Estrutura de Dados I

Estrutura de dados: lista duplamente encadeada

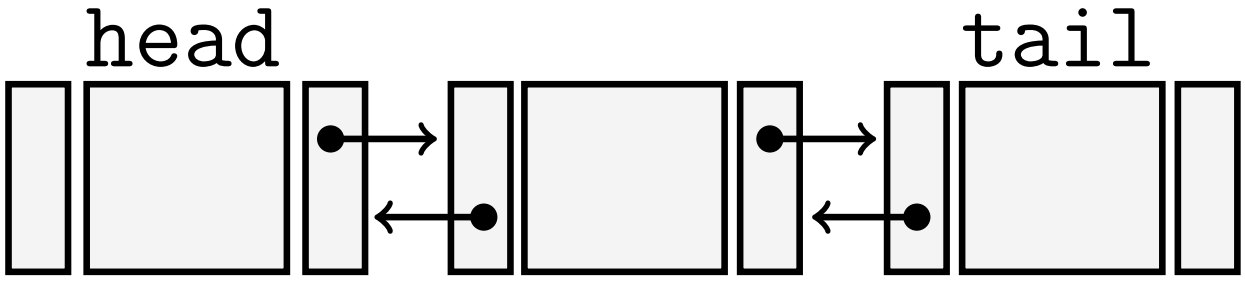
Prof. Rodrigo Minetto

Universidade Tecnológica Federal do Paraná

Sumário

Introdução

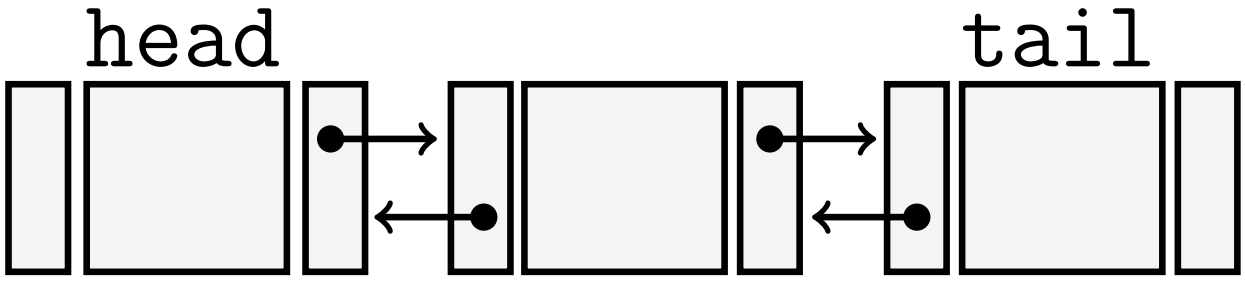
A estrutura de dados **lista** com **encadeamento duplo** é excelente para construir diversas outras estrutura de dados.



Introdução

A estrutura de dados **lista** com **encadeamento duplo** é excelente para construir diversas outras estrutura de dados.

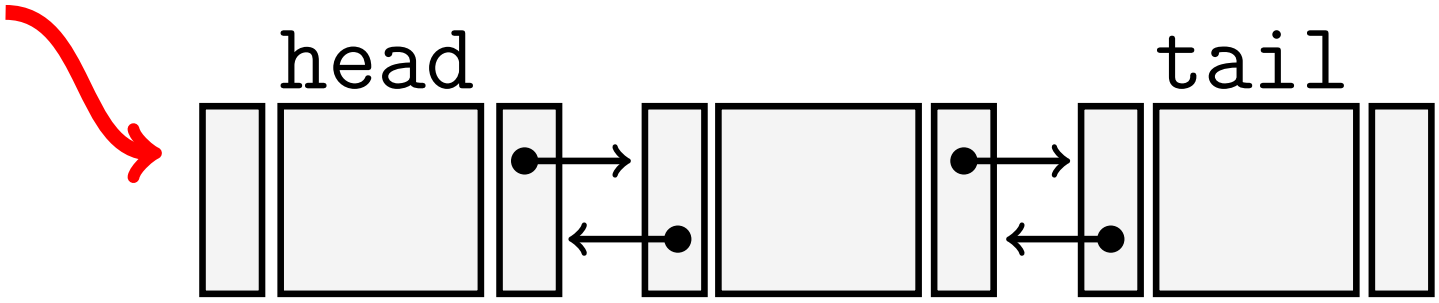
Pilha:



Introdução

A estrutura de dados **lista** com **encadeamento duplo** é excelente para construir diversas outras estrutura de dados.

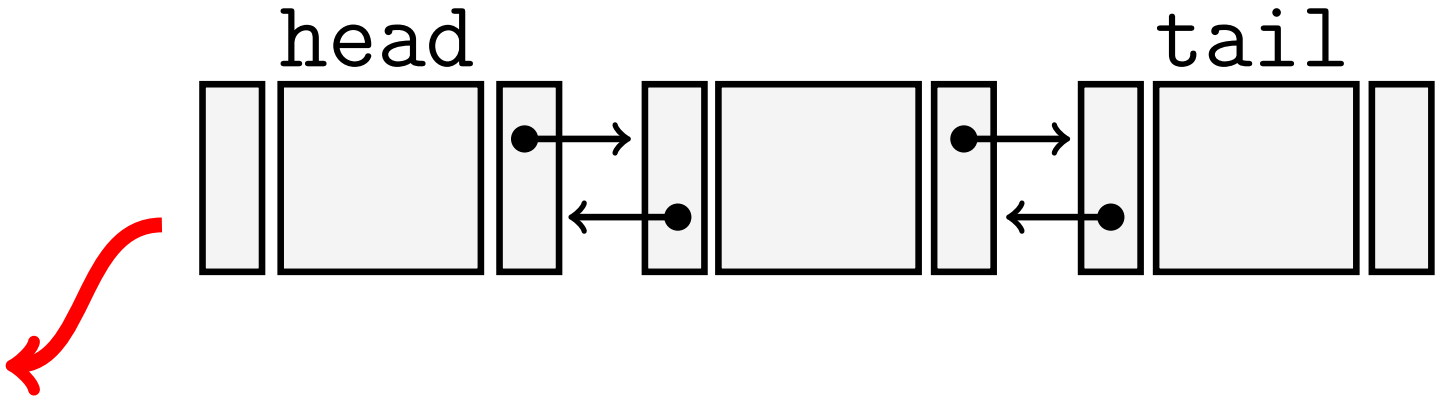
Pilha: **push** (cabeça da lista)



Introdução

A estrutura de dados **lista** com **encadeamento duplo** é excelente para construir diversas outras estrutura de dados.

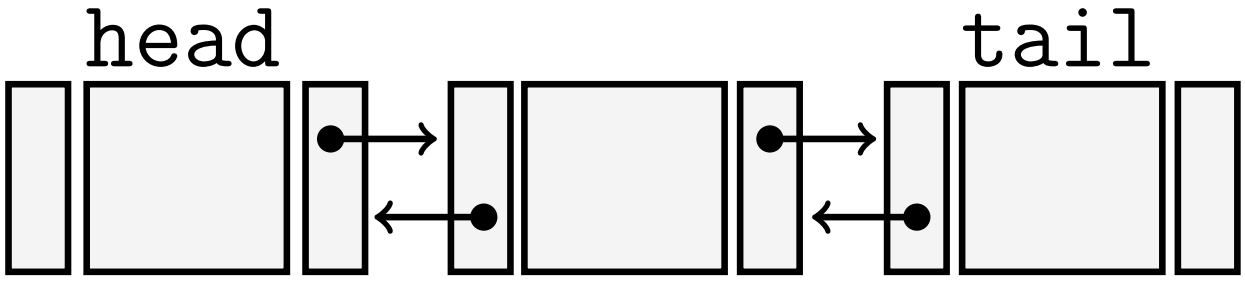
Pilha: **pop** (cabeça da lista)



Introdução

A estrutura de dados **lista** com **encadeamento duplo** é excelente para construir diversas outras estrutura de dados.

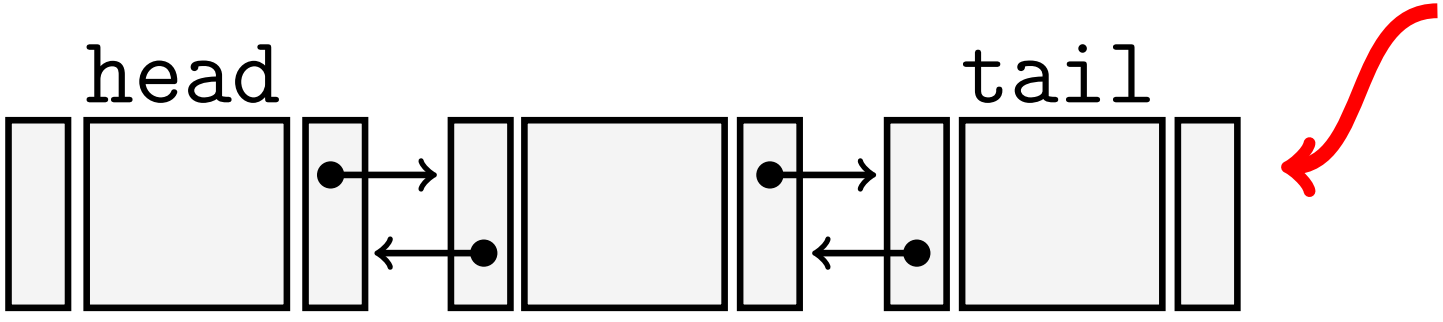
Fila:



Introdução

A estrutura de dados **lista** com **encadeamento duplo** é excelente para construir diversas outras estrutura de dados.

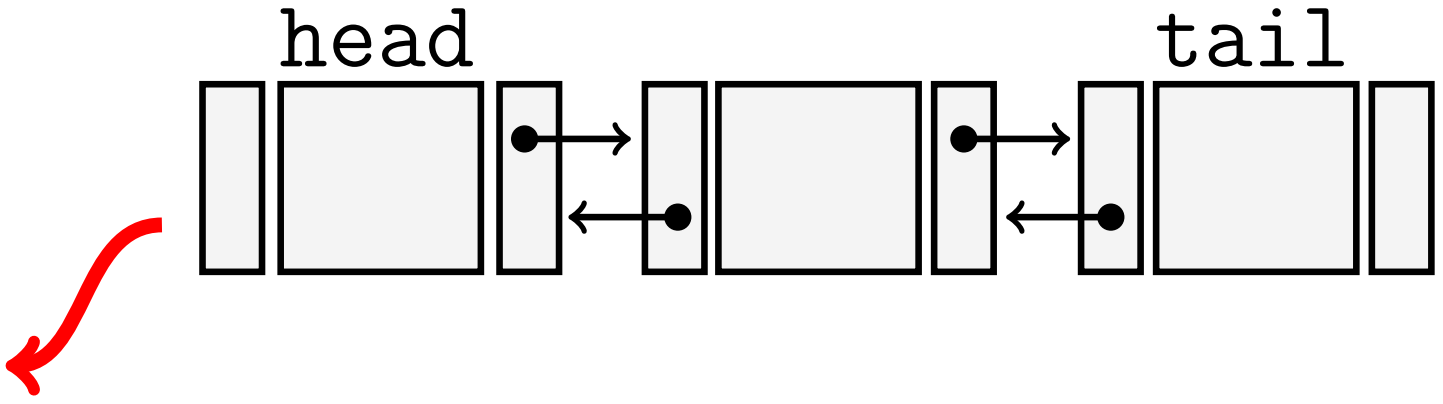
Fila: **enqueue** (cauda da lista)



Introdução

A estrutura de dados **lista** com **encadeamento duplo** é excelente para construir diversas outras estrutura de dados.

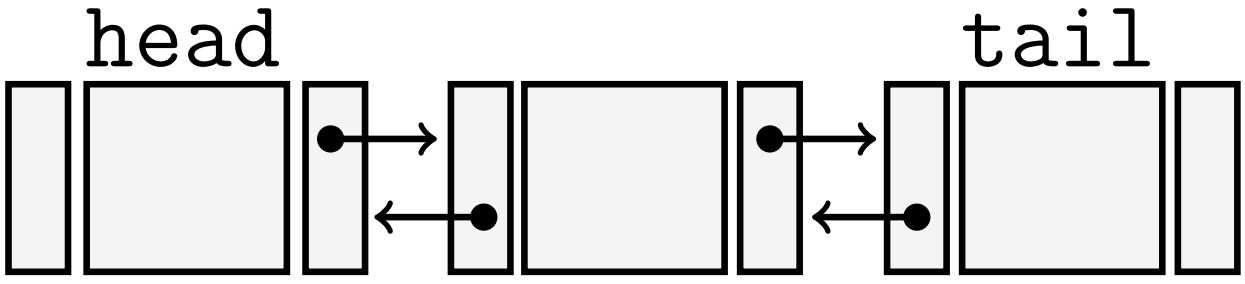
Fila: **dequeue** (cabeça da lista)



Introdução

A estrutura de dados **lista** com **encadeamento duplo** é excelente para construir diversas outras estrutura de dados.

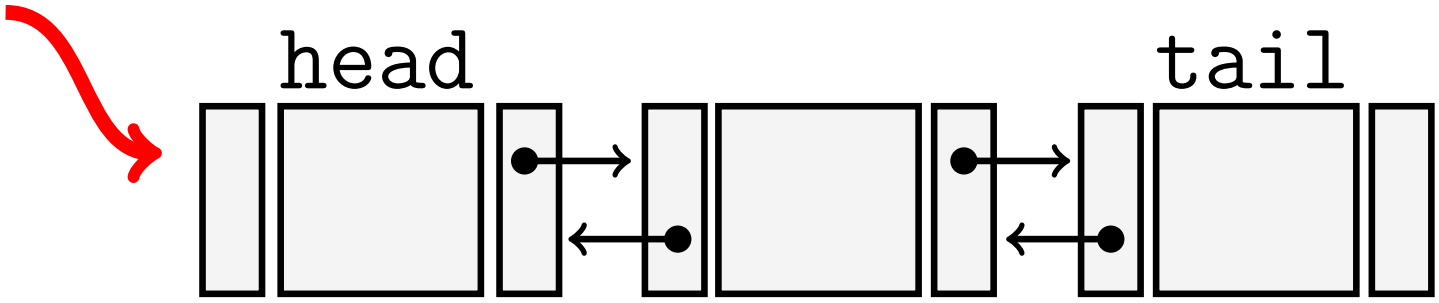
Deque:



Introdução

A estrutura de dados **lista** com **encadeamento duplo** é excelente para construir diversas outras estrutura de dados.

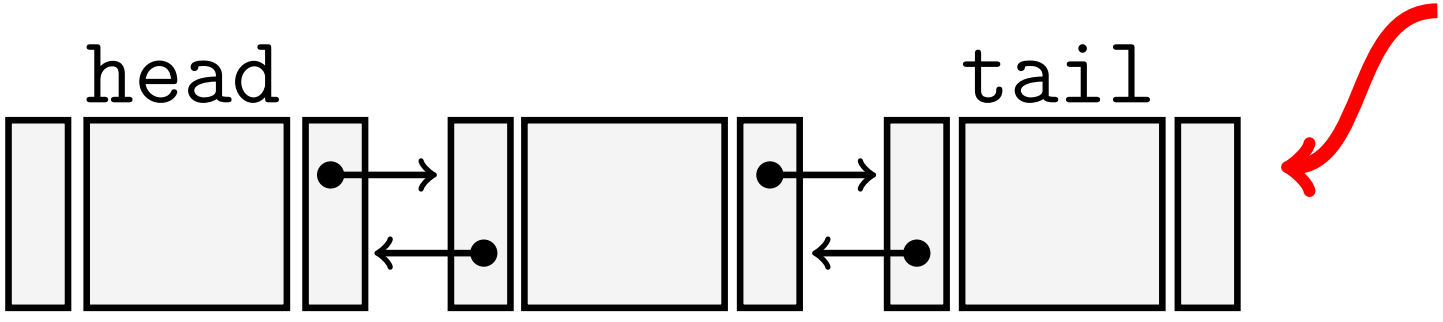
Deque: **insert** (cabeça da lista)



Introdução

A estrutura de dados **lista** com **encadeamento duplo** é excelente para construir diversas outras estrutura de dados.

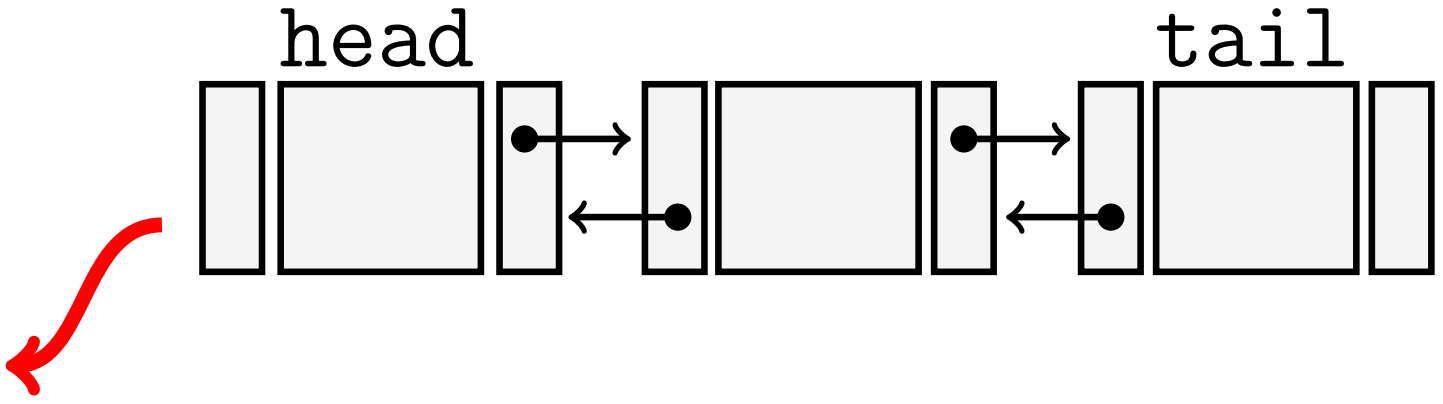
Deque: **insert** (cauda da lista)



Introdução

A estrutura de dados **lista** com **encadeamento duplo** é excelente para construir diversas outras estrutura de dados.

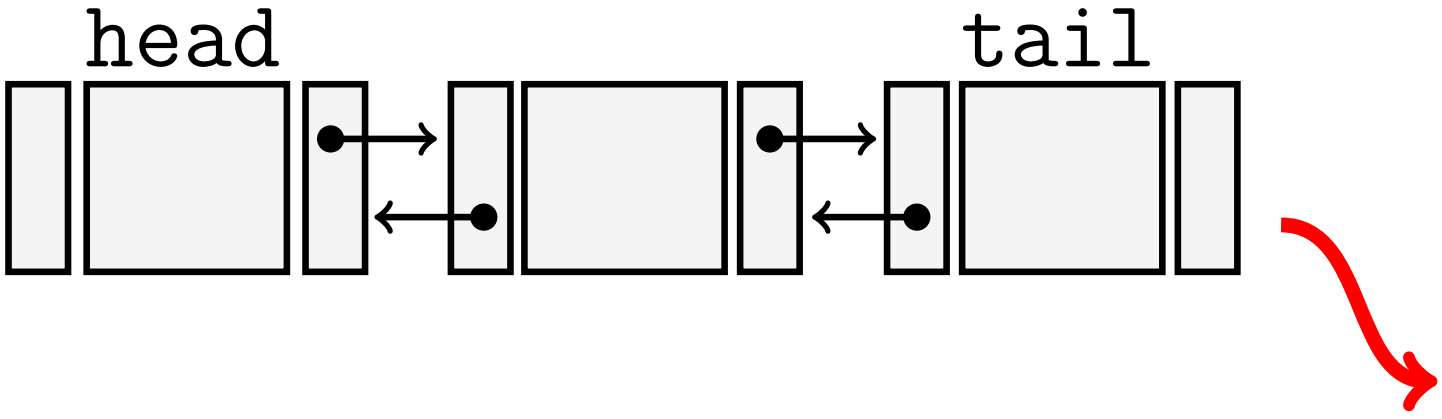
Deque: **remove** (cabeça da lista)



Introdução

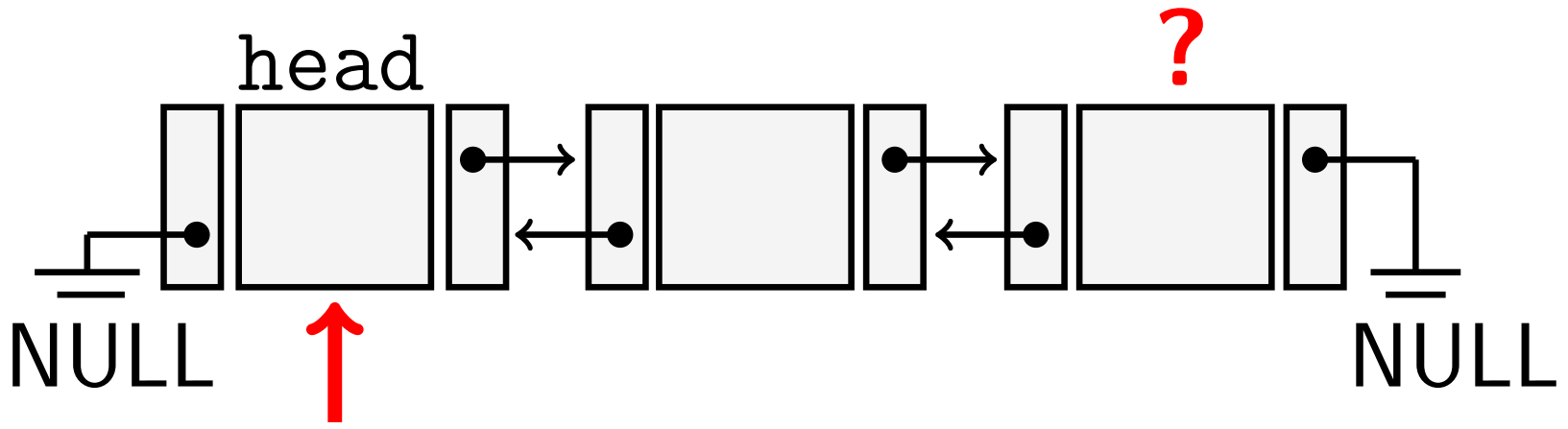
A estrutura de dados **lista** com **encadeamento duplo** é excelente para construir diversas outras estrutura de dados.

Deque: **remove** (cauda da lista)



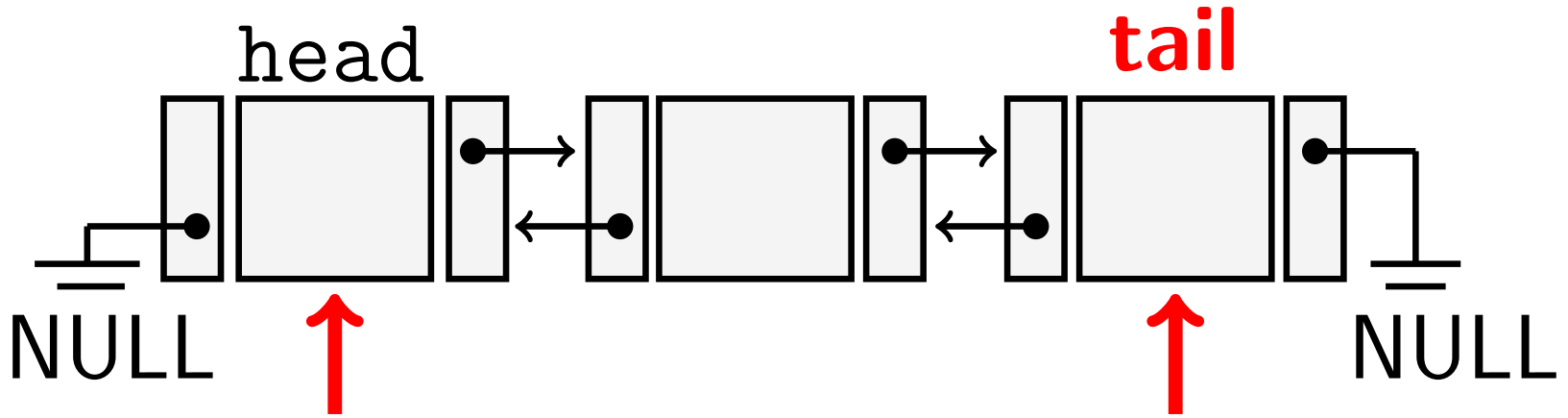
Introdução

No entanto, a lista com duplo encadeamento vista aula passada **não é eficiente** por exemplo para operações de cauda (complexidade $\mathcal{O}(n)$). Observe que o único ponteiro direto é para a cabeça da lista (**sem acesso direto a cauda**):



Introdução

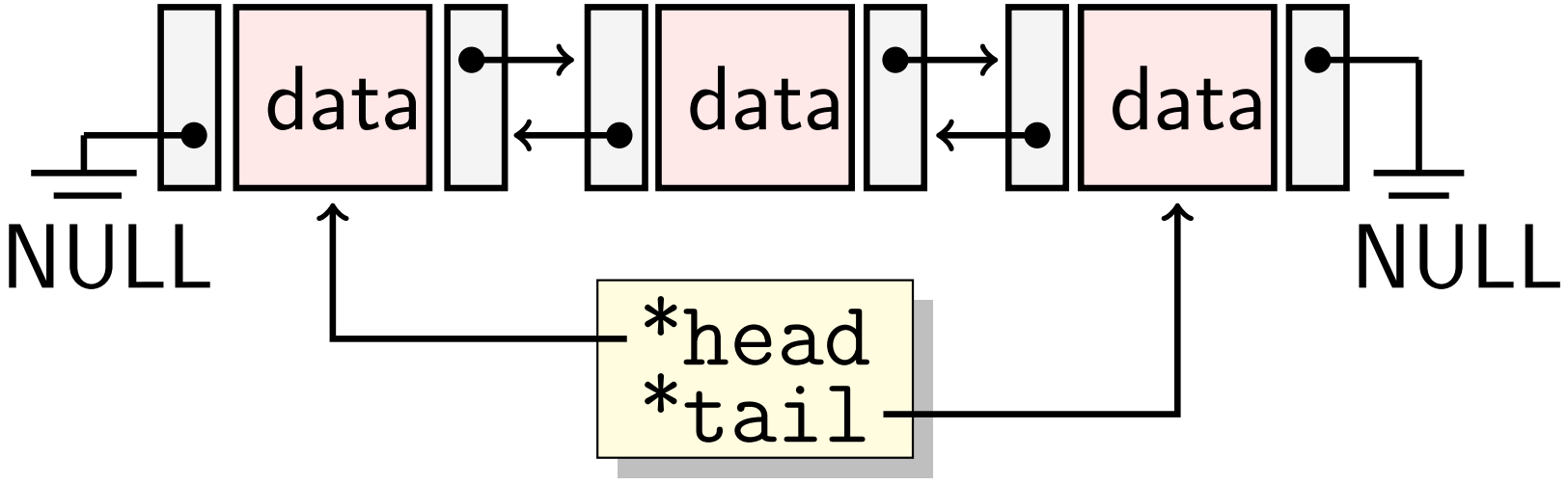
Solução: acrescentar e manipular um ponteiro com **acesso direto a cauda da lista**. A complexidade para operações de cauda (inserção, busca, remoção de elementos, ...) com esta otimização é $\mathcal{O}(1)$.



Estrutura

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

```
typedef struct {
  Node *head;
  Node *tail;
} List;
```



Tipo abstrato de dados

Lista duplamente encadeada (**interface**)

create: inicializa uma estrutura de dados lista

insert-front: adiciona um nó no início

insert-back: adiciona um nó no fim

remove-front: retira um nó do início

remove-back: retira um nó do fim

print-front-back: imprime nós do início p/ fim

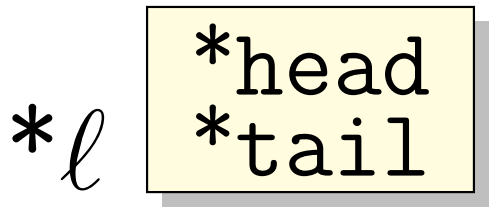
print-back-front: imprime nós do fim p/início

Sumário

Criar

```
1: List* create (void)
---
2:     List* ℓ = (List*)malloc(sizeof(List));
3:     ℓ →head = NULL;
4:     ℓ →tail = NULL;
5:     return ℓ;
```

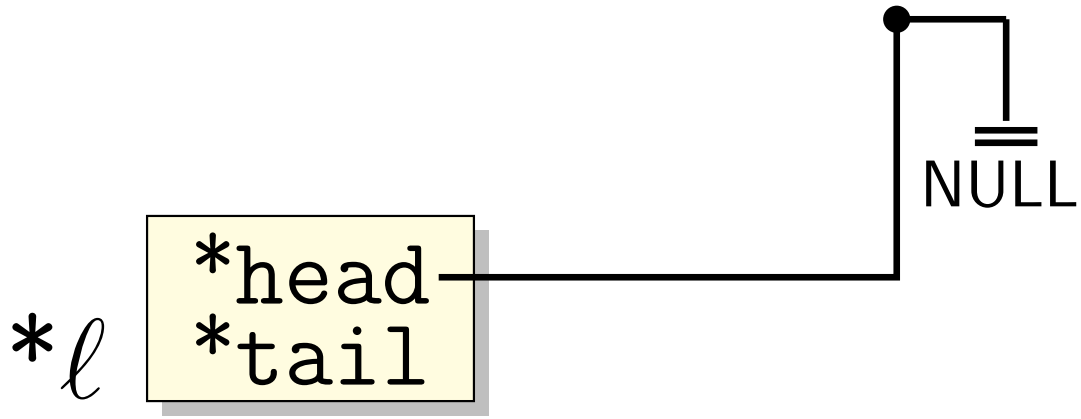
Criar



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

2: List* ℓ = (List*)malloc(sizeof(List));
3: $\ell \rightarrow$head = NULL;
4: $\ell \rightarrow$tail = NULL;
5: **return** ℓ;

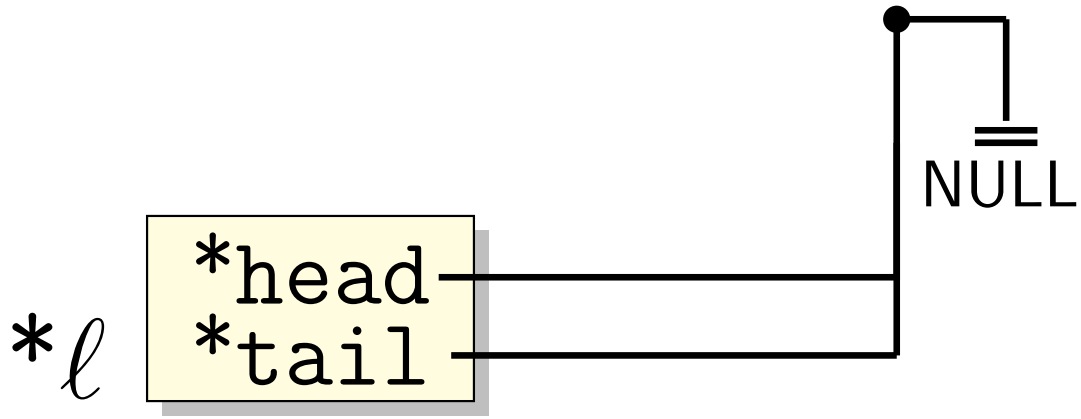
Criar



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

2: List* ℓ = (List*)malloc(sizeof(List));
3: $\ell \rightarrow$head = NULL;
4: $\ell \rightarrow$tail = NULL;
5: **return** ℓ;

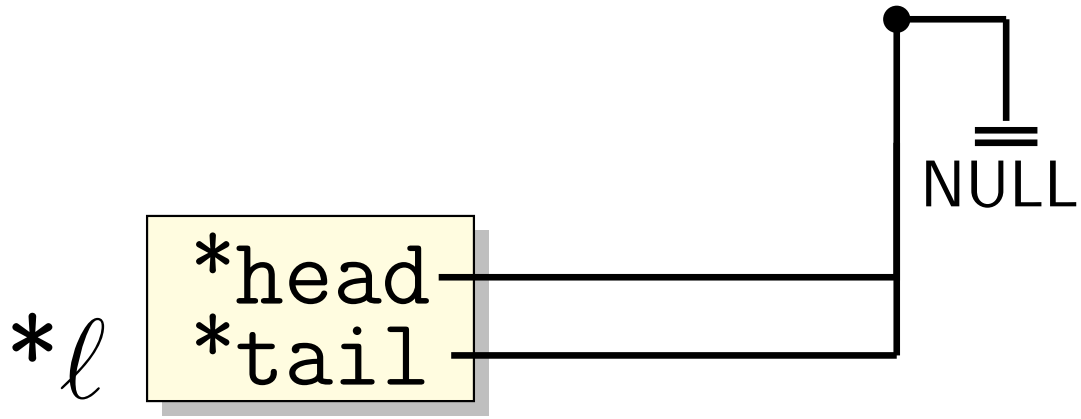
Criar



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

2: List* ℓ = (List*)malloc(sizeof(List));
3: $\ell \rightarrow$head = NULL;
4: $\ell \rightarrow$tail = NULL;
5: **return** ℓ;

Criar

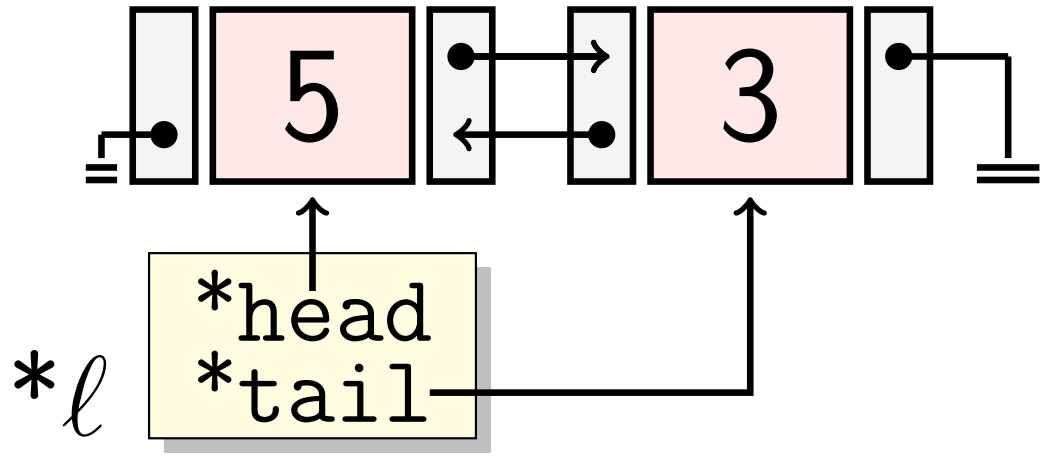


1: **List* create** (void)

2: List* ℓ = (List*)malloc(sizeof(List));
3: $\ell \rightarrow$head = NULL;
4: $\ell \rightarrow$tail = NULL;
5: **return** ℓ;

Sumário

Inserir (início)

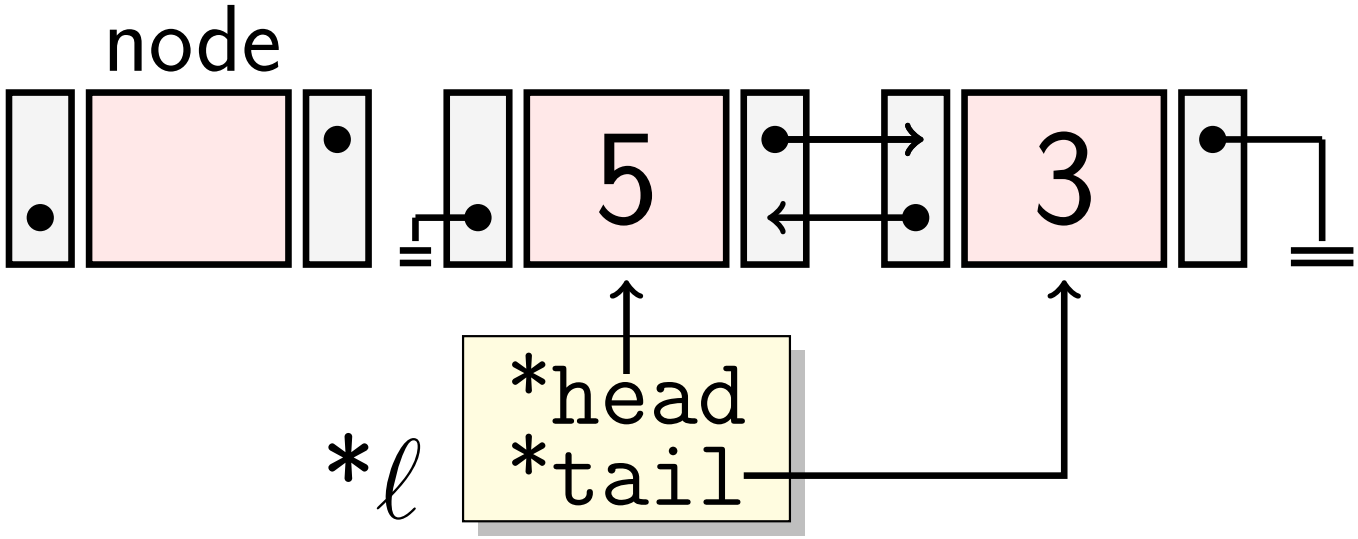


1: **void insert-front** (List *ℓ, int elem)

2: Node* node = (Node*)malloc(sizeof(Node));
3: node→data = elem;
4: node→prev = NULL;
5: ...

Inserir: 7

Inserir (início)

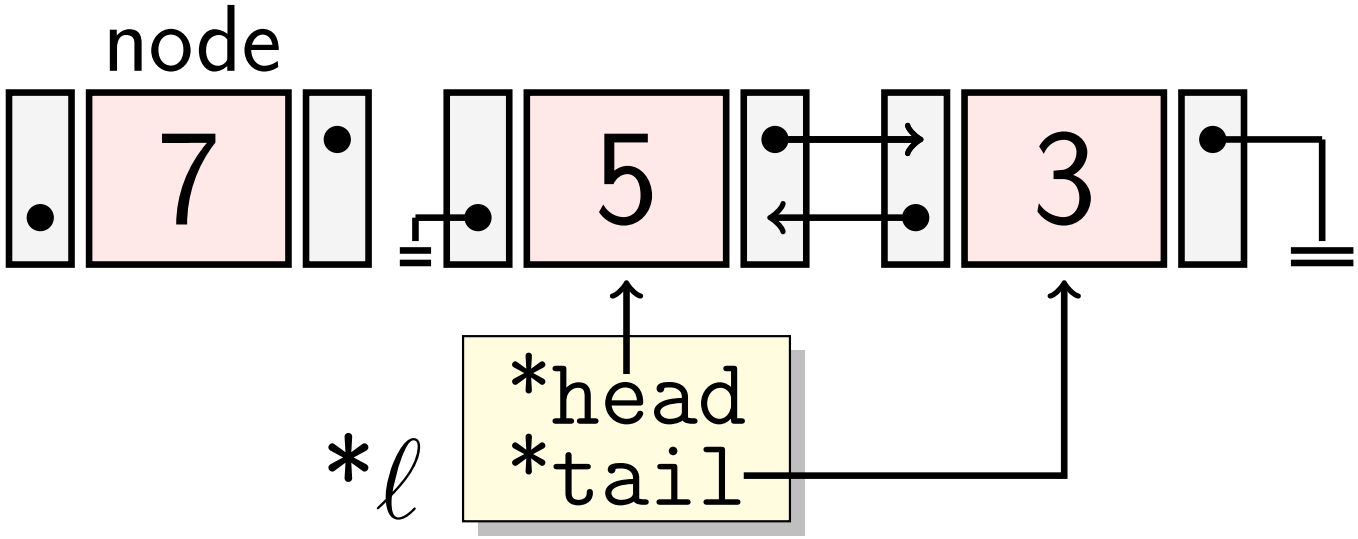


1: **void insert-front** (List *ℓ, int elem)

2: Node* node = (Node*)malloc(sizeof(Node));
3: node→data = elem;
4: node→prev = NULL;
5: ...

Inserir: 7

Inserir (início)

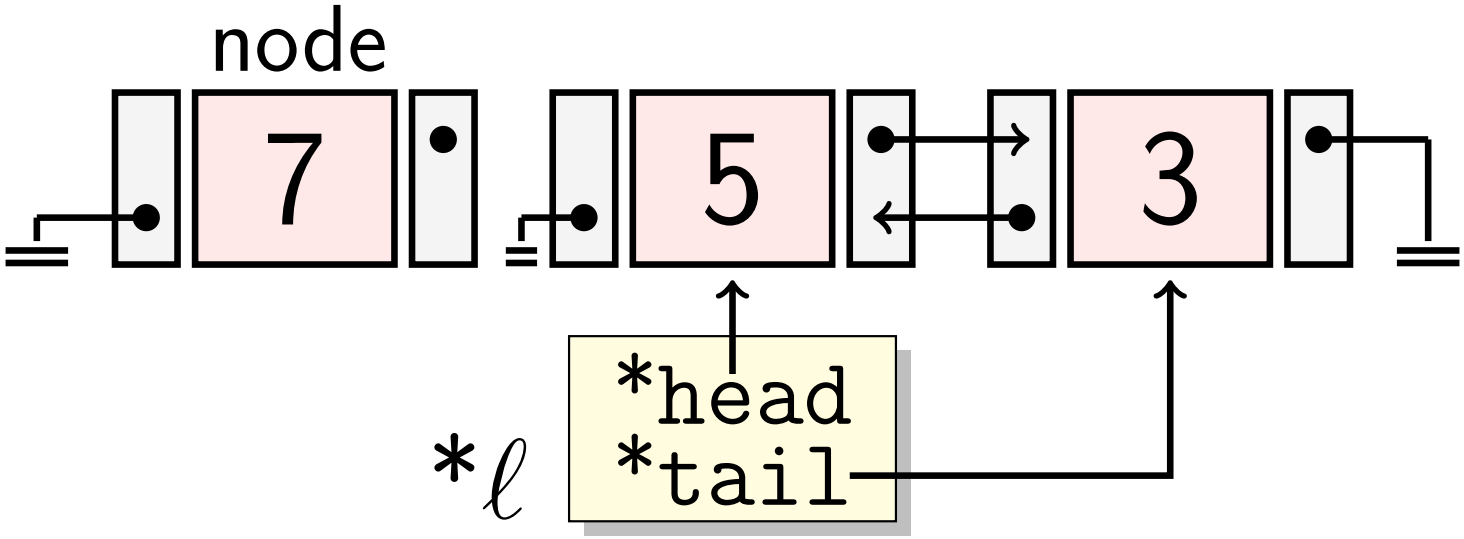


1: **void insert-front** (List *ℓ, int elem)

2: Node* node = (Node*)malloc(sizeof(Node));
3: node→data = elem;
4: node→prev = NULL;
5: ...

Inserir: 7

Inserir (início)

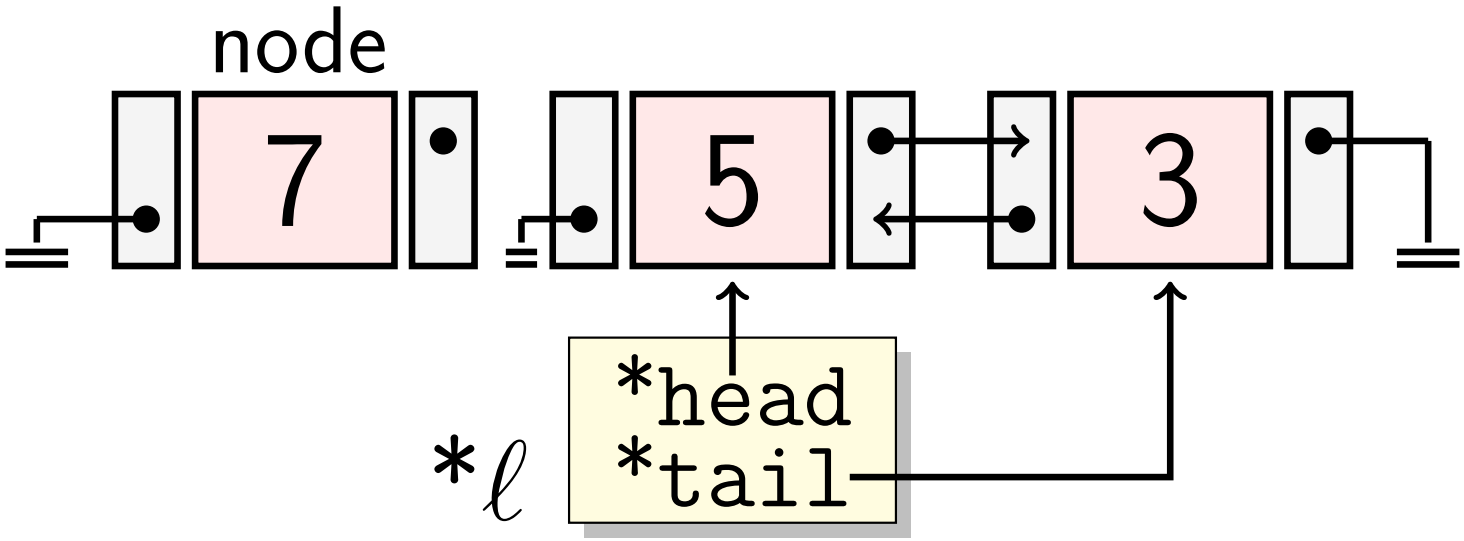


1: **void insert-front** (List *ℓ, int elem)

2: Node* node = (Node*)malloc(sizeof(Node));
3: node→data = elem;
4: node→prev = NULL;
5: ...

Inserir: 7

Inserir (início)

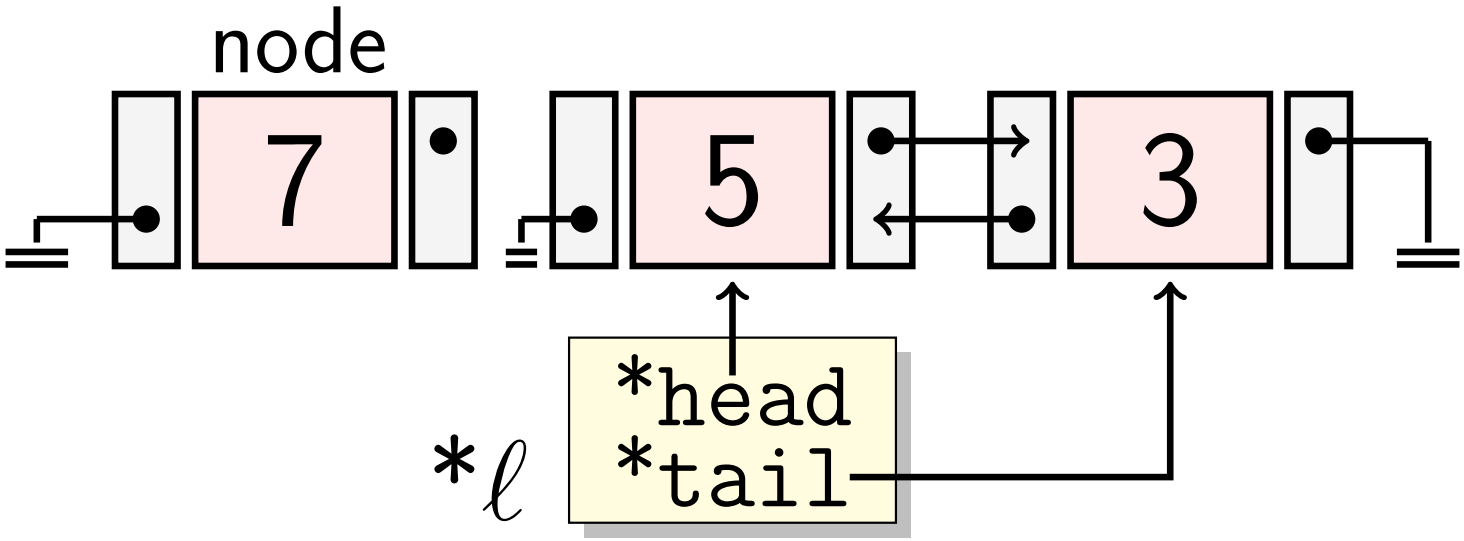


1: **void insert-front** (List *ℓ, int elem)

2: **if** $\ell \rightarrow$head $\neq$ NULL;
3: node$\rightarrow$next = $\ell \rightarrow$head;
4: $\ell \rightarrow$head$\rightarrow$prev = node;
5: **else**
6: node$\rightarrow$next = NULL;
7: $\ell \rightarrow$tail = node;
8: $\ell \rightarrow$head = node;

Inserir: 7

Inserir (início)

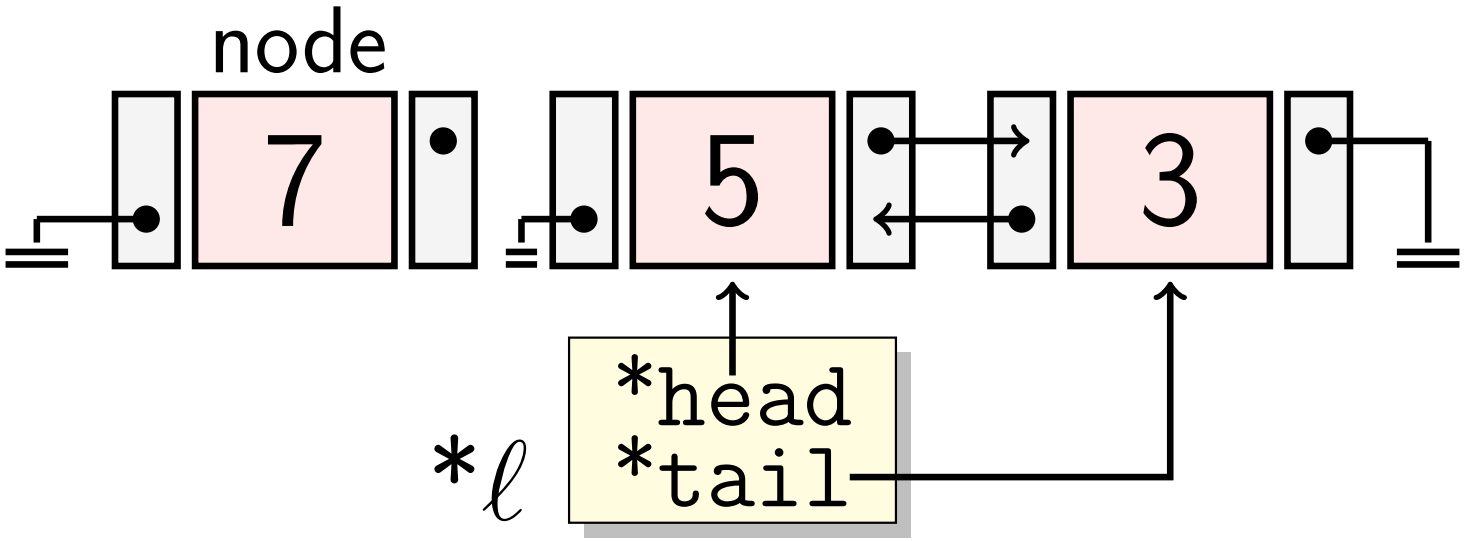


1: **void insert-front** (List *ℓ, int elem)

2: **if** $\ell \rightarrow$head $\neq$ NULL;
3: node$\rightarrow$next = $\ell \rightarrow$head;
4: $\ell \rightarrow$head$\rightarrow$prev = node;
5: **else**
6: node$\rightarrow$next = NULL;
7: $\ell \rightarrow$tail = node;
8: $\ell \rightarrow$head = node;

Inserir: 7

Inserir (início)

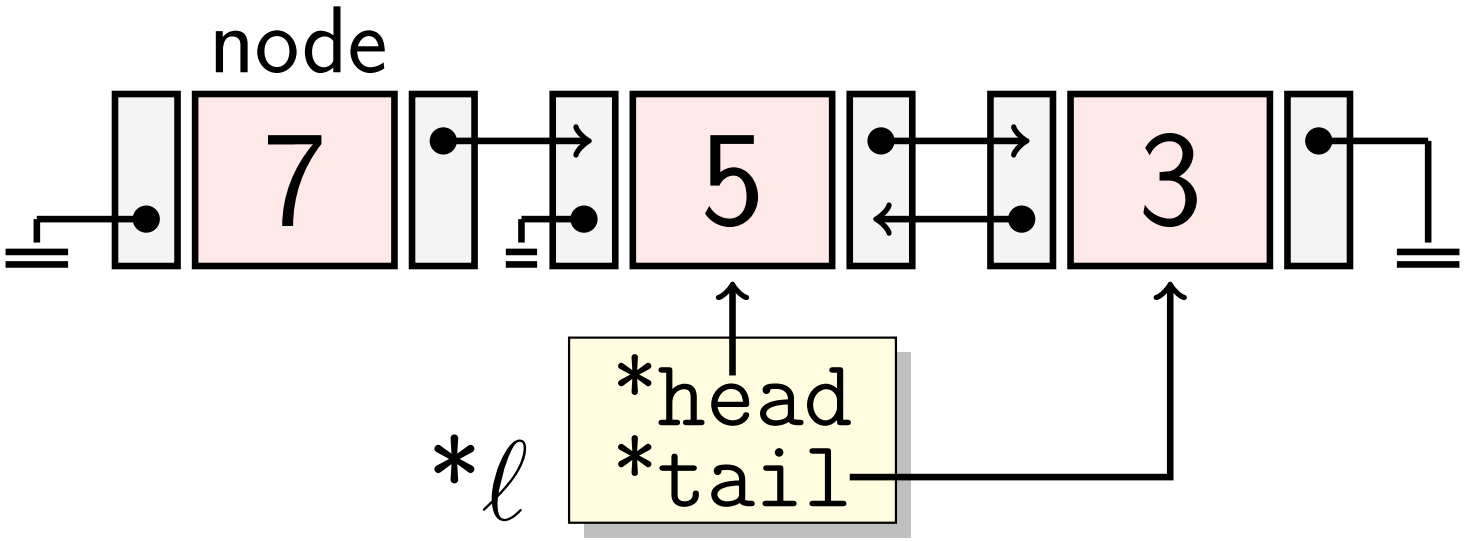


1: **void insert-front** (List *ℓ, int elem)

2: **if** $\ell \rightarrow$head $\neq$ NULL;
3: node$\rightarrow$next $= \ell \rightarrow$head;
4: $\ell \rightarrow$head$\rightarrow$prev $=$ node;
5: **else**
6: node$\rightarrow$next $=$ NULL;
7: $\ell \rightarrow$tail $=$ node;
8: $\ell \rightarrow$head $=$ node;

Inserir: 7

Inserir (início)

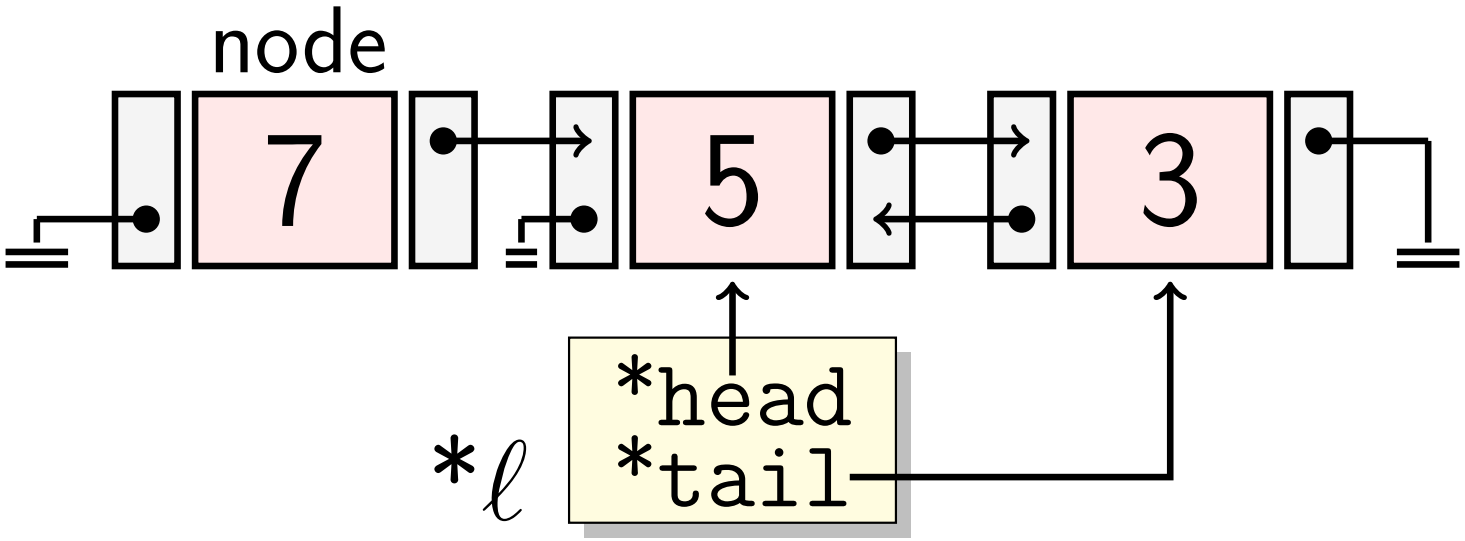


1: **void insert-front** (List *ℓ, int elem)

2: **if** $\ell \rightarrow$head $\neq$ NULL;
3: node$\rightarrow$next $= \ell \rightarrow$head;
4: $\ell \rightarrow$head$\rightarrow$prev $=$ node;
5: **else**
6: node$\rightarrow$next $=$ NULL;
7: $\ell \rightarrow$tail $=$ node;
8: $\ell \rightarrow$head $=$ node;

Inserir: 7

Inserir (início)

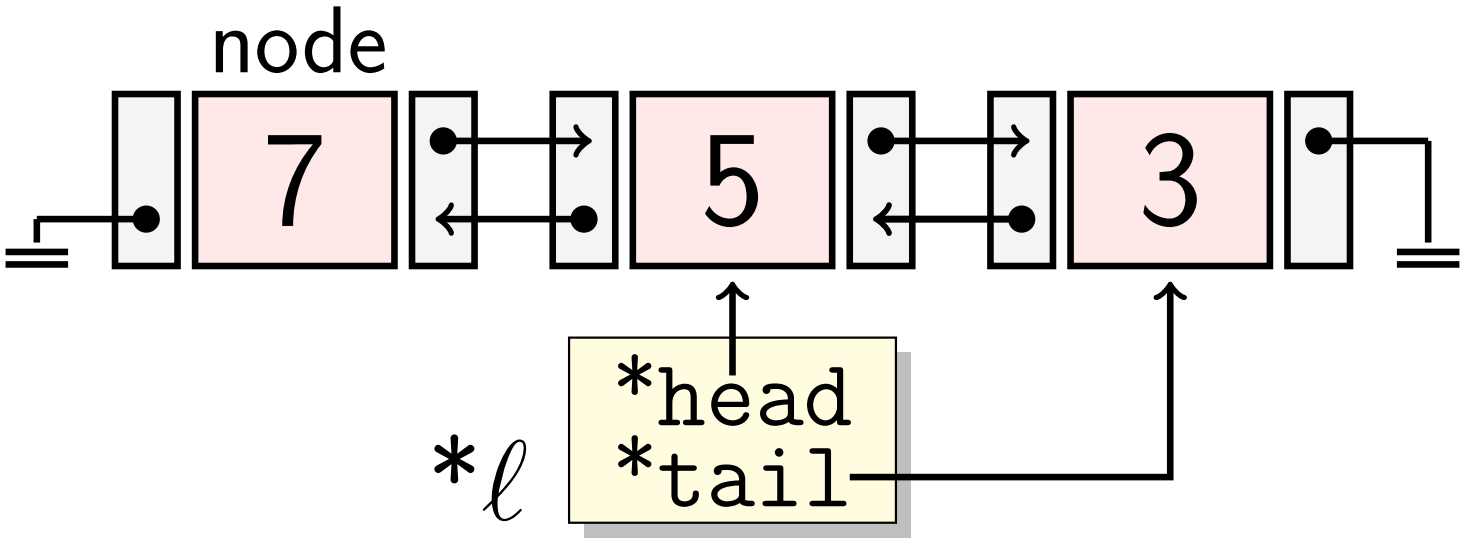


1: **void insert-front** (List *ℓ, int elem)

2: **if** $\ell \rightarrow$head $\neq$ NULL;
3: node$\rightarrow$next = $\ell \rightarrow$head;
4: $\ell \rightarrow$head$\rightarrow$prev = node;
5: **else**
6: node$\rightarrow$next = NULL;
7: $\ell \rightarrow$tail = node;
8: $\ell \rightarrow$head = node;

Inserir: 7

Inserir (início)

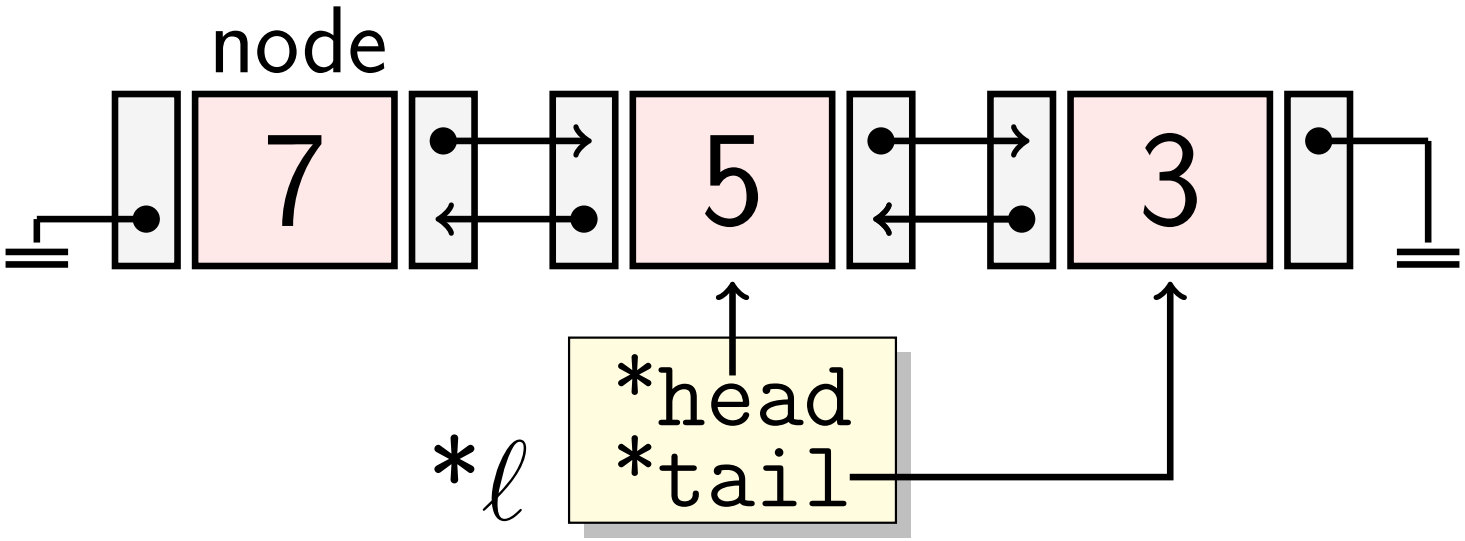


1: **void insert-front** (List *ℓ, int elem)

2: **if** $\ell \rightarrow$head $\neq$ NULL;
3: node$\rightarrow$next = $\ell \rightarrow$head;
4: $\ell \rightarrow$head$\rightarrow$prev = node;
5: **else**
6: node$\rightarrow$next = NULL;
7: $\ell \rightarrow$tail = node;
8: $\ell \rightarrow$head = node;

Inserir: 7

Inserir (início)

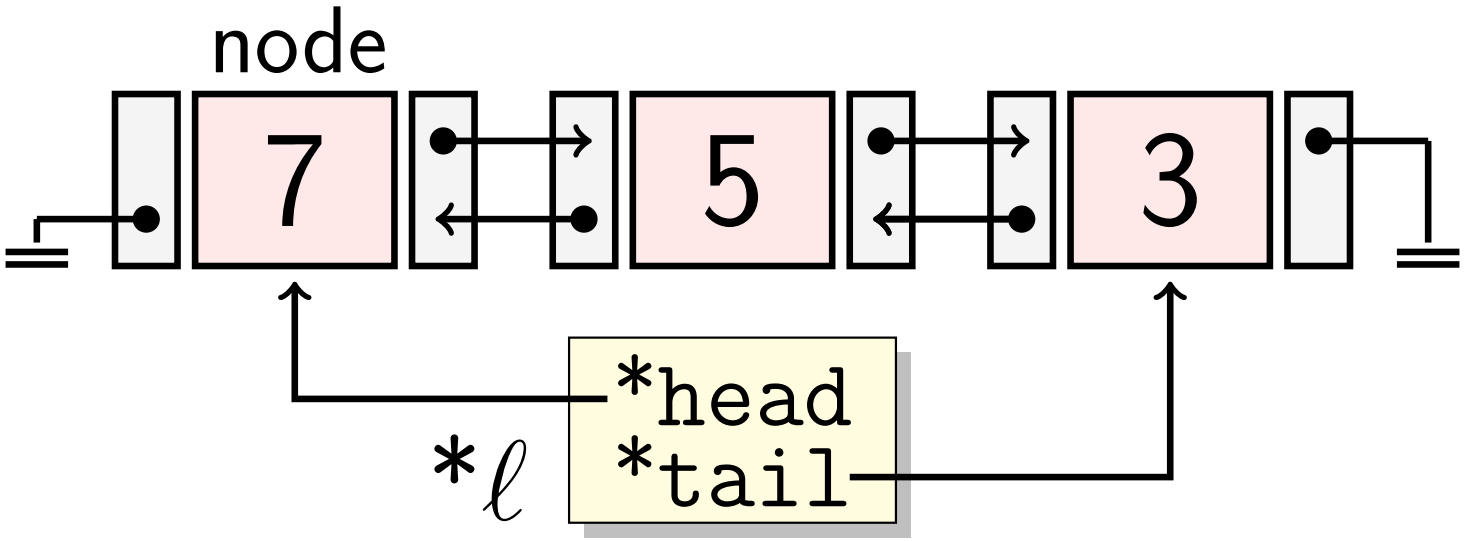


1: **void insert-front** (List *ℓ, int elem)

2: **if** $\ell \rightarrow$head $\neq$ NULL;
3: node$\rightarrow$next = $\ell \rightarrow$head;
4: $\ell \rightarrow$head$\rightarrow$prev = node;
5: **else**
6: node$\rightarrow$next = NULL;
7: $\ell \rightarrow$tail = node;
8: $\ell \rightarrow$head = node;

Inserir: 7

Inserir (início)



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

2: **if** $\ell \rightarrow$head $\neq$ NULL;
3: node$\rightarrow$next = $\ell \rightarrow$head;
4: $\ell \rightarrow$head$\rightarrow$prev = node;
5: **else**
6: node$\rightarrow$next = NULL;
7: $\ell \rightarrow$tail = node;
8: $\ell \rightarrow$head = node;

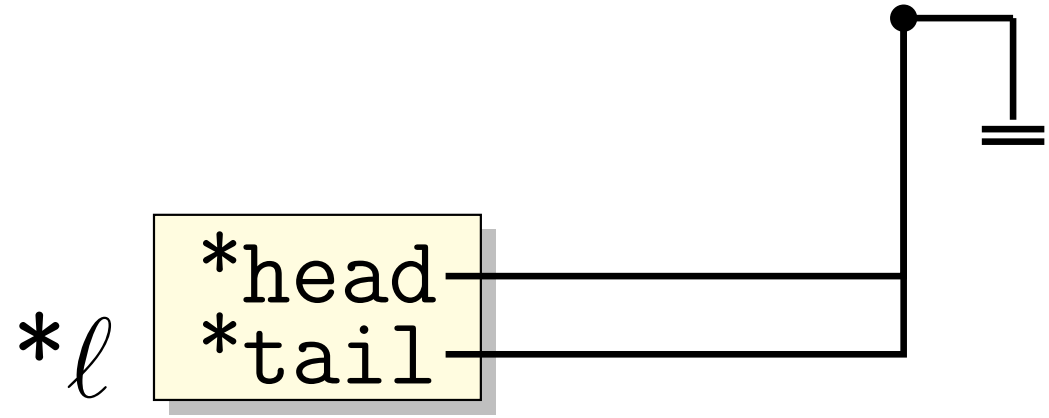
Inserir: 7

Sumário

Inserir-front (funcionamento)

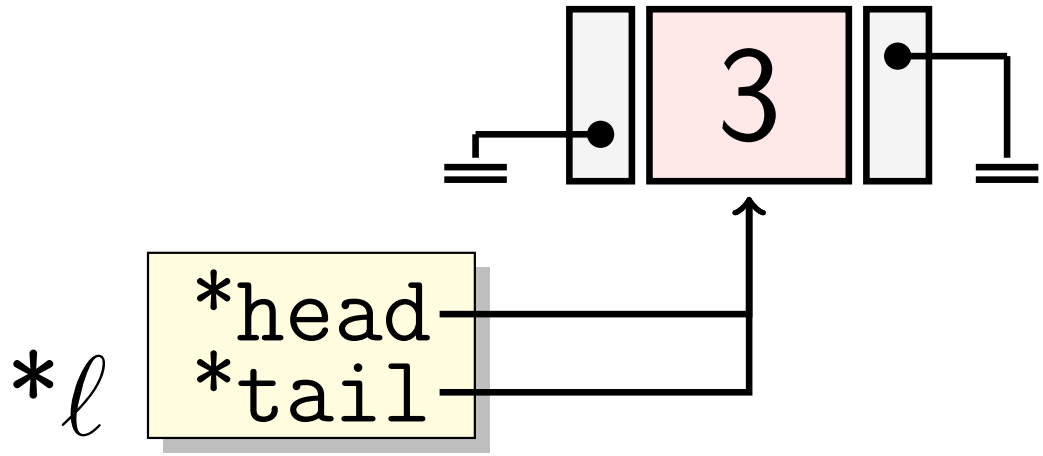
```
1: int main (void)
---
2:     List *ℓ = create ();
3:     insert-front (ℓ, 3);
4:     insert-front (ℓ, 5);
5:     insert-front (ℓ, 7);
6: return 0
```

Inserir-front (funcionamento)



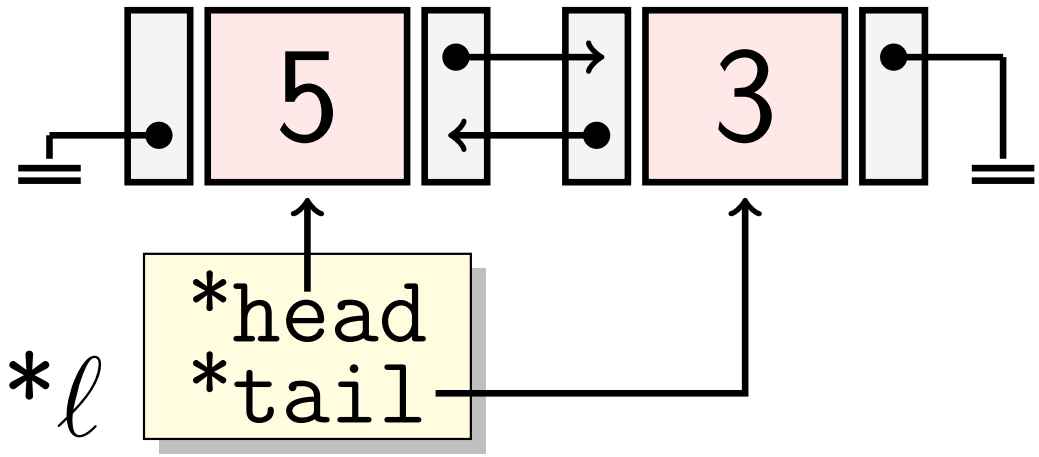

```
1: int main (void)
---
2:     List *ℓ = create ();
3:     insert-front (ℓ, 3);
4:     insert-front (ℓ, 5);
5:     insert-front (ℓ, 7);
6: return 0
```

Inserir-front (funcionamento)



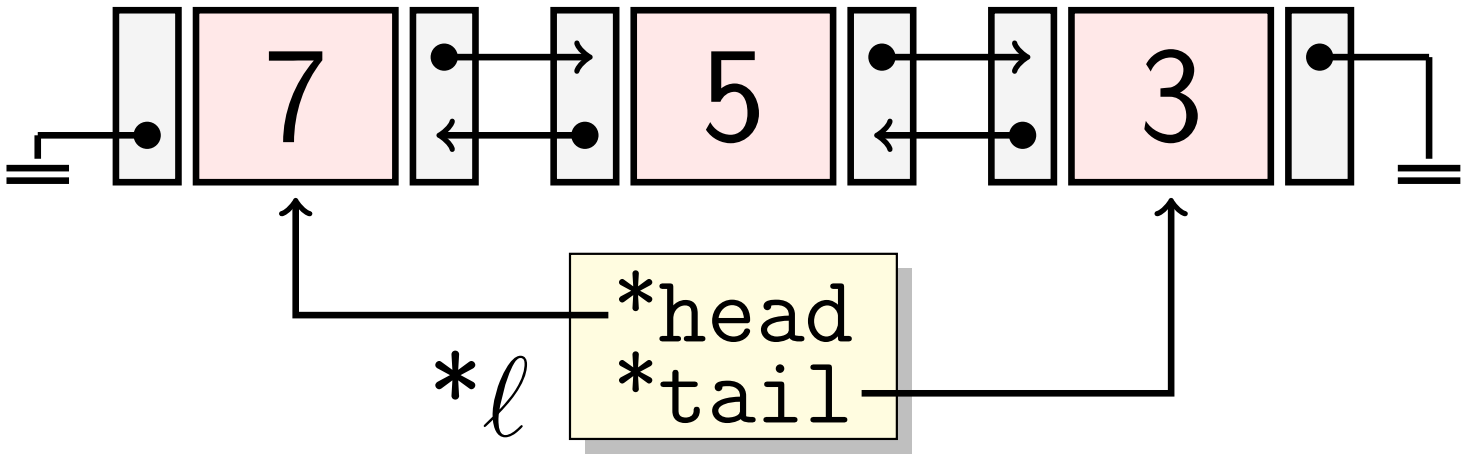

```
1: int main (void)
---
2:     List *ℓ = create ();
3:     insert-front (ℓ, 3);
4:     insert-front (ℓ, 5);
5:     insert-front (ℓ, 7);
6: return 0
```

Inserir-front (funcionamento)




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

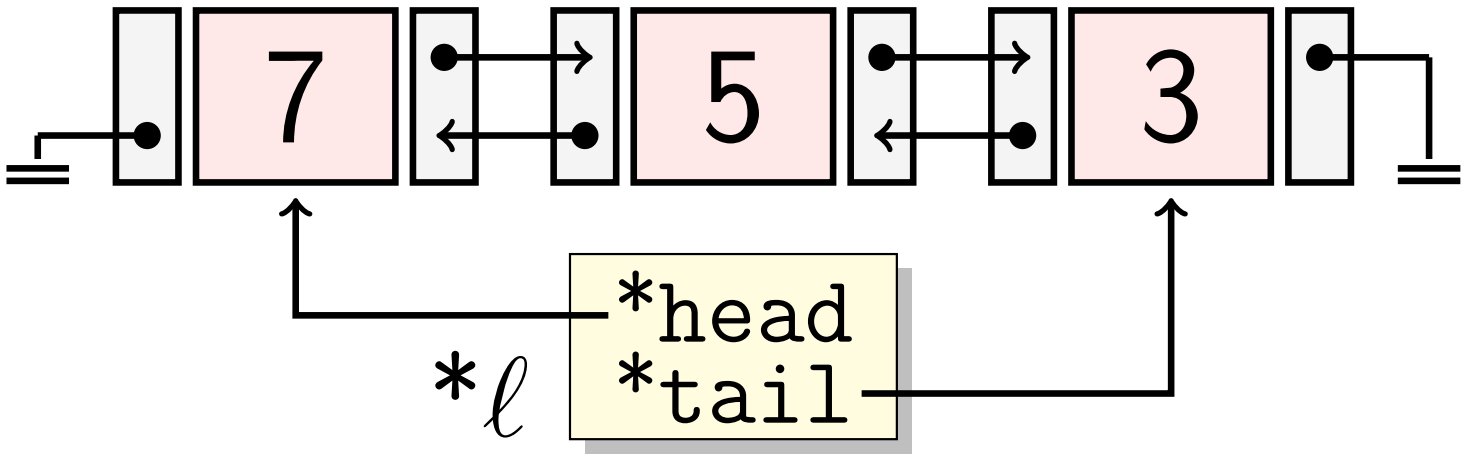
Inserir-front (funcionamento)




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

Sumário

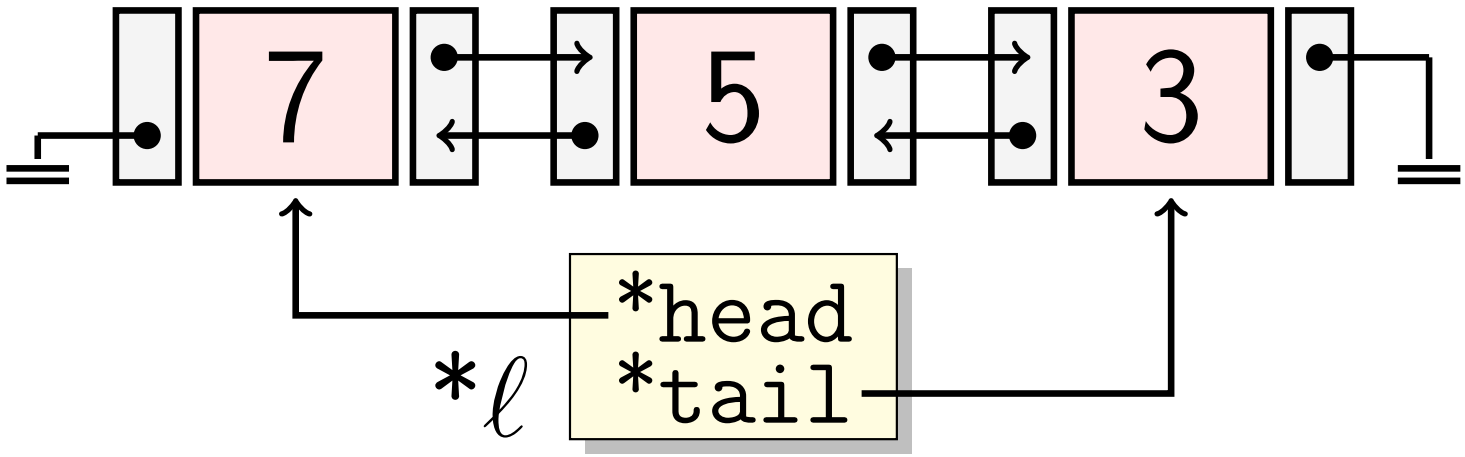
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 7

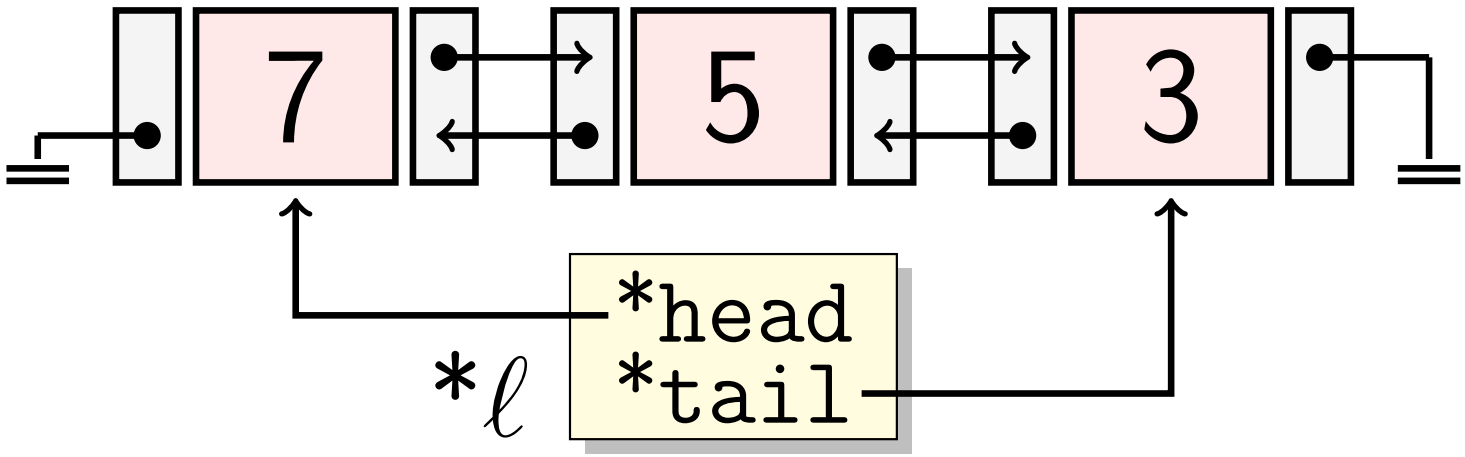
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 7

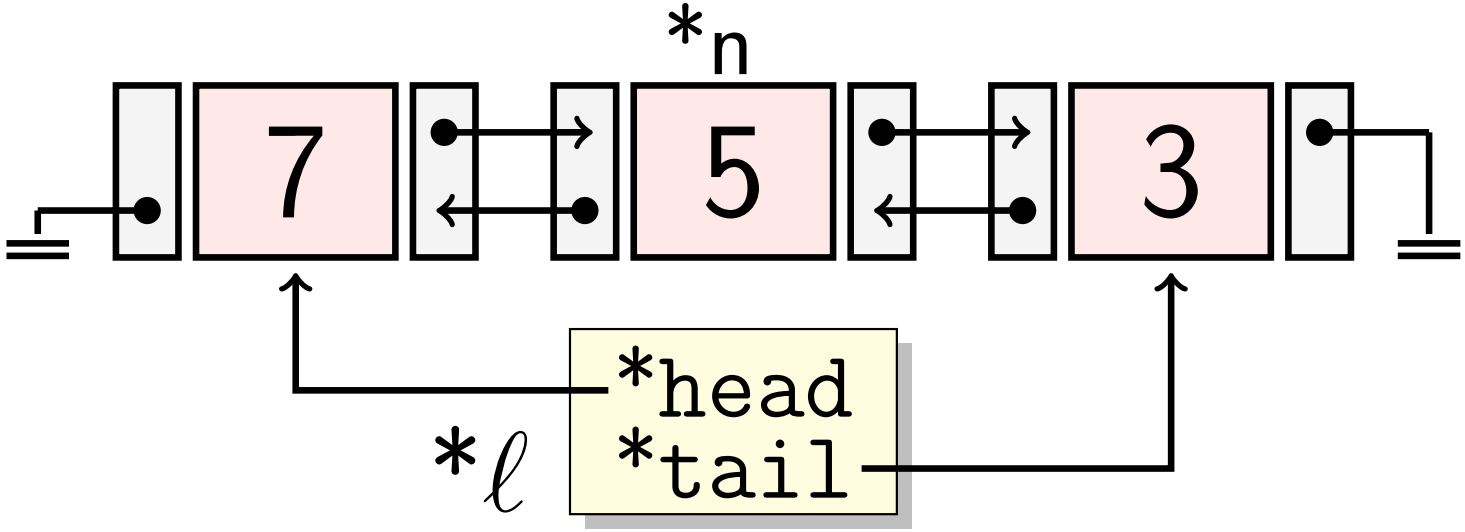
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 7

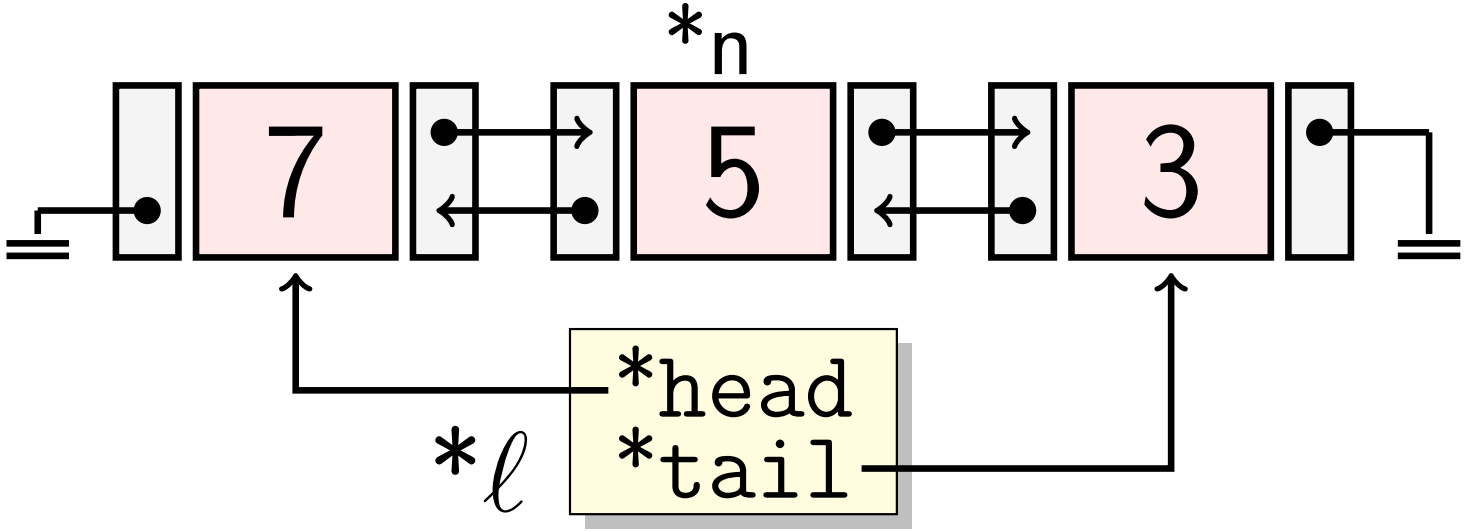
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 7

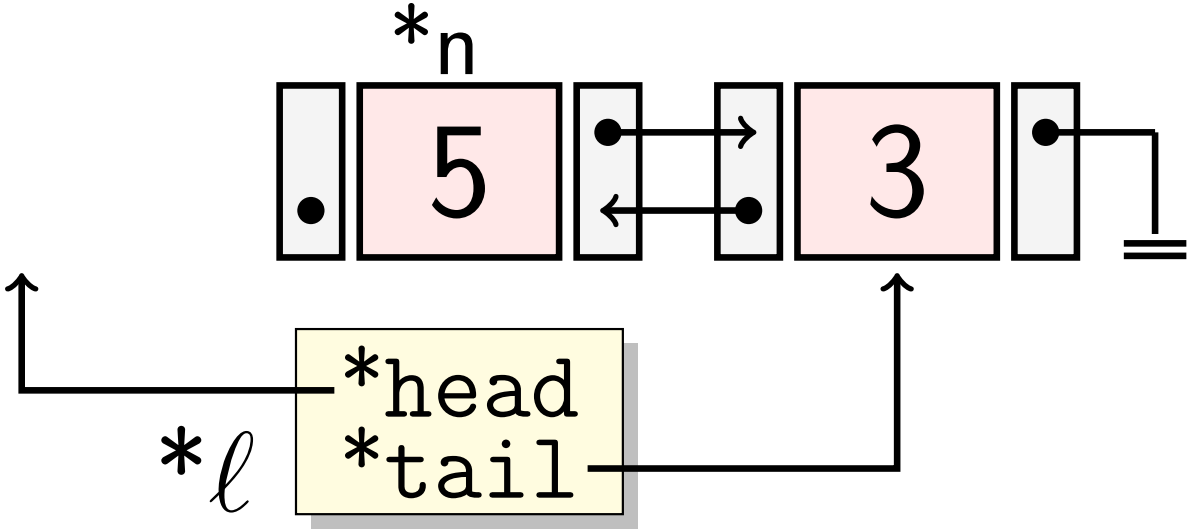
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 7

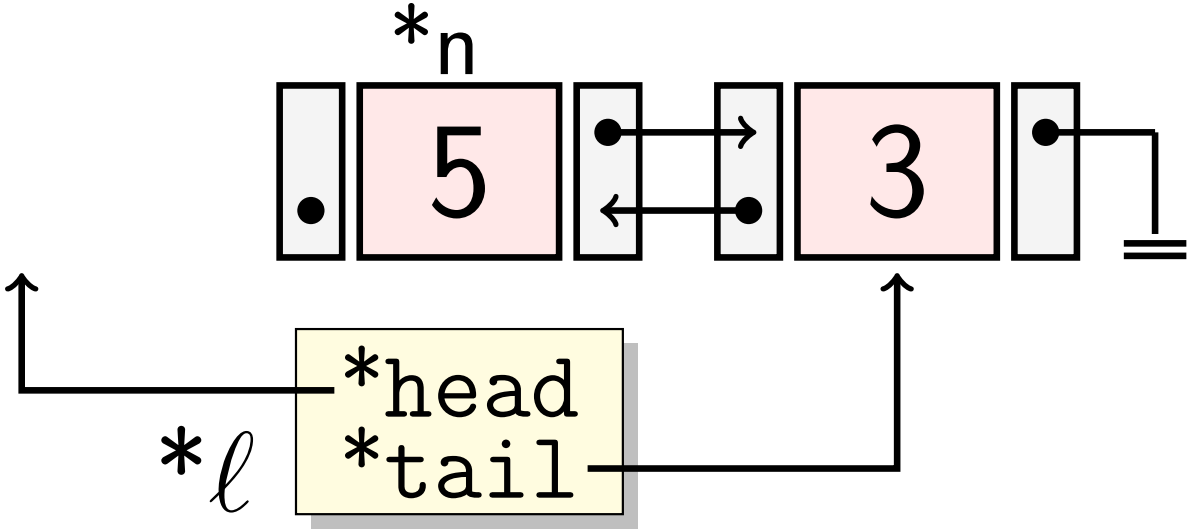
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 7

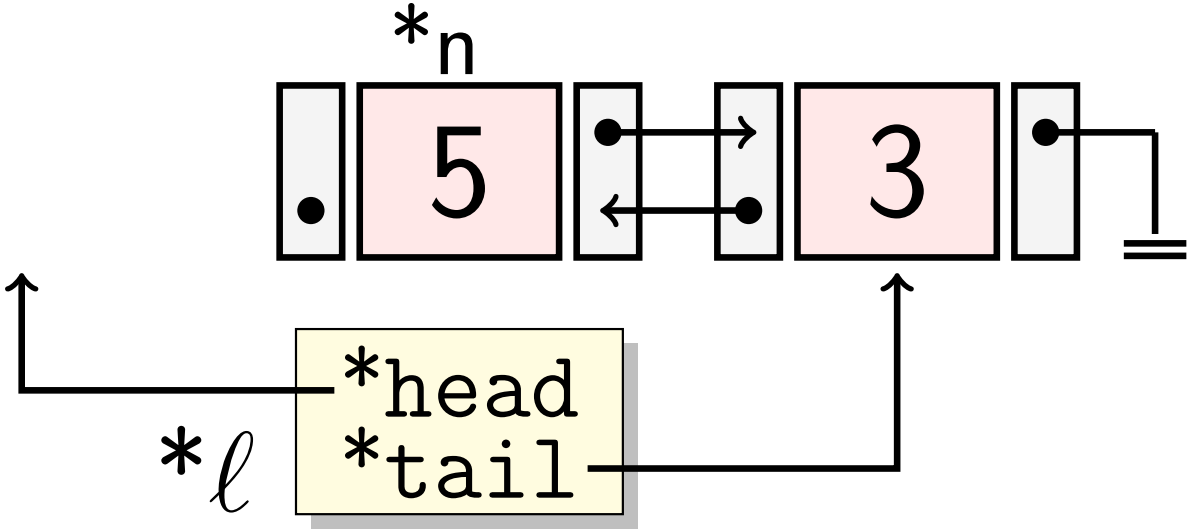
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 7

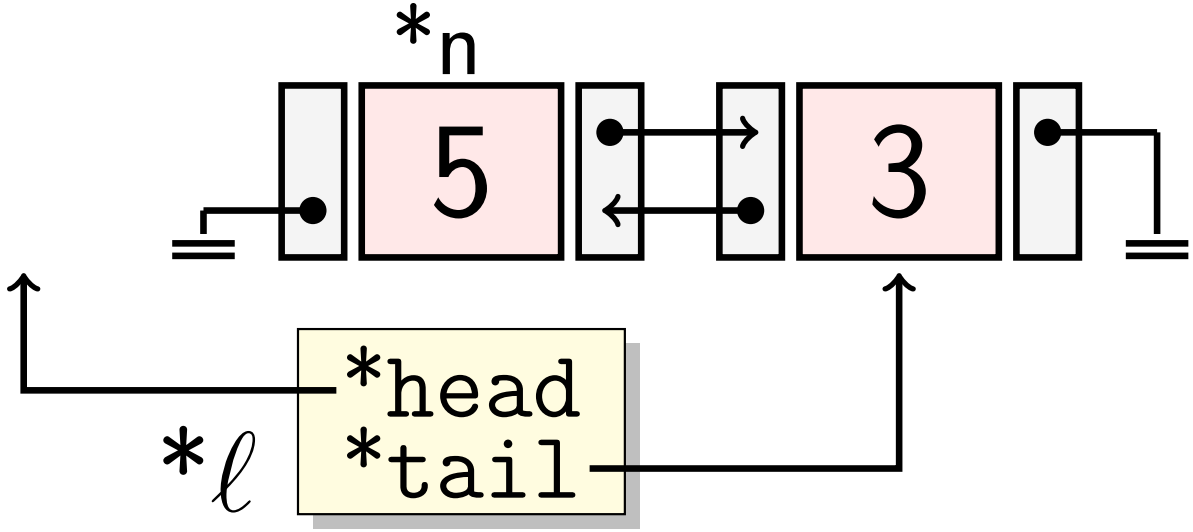
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 7

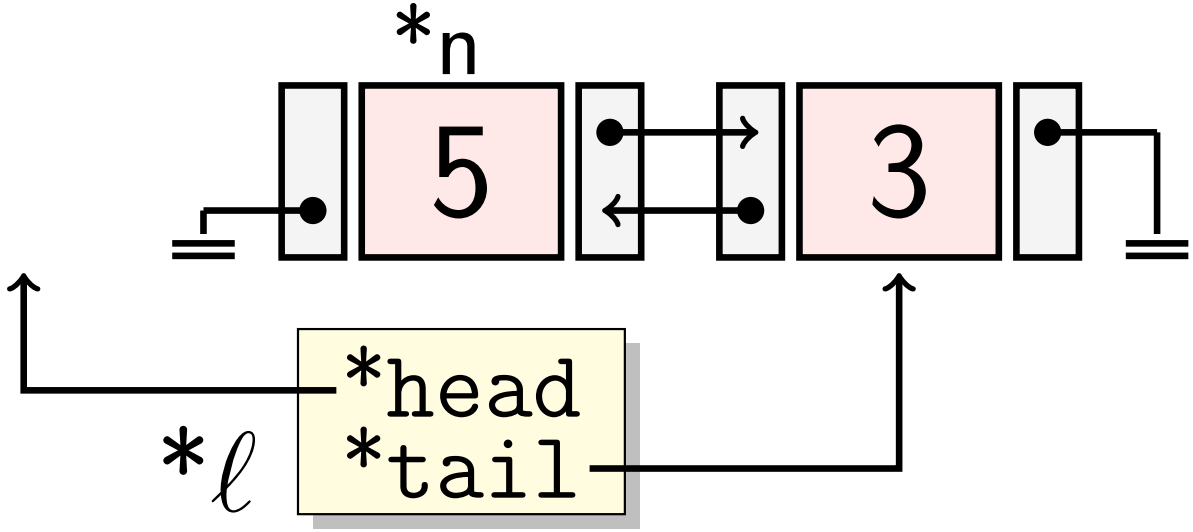
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 7

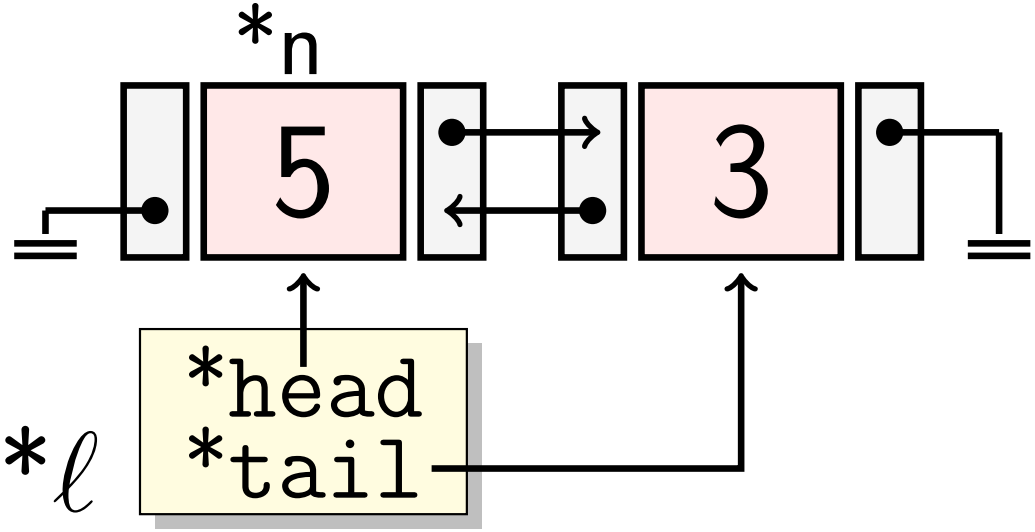
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 7

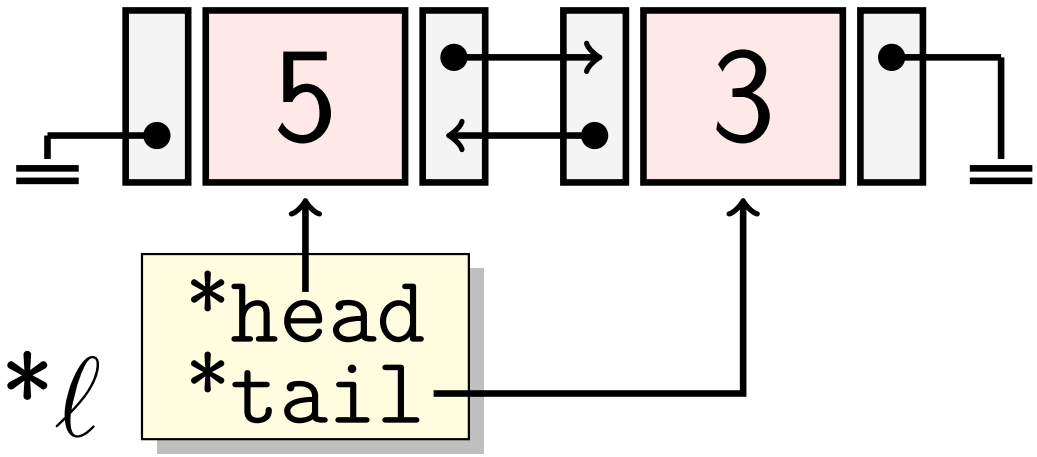
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 7

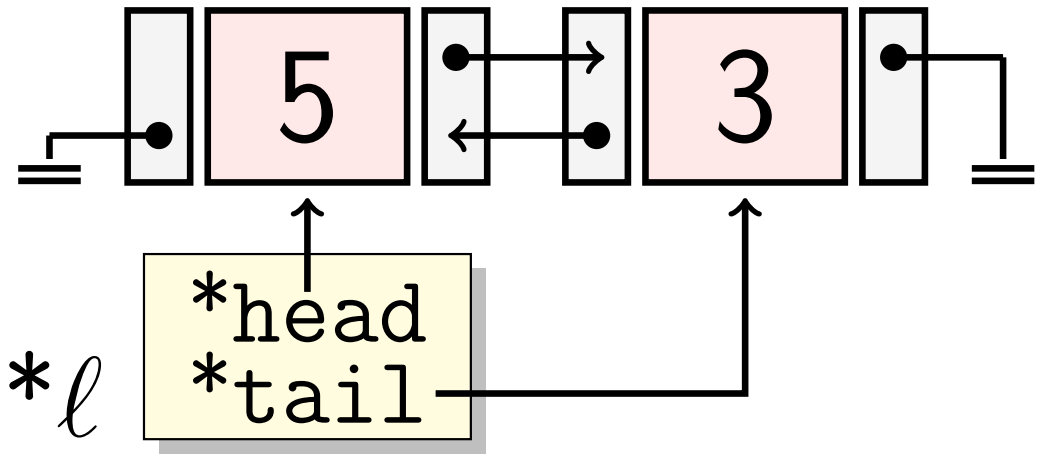
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 5

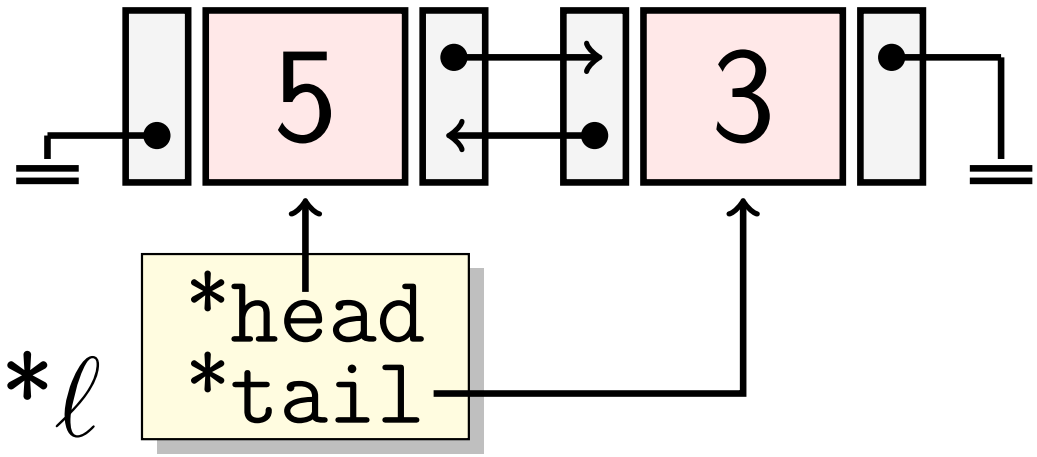
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 5

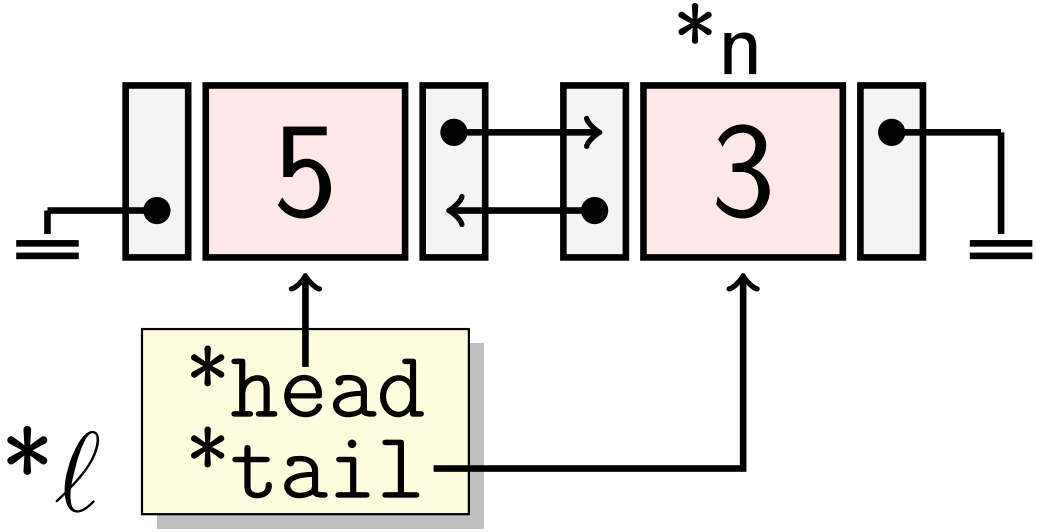
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 5

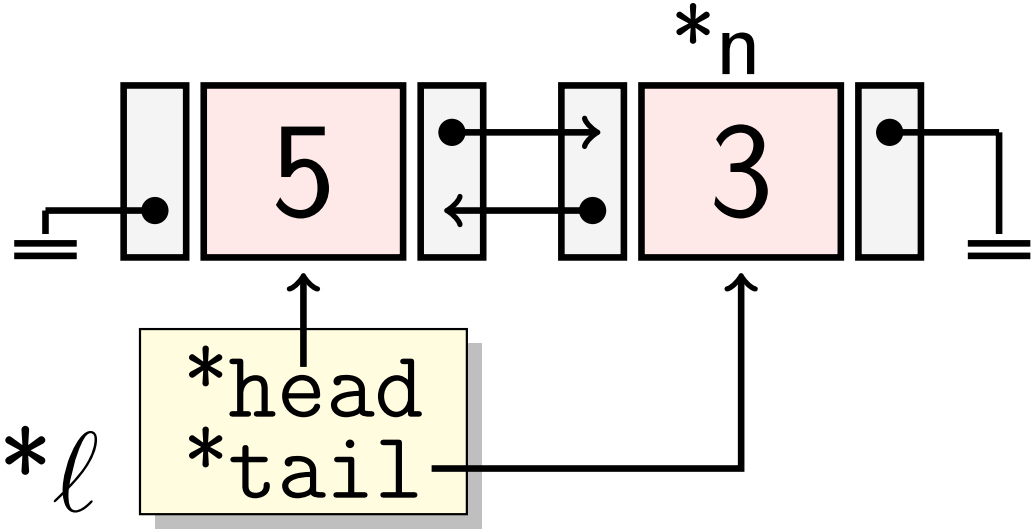
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 5

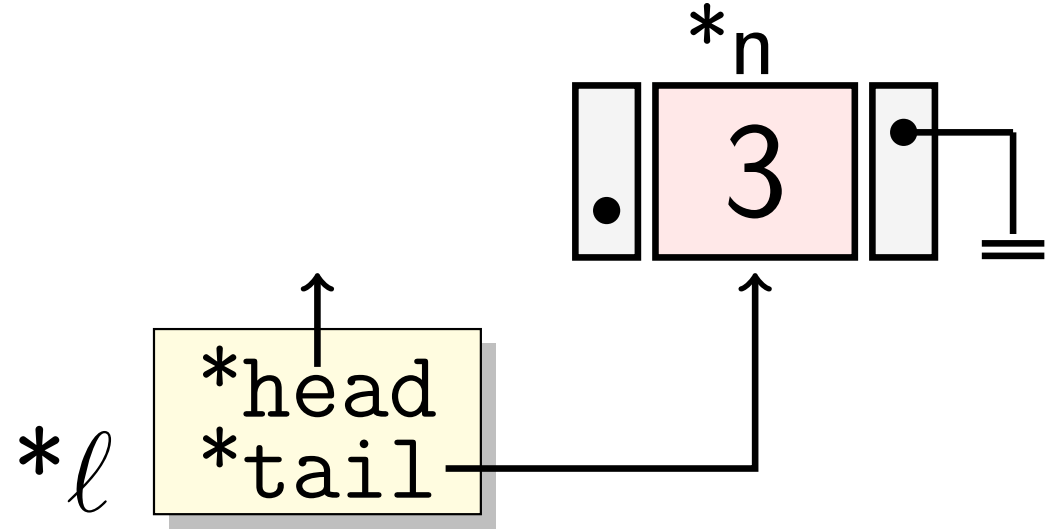
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 5

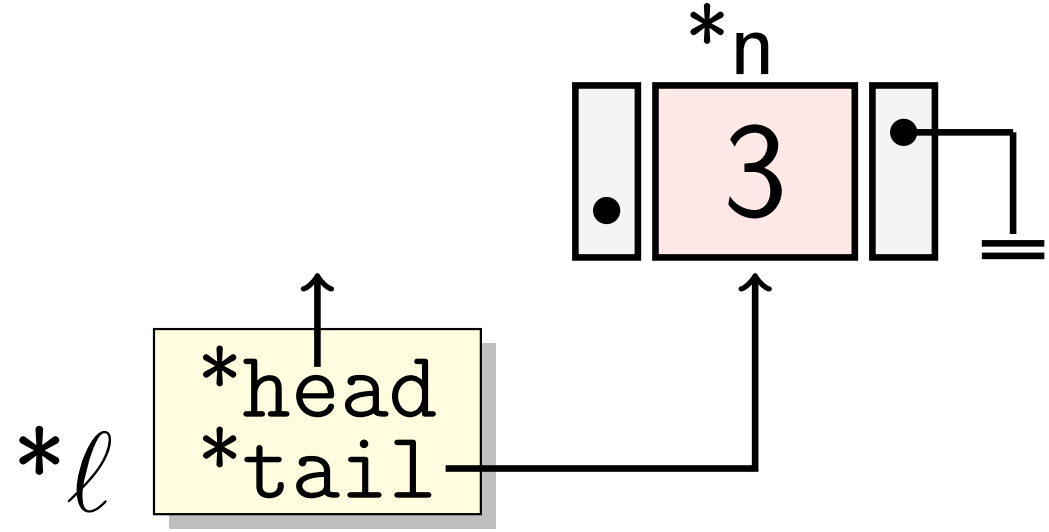
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 5

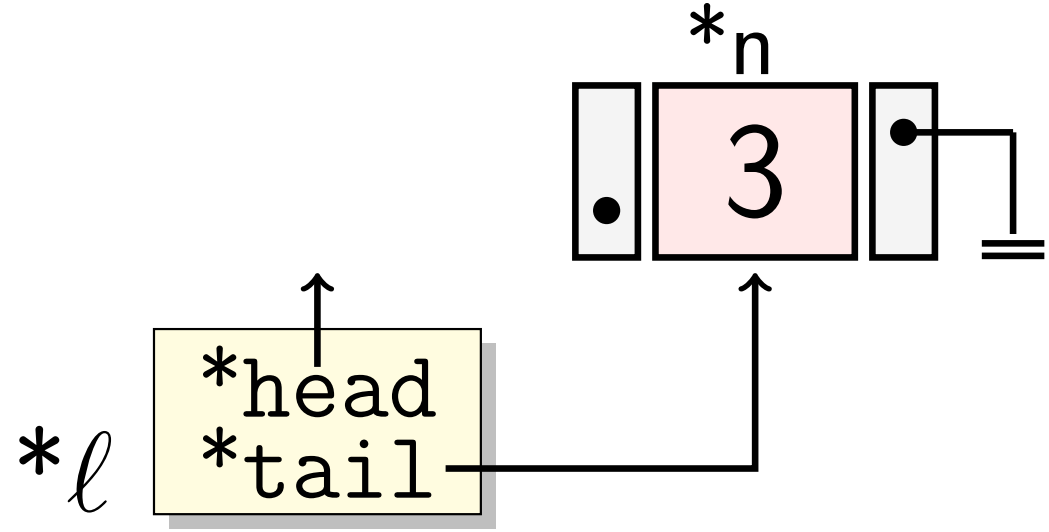
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 5

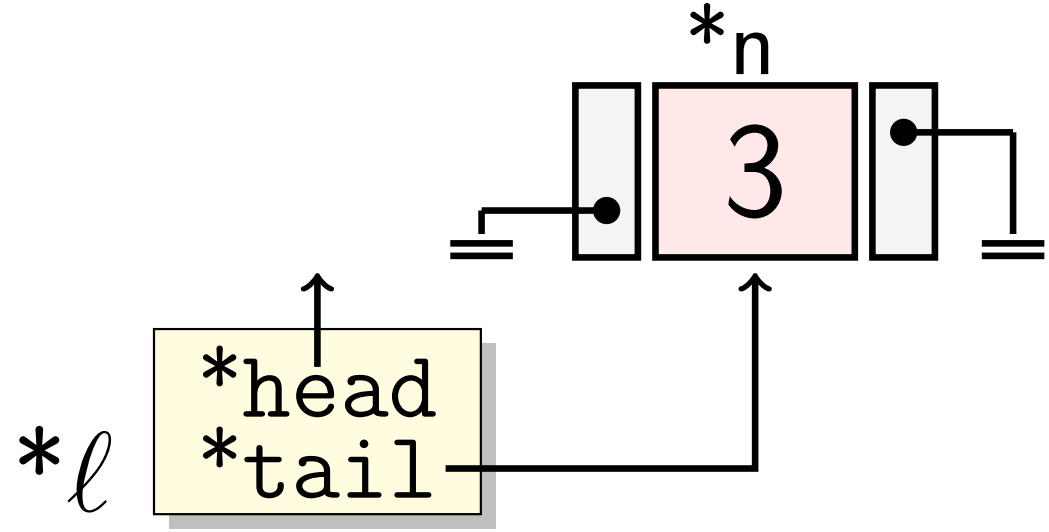
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 5

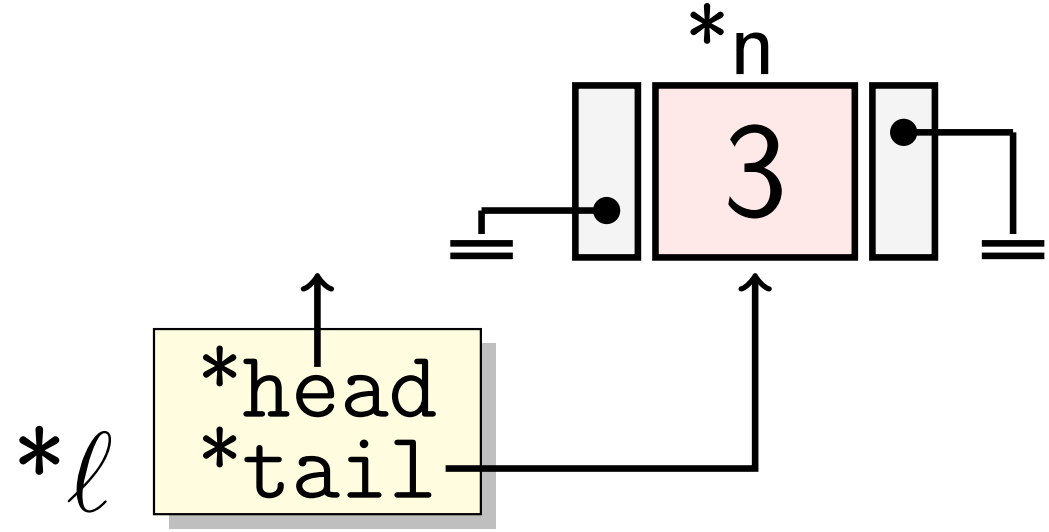
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 5

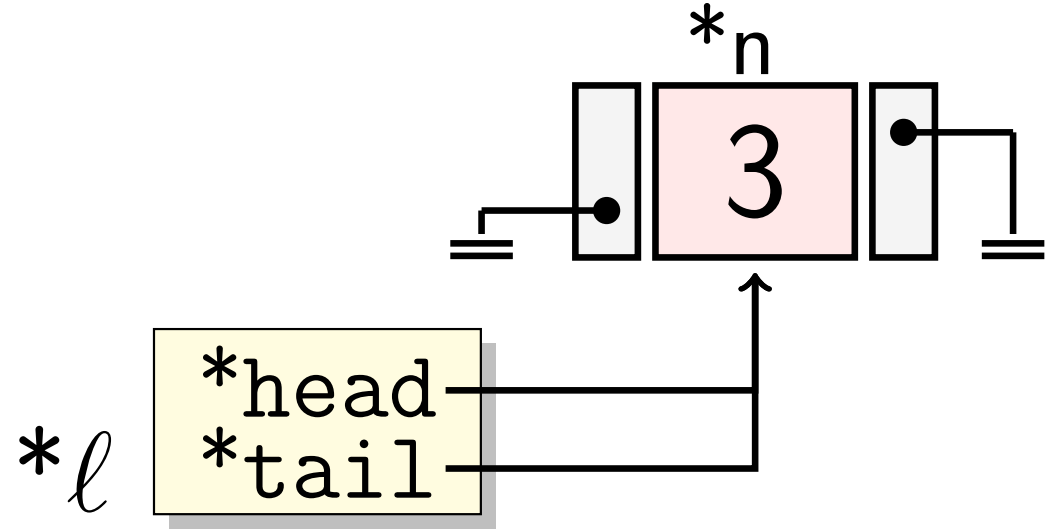
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 5

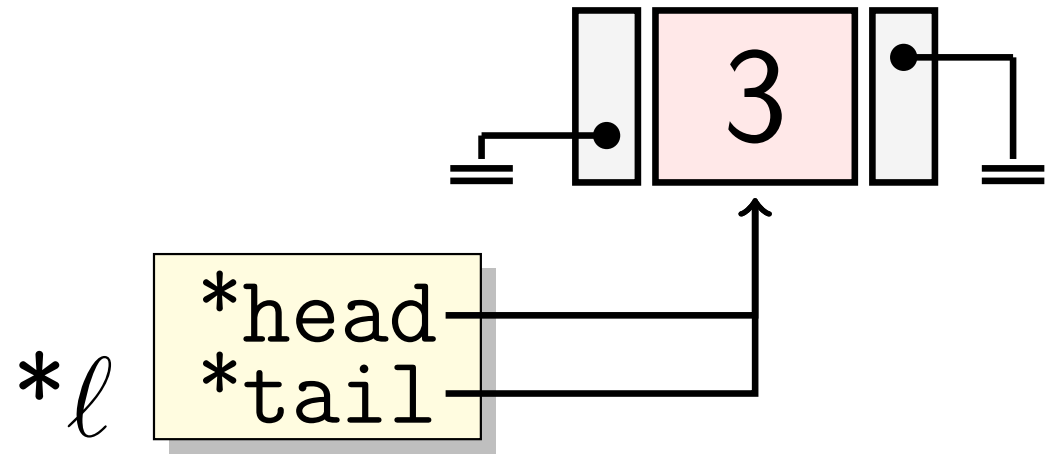
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 5

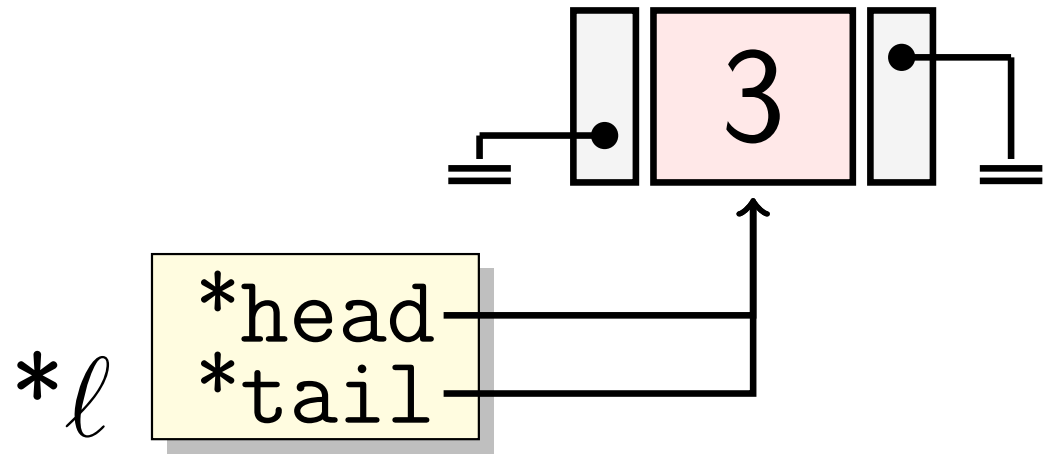
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 3

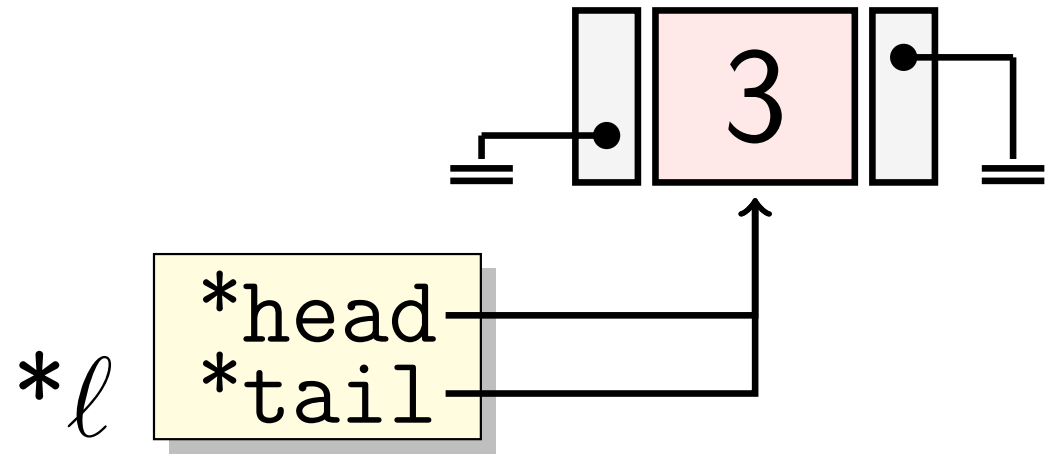
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 3

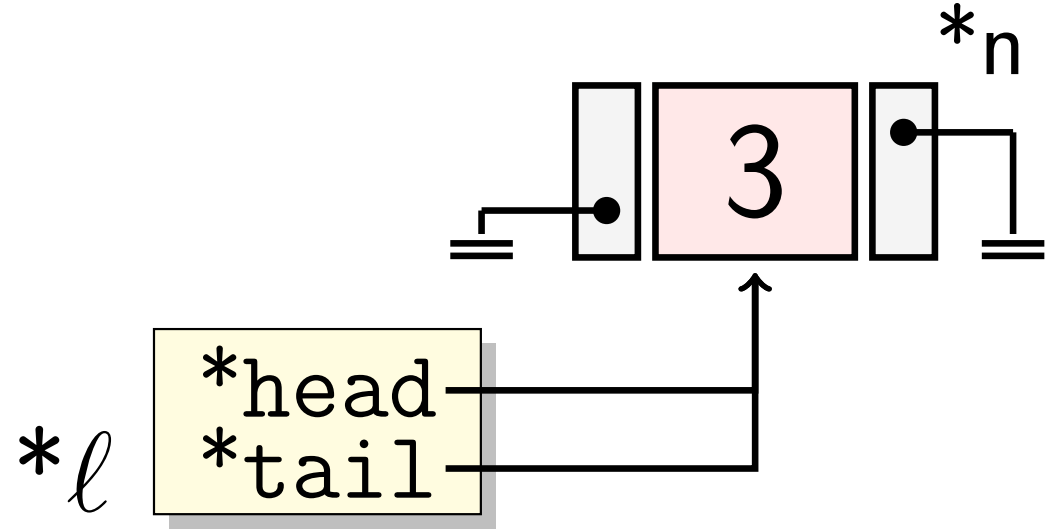
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 3

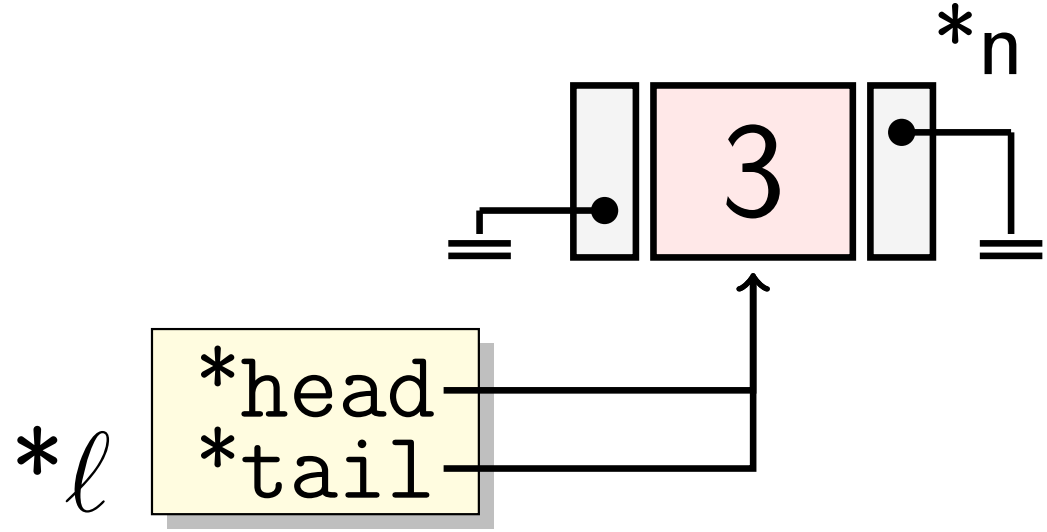
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 3

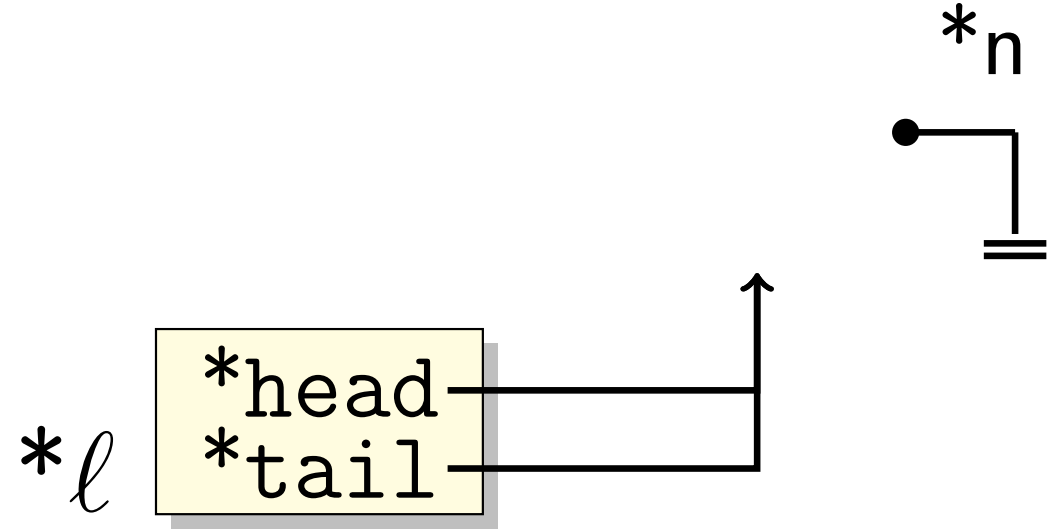
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 3

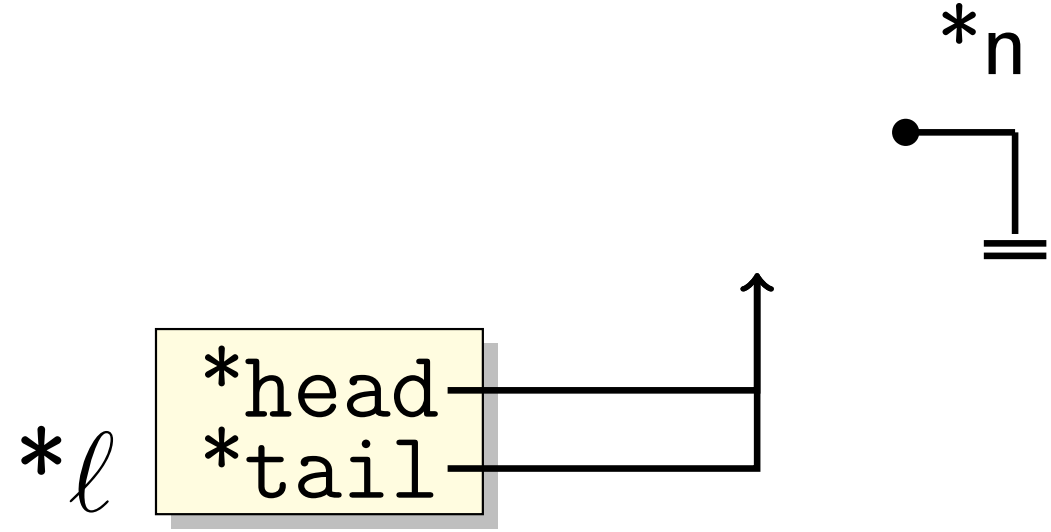
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 3

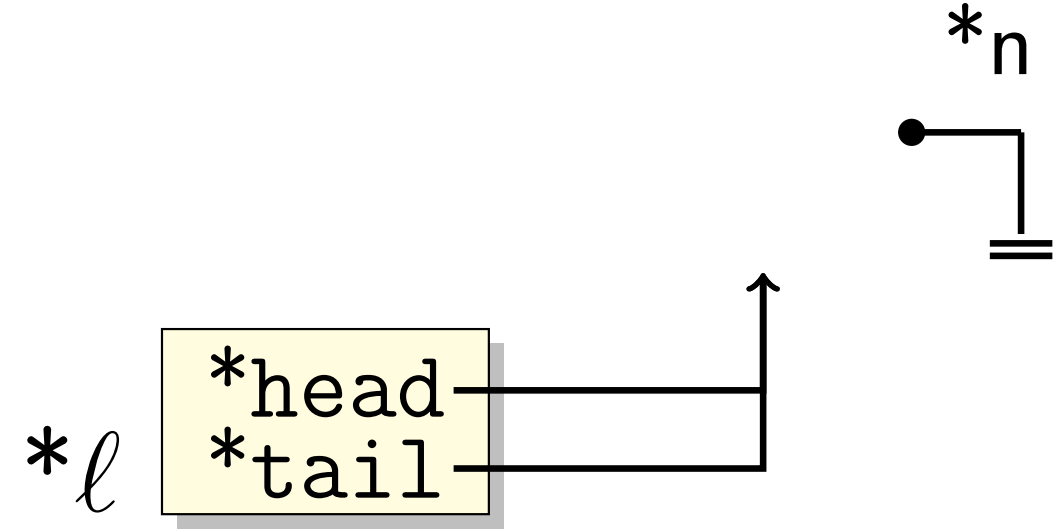
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 3

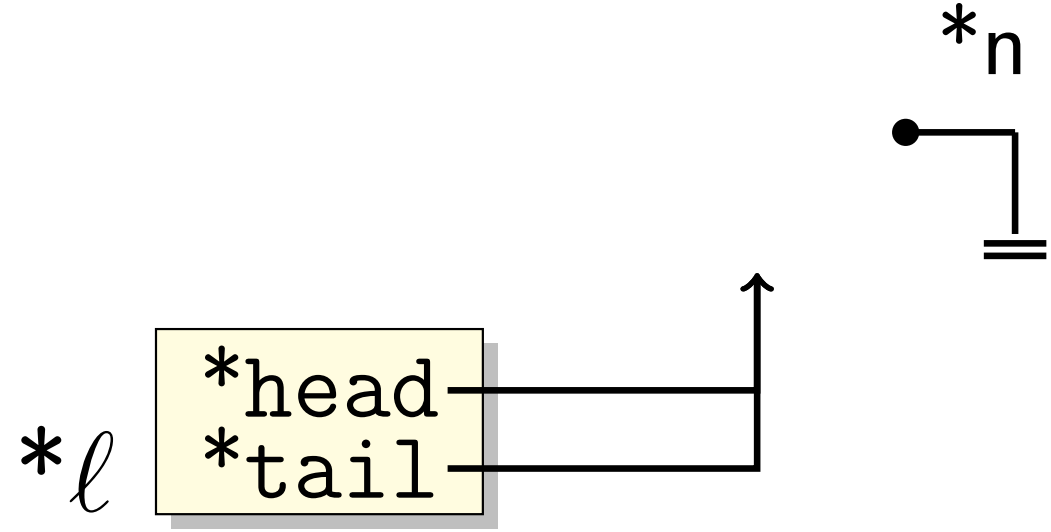
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 3

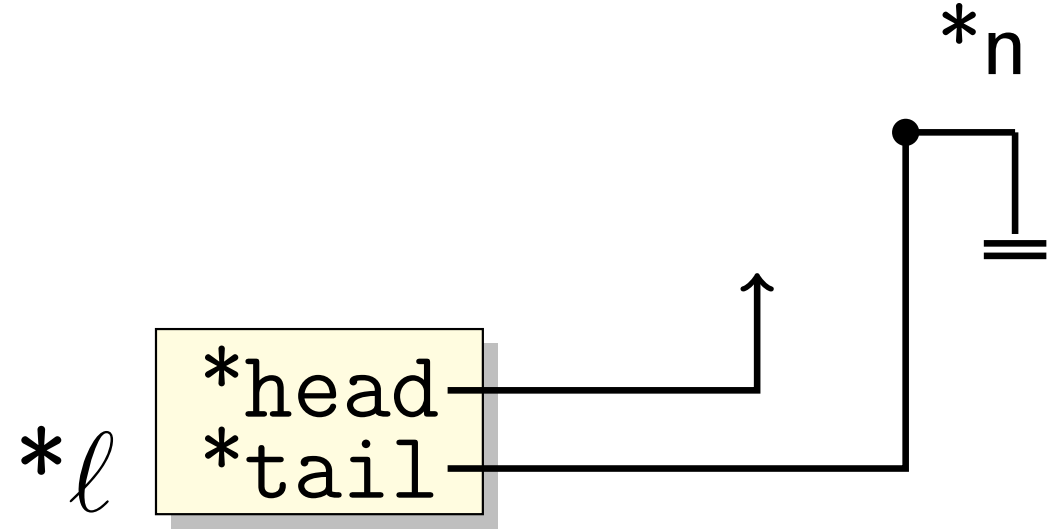
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 3

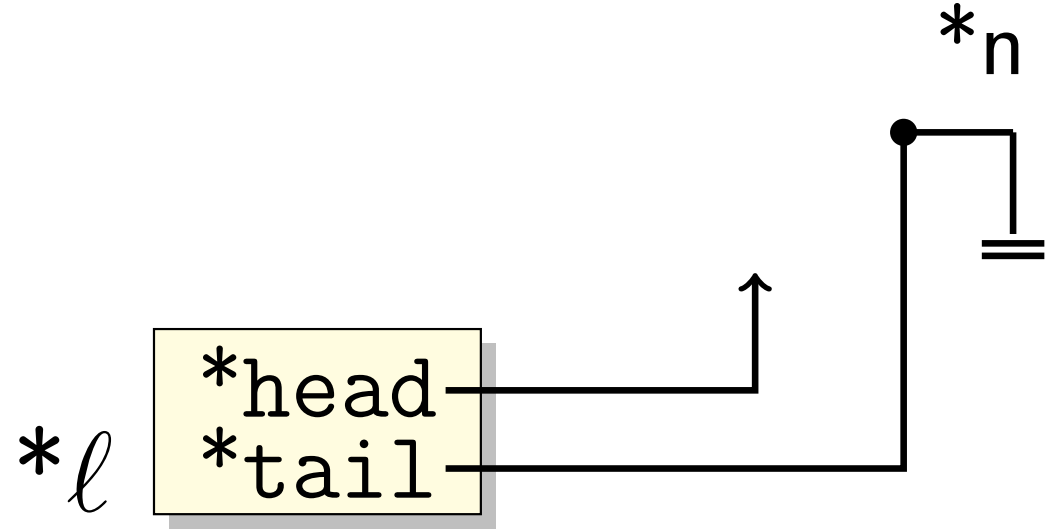
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 3

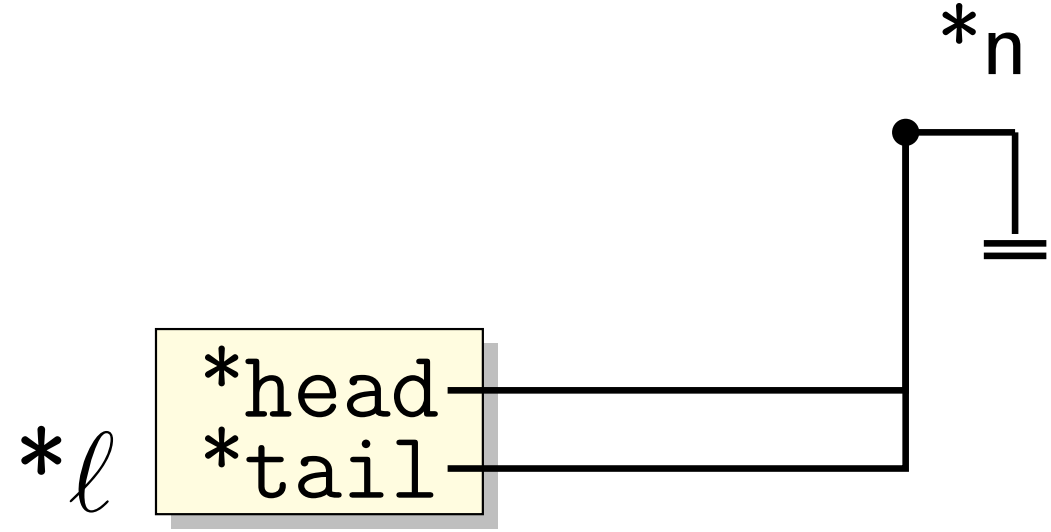
Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 3

Remover (início)




```
1: void remove-front (List *ℓ, int elem)
2:     if (ℓ →head ≠ NULL)
3:         Node *n = ℓ →head→next;
4:         free (ℓ →head);
5:         if (n ≠ NULL)
6:             n→prev = NULL;
7:         else
8:             ℓ →tail = NULL;
9:         ℓ →head = n;
```

Remover: 3