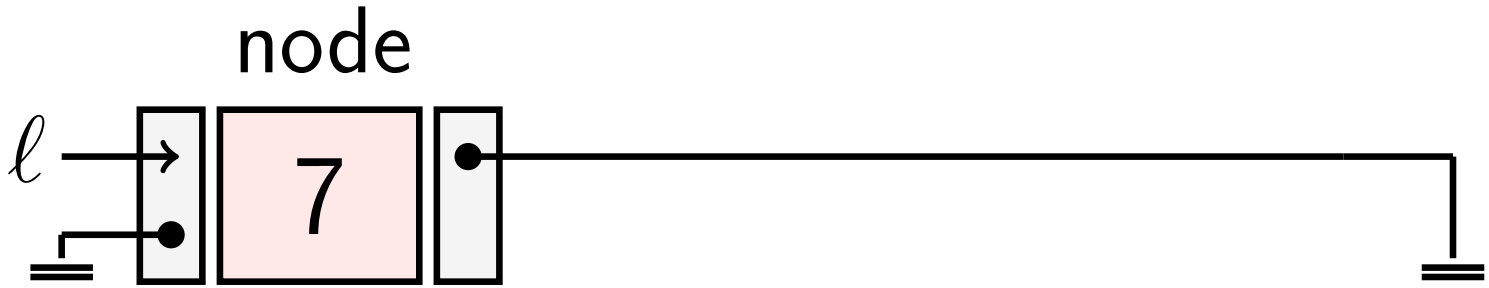
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 7

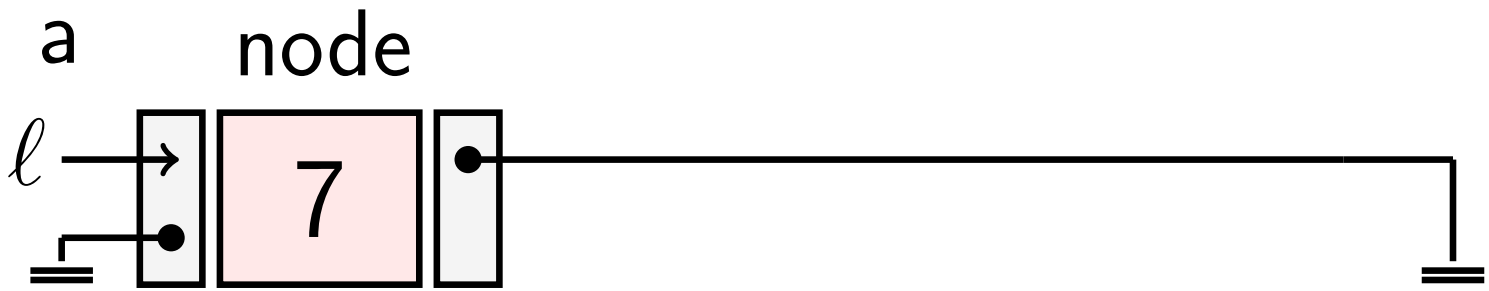
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 7

Remover

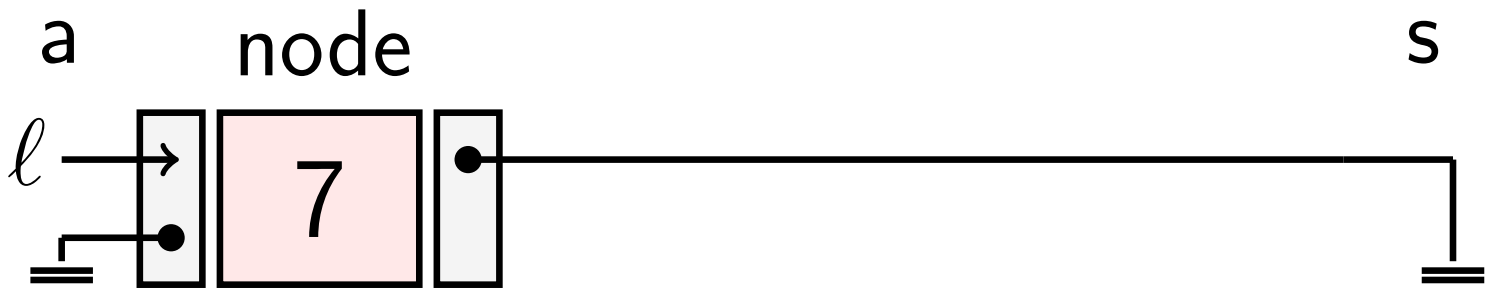


1: **List* remove** (List *ℓ, int elem)

```
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 7

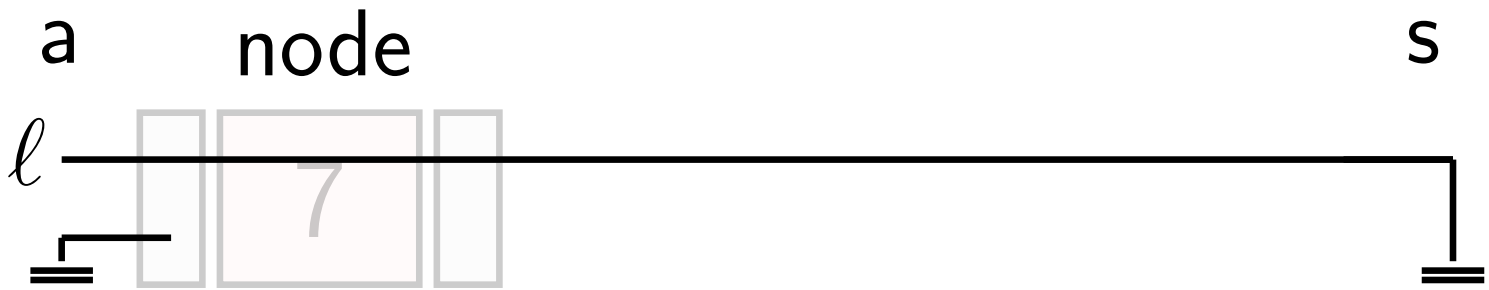
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 7

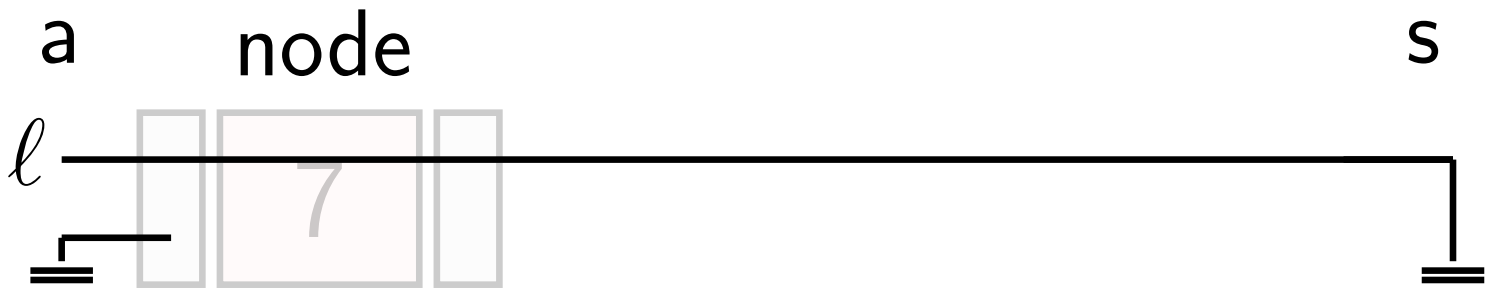
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 7

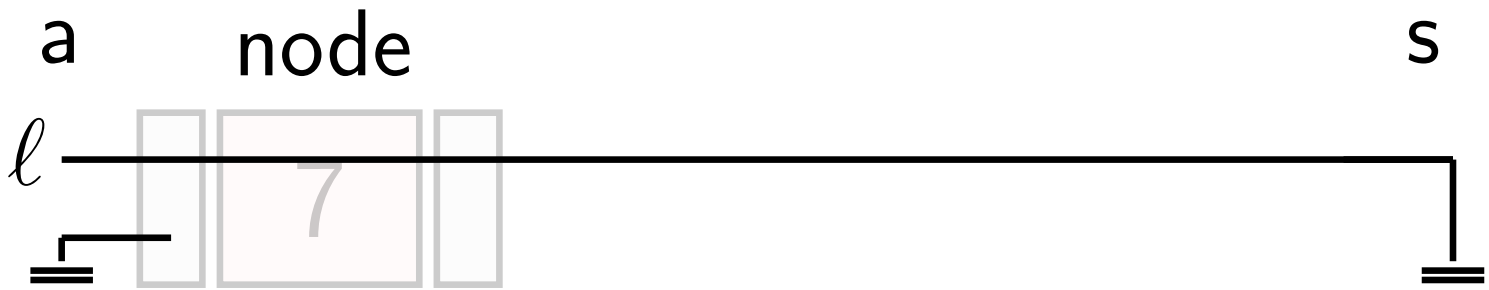
Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 7

Remover

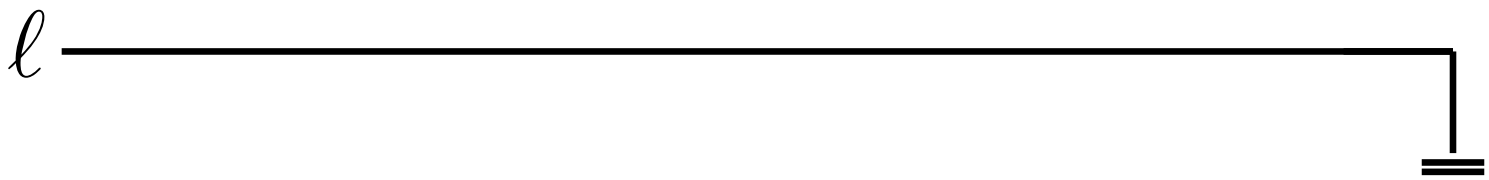


1: **List* remove** (List *ℓ, int elem)

```
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 7

Remover




```
1:  List* remove (List *ℓ, int elem)
2:      List *node = search (ℓ, elem); /*busca pelo nó!*/
3:      if (node = NULL) { return ℓ; }
4:      List *a = node→prev;     /*antecessor!*/
5:      List *s = node→next;     /*sucessor!*/
6:      if (ℓ = node) { ℓ = s; }
7:      if (s ≠ NULL) { s→prev = a; }
8:      if (a ≠ NULL) { a→next = s; }
9:      free (node);
10: return ℓ;
```

Remover: 7