

LISTA ENCADEADA DINÂMICA

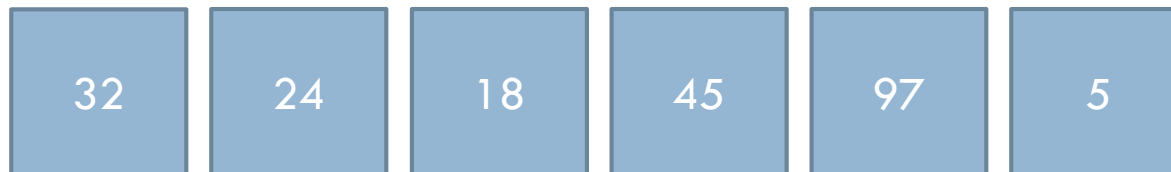
Prof. Muriel Mazzetto
Estrutura de Dados

Lista

2

- Uma lista pode ser descrita como:
 - ▣ Vazia.
 - ▣ $(a_1, a_2, a_3, \dots, a_N)$.
- Exemplos:
 - ▣ Listas de contatos, de compras, etc.

LISTA



Lista

3

- Operações comuns de uma Lista:
 - ▣ Criação da lista (alocar a estrutura);
 - ▣ Verificação se a lista está vazia;
 - ▣ Inserção de elemento;
 - ▣ Exclusão de elemento;
 - ▣ Acesso a um elemento;
 - ▣ Busca de um elemento;
 - ▣ Ordenar elementos;
 - ▣ Impressão da lista;
 - ▣ Destruição da lista (desalocar toda a estrutura);

Lista Dinâmica

4

- Essa estrutura possui diferentes nomes na literatura:
 - ▣ Lista Dinâmica.
 - ▣ Lista Encadeada.
- Definição: Uma estrutura do tipo “Lista” é uma sequência de elementos do mesmo tipo.
- Usam-se TADs para abstrair a sua estrutura interna.

Lista Dinâmica

5

❑ Características da Lista **Dinâmica**:

- ❑ Encadeada: o sucessor de um elemento não implica em uma posição física consecutiva na memória.
- ❑ Espaço de memória alocado no momento da inserção de elementos (em tempo de execução).
- ❑ O tamanho máximo dependerá de quantos blocos poderão ser **alocados dinamicamente** na memória.
- ❑ Possuir campo chave, que diferencia os elementos (CPF, RA, Identificação).

Lista Dinâmica

6

- Características da Lista **Dinâmica**:
 - ▣ A lista encadeada é formada por sucessivos elementos indicando a sequência.
 - ▣ Cada elemento **aponta** para seu próximo.
 - ▣ Chama-se encadeada por possuir um **encadeamento de elementos unitários**.

Lista Dinâmica: Simples

7

ELEMENTO

DADOS

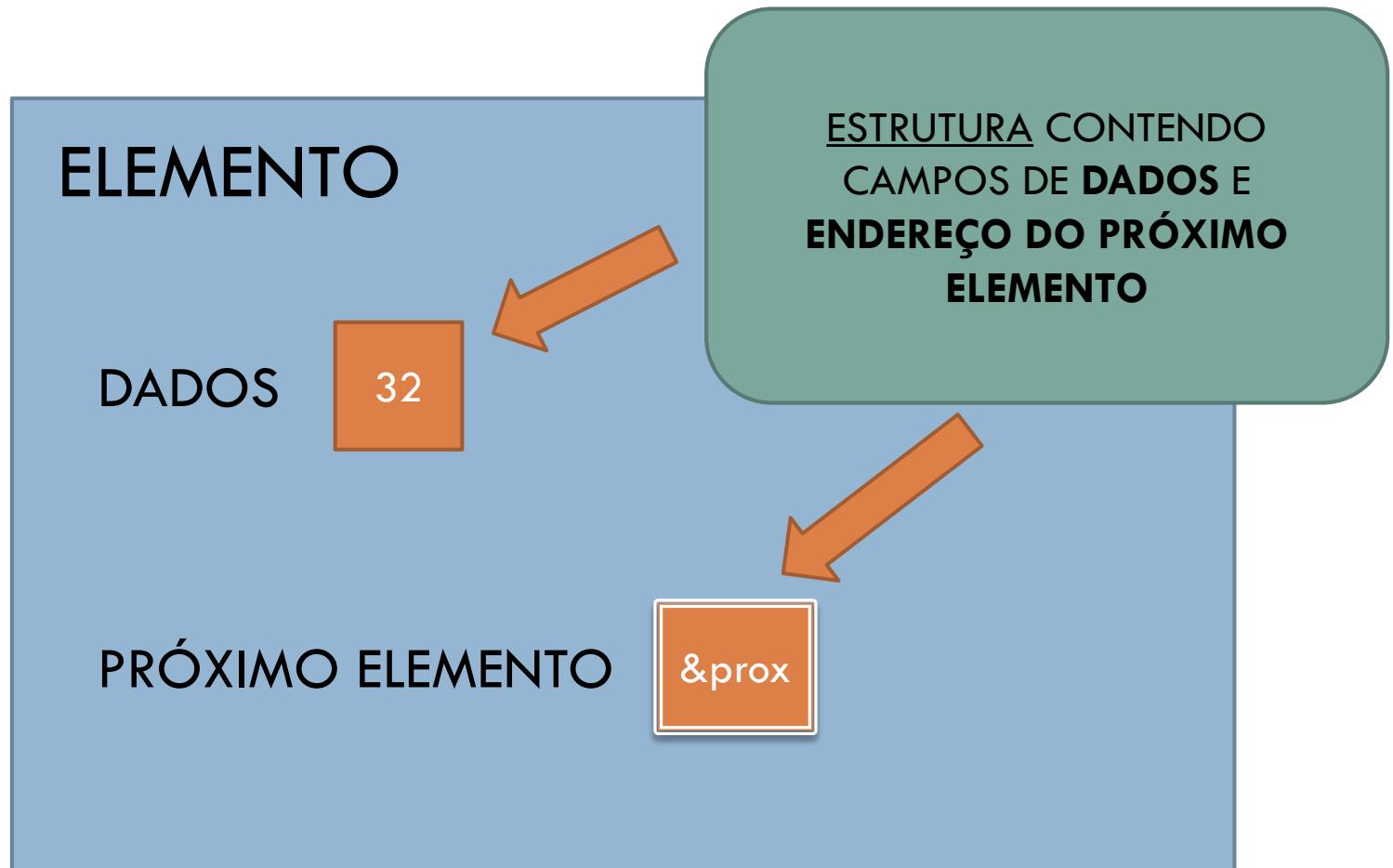
32

PRÓXIMO ELEMENTO

&prox

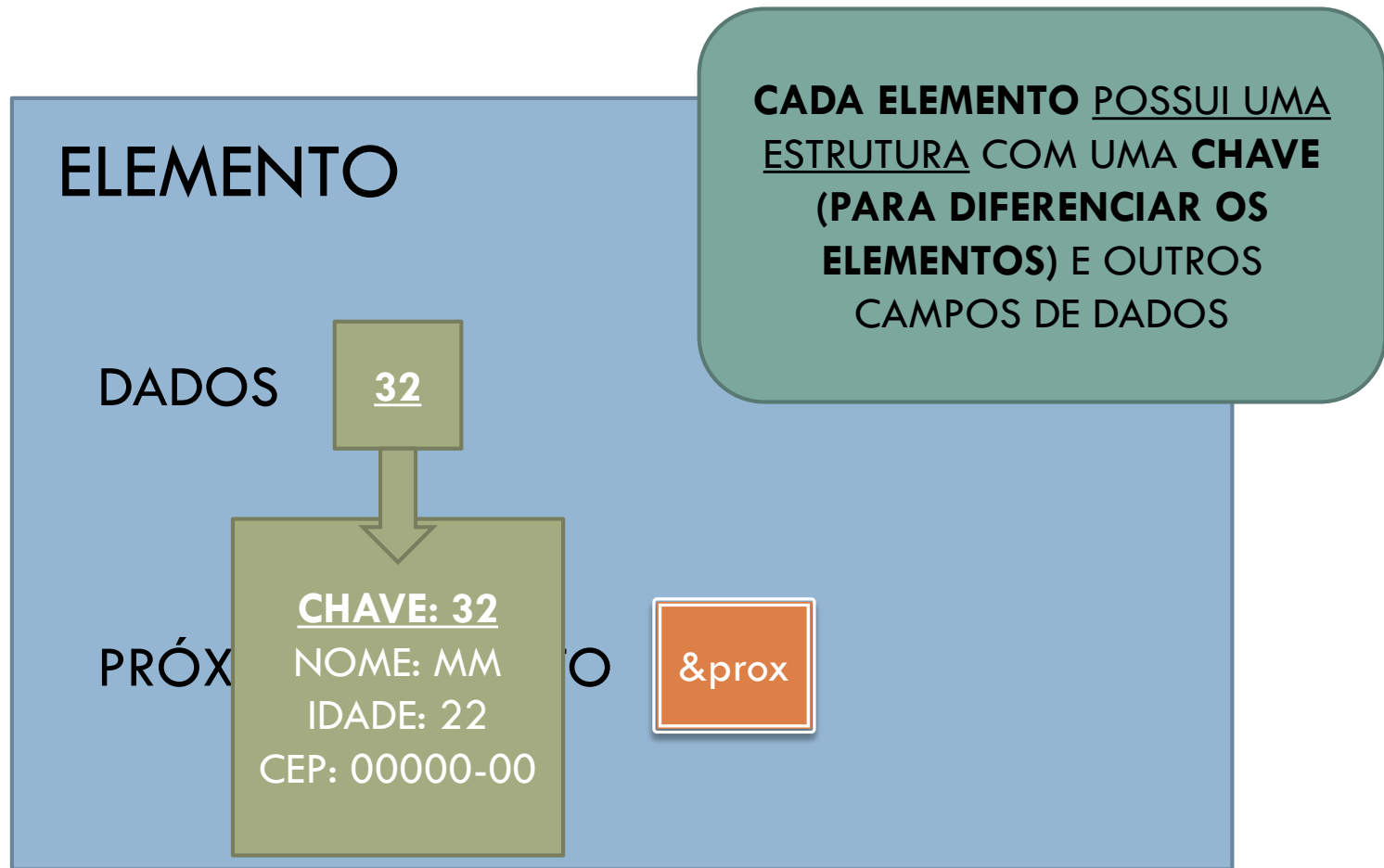
Lista Dinâmica: Simples

8



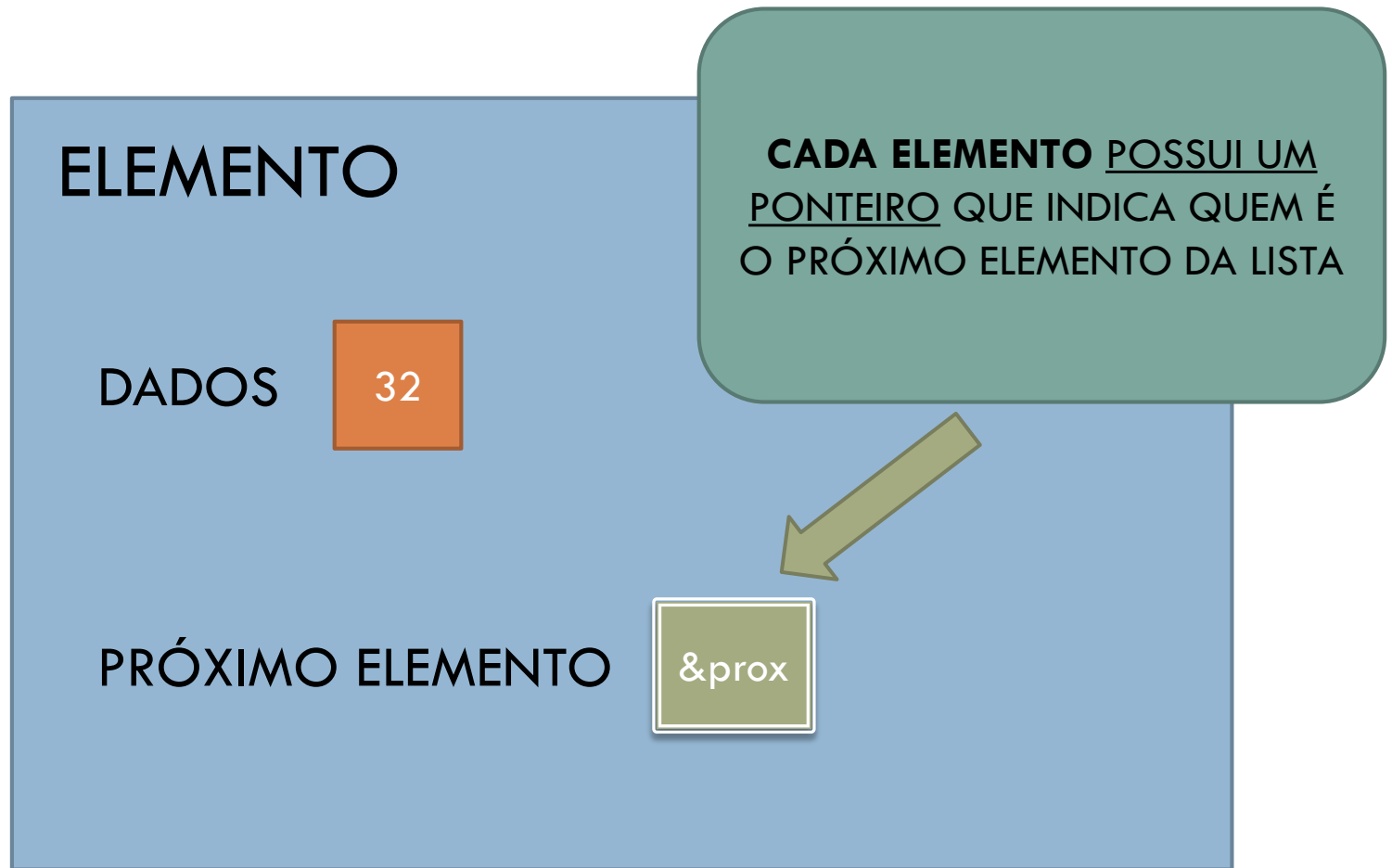
Lista Dinâmica: Simples

9



Lista Dinâmica: Simples

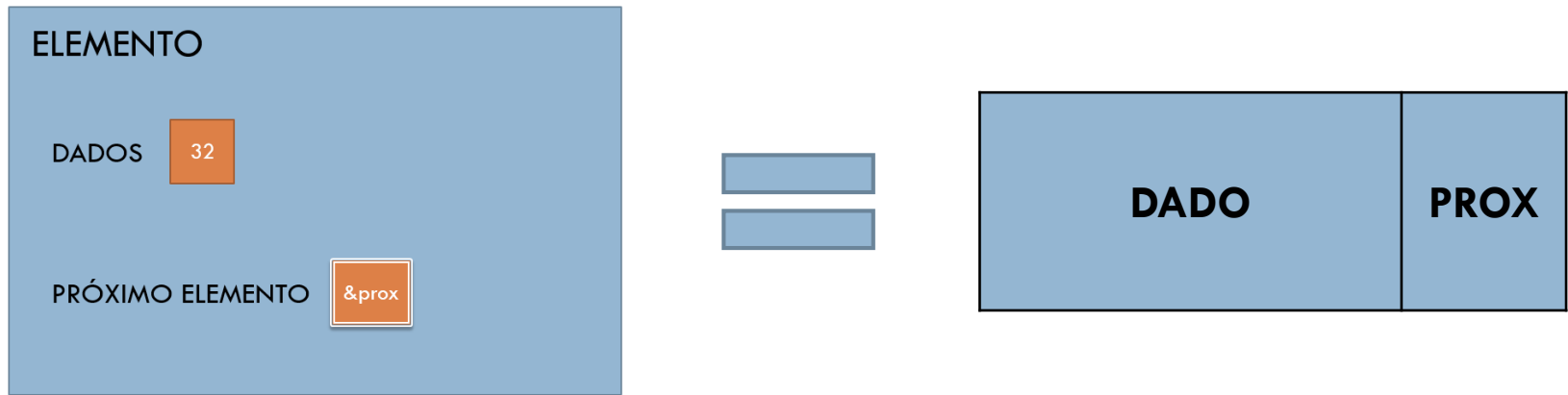
10



Lista Dinâmica: Simples

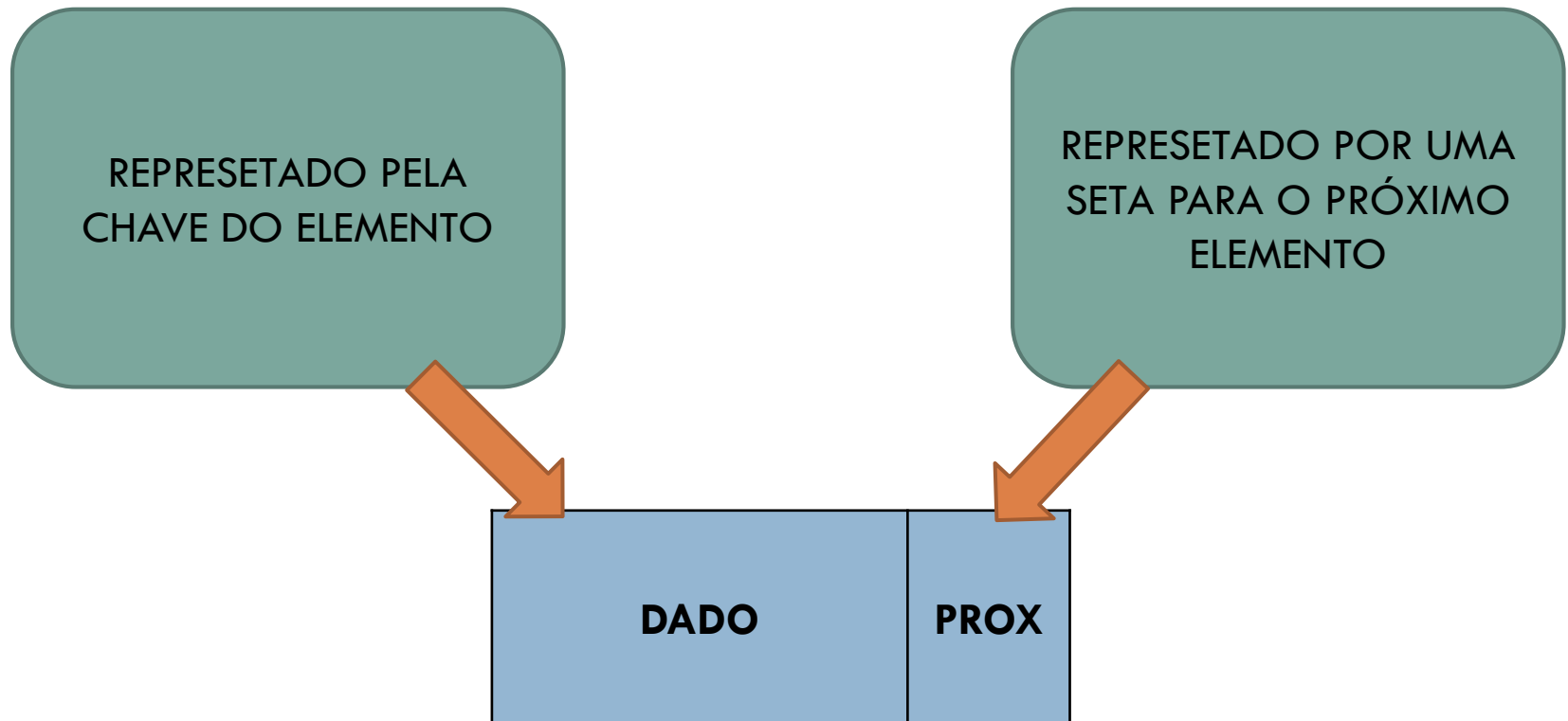
11

FORMA CONVENCIONAL DE SE
REPRESENTAR UM ELEMENTO DE
UMA LISTA ENCADEADA



Lista Dinâmica: Simples

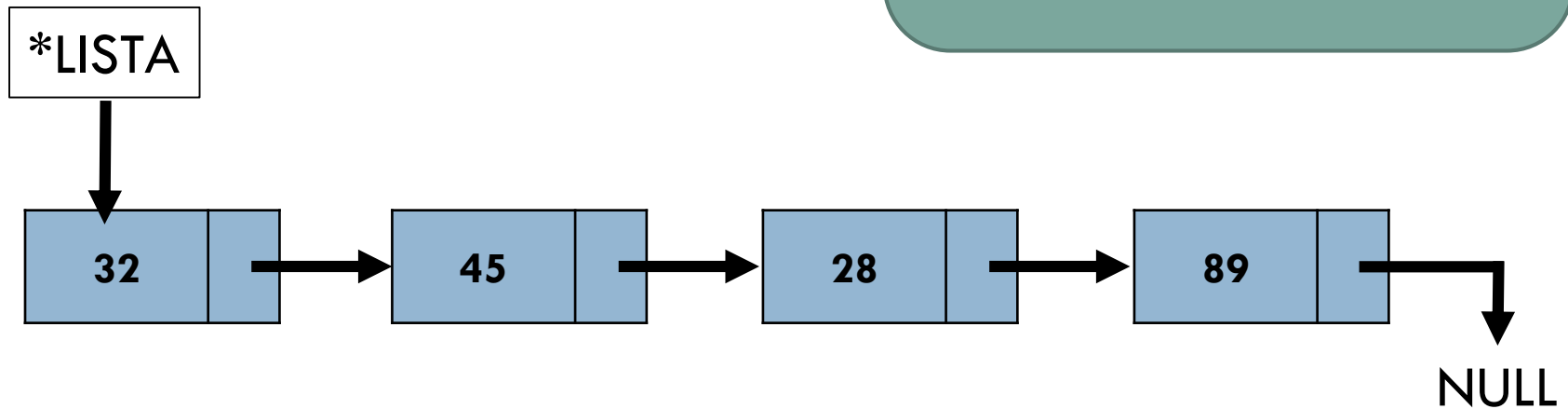
12



Lista Dinâmica: Simples

13

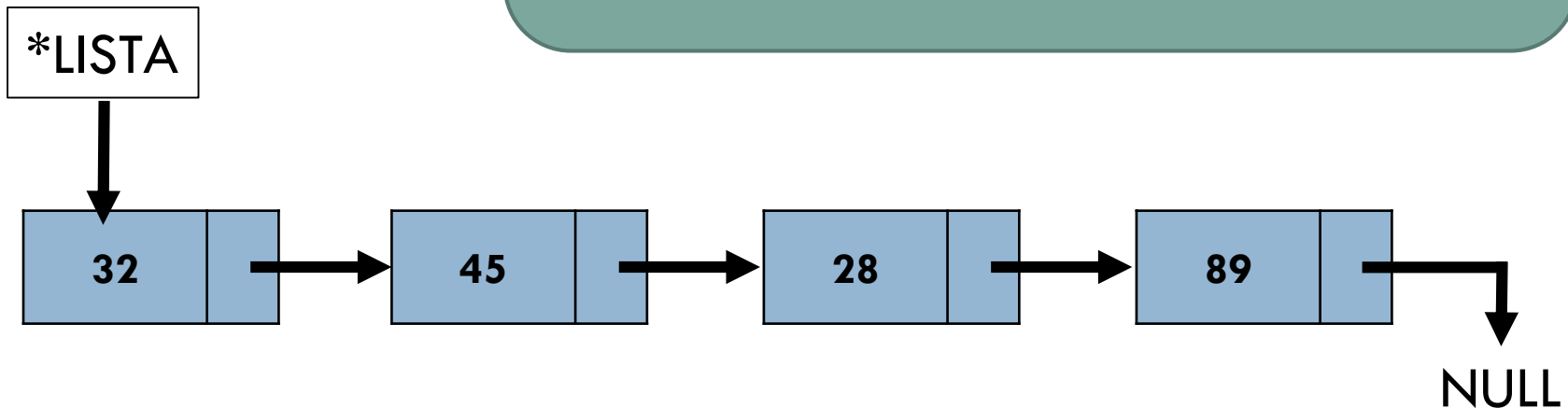
REPRESENTAÇÃO DE UMA LISTA
ENCADEADA



Lista Dinâmica: Simples

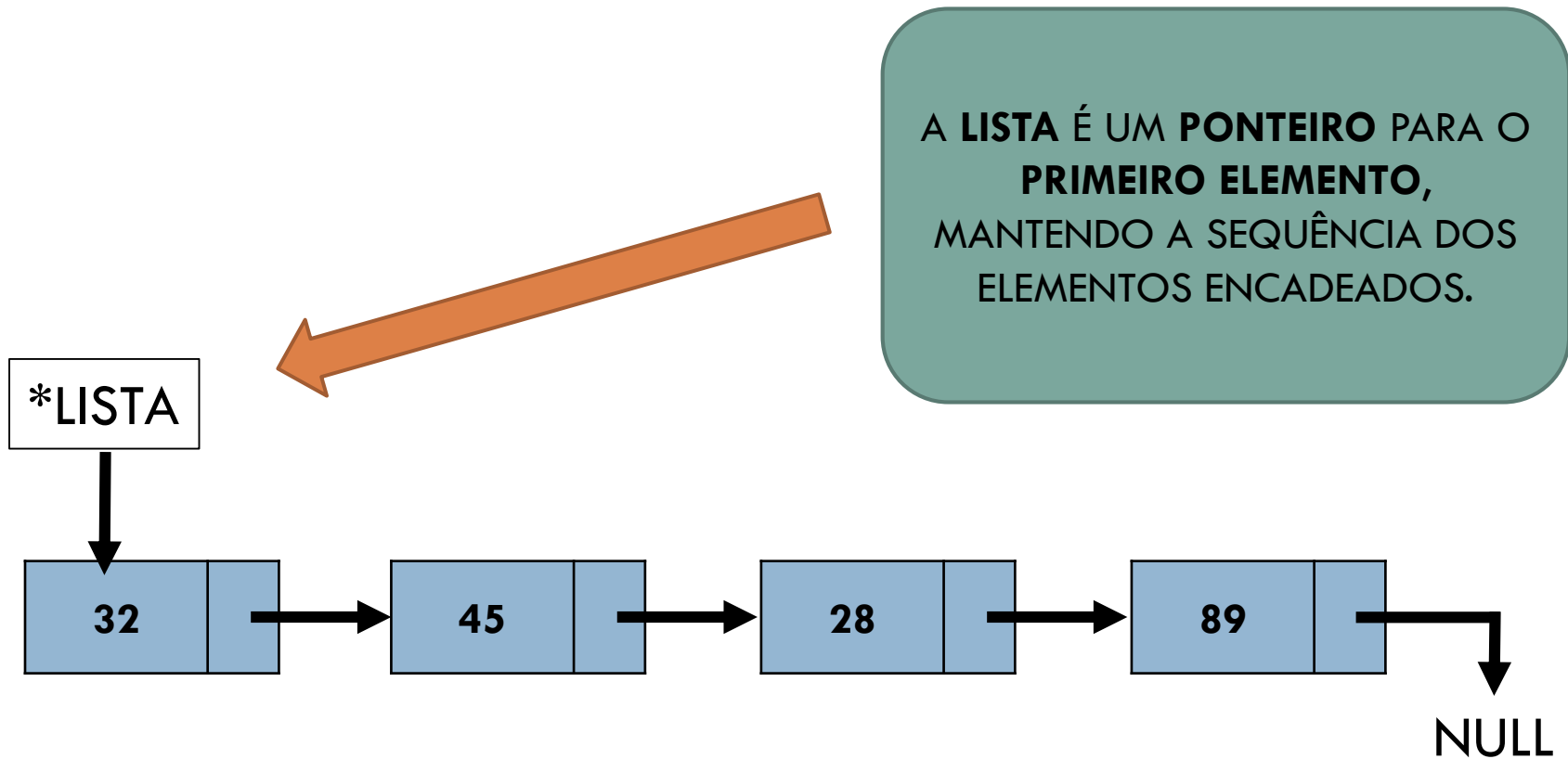
14

- NÃO EXISTE VETOR DE DADOS;
- NÃO SE TRABALHA COM ÍNDICE;
- SÃO ELEMENTOS DISPERSOS NA MEMÓRIA, QUE SE REFERENCIAM POR PONTEIROS.



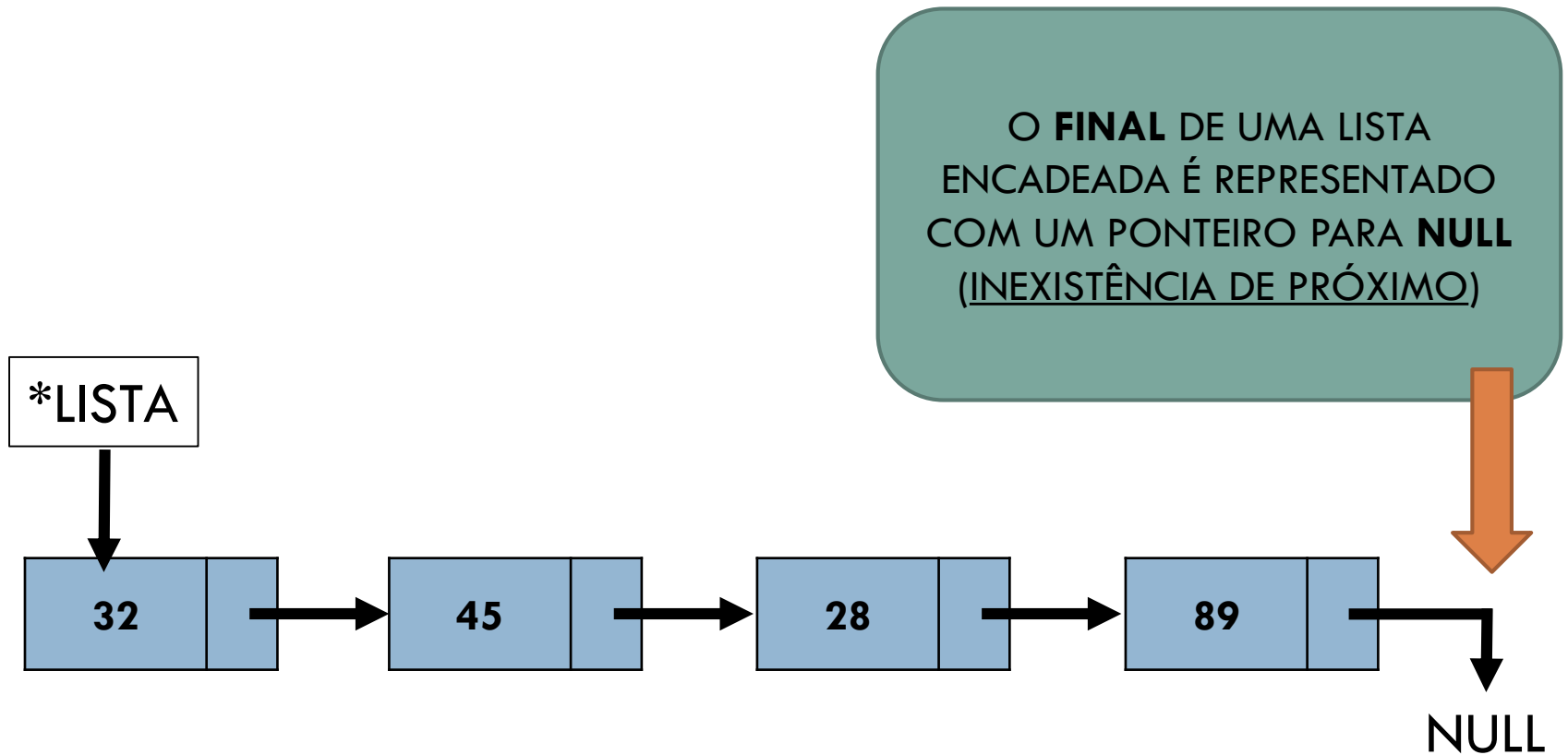
Lista Dinâmica: Simples

15



Lista Dinâmica: Simples

16



Lista Dinâmica: Exemplo

17

- Lista de alunos:
 - Matricula;
 - Nome;
 - Notas;
- Operações:
 - Criar lista;
 - Deletar lista;
 - Inserir aluno ordenado;
 - Remover aluno;
 - Imprimir lista;

Lista Dinâmica: Exemplo

18

□ Lista de alunos:

- Mat

- Non

- Not

□ Oper

- Cric

- Del

- Inse

- Rem

- Imprimir lista;

- main.c

- ListaDinamica.h

- ListaDinamica.c

Lista Dinâmica: Declarar tipos

19

```
//Arquivo ListaDinamica.h  
  
struct aluno{  
    int matricula;  
    char nome[30];  
    float n1,n2,n3;  
};
```

Lista Dinâmica: Declarar tipos

20

```
//Arquivo ListaDinamica.c

#include <stdio.h>
#include <stdlib.h>
#include "ListaDinEncad.h" //inclui os Protótipos

//Definição do tipo lista
struct elemento{
    struct aluno dados;
    struct elemento *prox;
};
typedef struct elemento Elem;
```

Lista Dinâmica: Declarar tipos

21

```
//Arquivo ListaDinamica.h

struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};

typedef struct elemento* Lista;
```

Lista Dinâmica: Declarar tipos

22

```
//Arquivo ListaDinamica.h

struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};

typedef struct elemento* Lista;
```

```
//Arquivo main.c
Lista* li = cria_lista();
```

Lista Dinâmica: Declarar tipos

23

```
//Arquivo ListaDinamica.h
```

```
struct aluno{  
    int matricula;  
    char nome[30];  
    float n1,n2,n3;  
};
```

```
typedef struct elemento* Lista;
```

- CADA ELEMENTO É REFERENCIADO POR UM PONTEIRO, POR SER ALOCADO DINAMICAMENTE.
- A LISTA SERÁ UM **PONTEIRO PARA PONTEIRO** DE ELEMENTO.

```
//Arquivo main.c  
Lista* li = cria_lista();
```

Lista Dinâmica: Declarar tipos

24

```
//Arquivo ListaDinamica.h

struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};

typedef struct elemento* Lista;

Lista* cria_lista();
void libera_lista(Lista* li);
int insere_lista_ordenada(Lista* li, struct aluno al);
int remove_lista(Lista* li, int mat);
void imprime_lista(Lista* li);
```


Lista Dinâmica: Criar lista

25

```
typedef struct elemento* Lista;
```

```
//Arquivo ListaDinamica.c
```

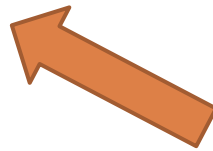
```
Lista* cria_lista(){  
    Lista* li = (Lista*) malloc(sizeof(Lista));  
    if(li != NULL)  
        *li = NULL;  
    return li;  
}
```

Lista Dinâmica: Criar lista

26

```
//Arquivo ListaDinamica.c
```

```
Lista* cria_lista(){  
    Lista* li = (Lista*) malloc(sizeof(Lista));  
    if(li != NULL)  
        *li = NULL;  
    return li;  
}
```



A **LISTA** FOI CRIADA **VAZIA**, SEM ELEMENTOS, PORTANTO APONTA PARA **NULL**.

Lista Dinâmica: Desalocar lista

27

```
//Arquivo ListaDinamica.c

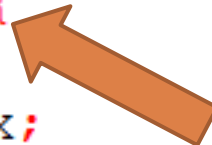
void libera_lista(Lista* li) {
    if(li != NULL) {
        Elem* no;
        while((*li) != NULL) {
            no = *li;
            *li = (*li)->prox;
            free(no);
        }
        free(li);
    }
}
```

Lista Dinâmica: Desalocar lista

28

```
//Arquivo ListaDinamica.c
```

```
void libera_lista(Lista* li) {  
    if(li != NULL) {  
        Elem* no;  
        while ((*li) != NULL) {  
            no = *li;  
            *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

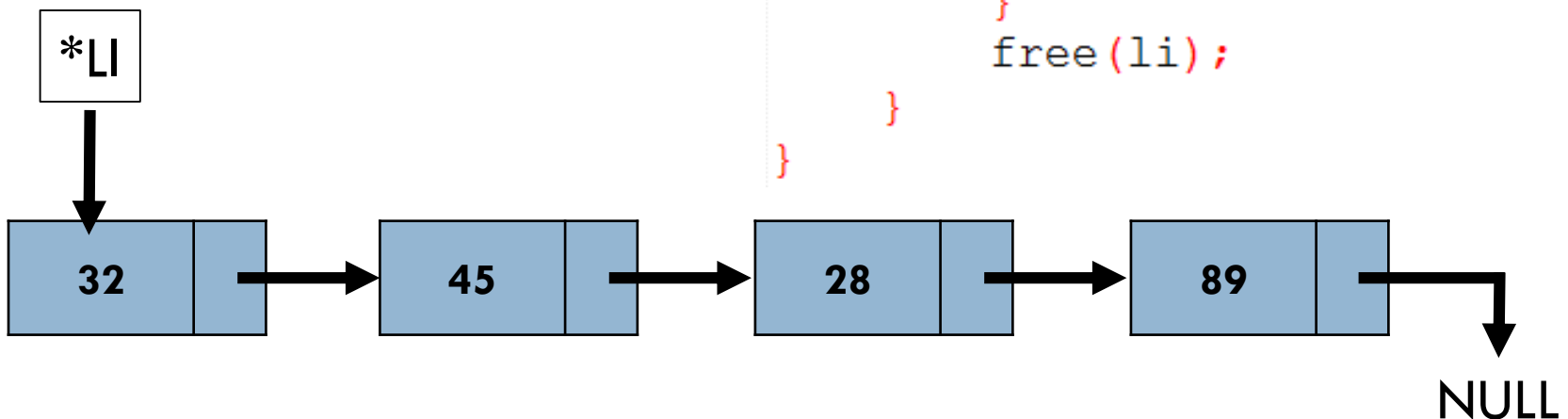
- 
- LISTA FORMADA POR **VARIOS ELEMENTOS** DISPERSOS NA MEMÓRIA.
 - NECESSÁRIO **DESALOCAR CADA UM DOS ELEMENTOS**.

Lista Dinâmica: Desalocar lista

29

```
//Arquivo ListaDinamica.c

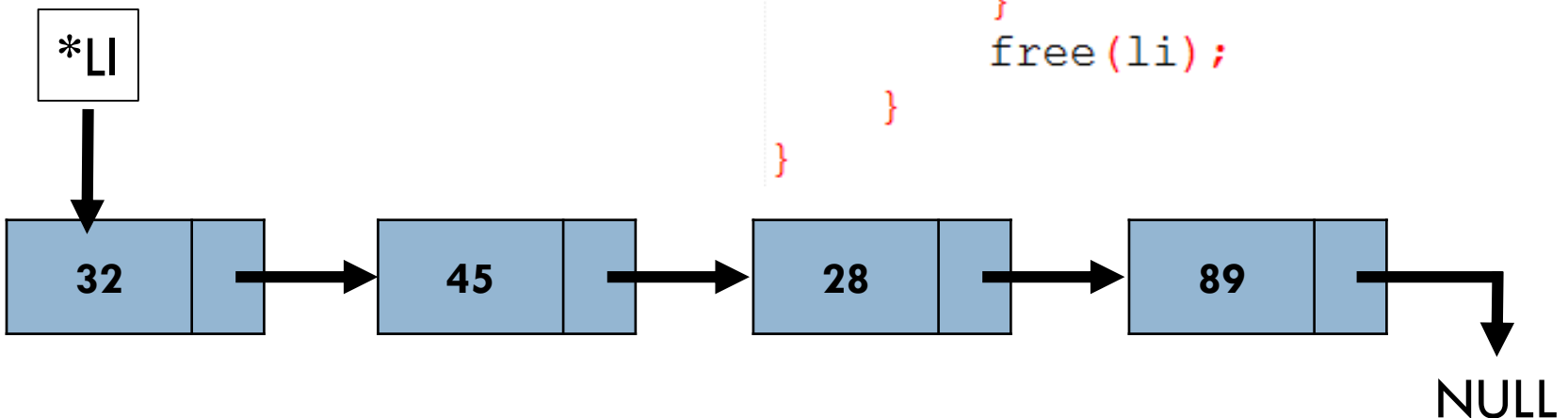
void libera_lista(Lista* li){
    if(li != NULL){
        Elem* no;
        while((*li) != NULL){
            no = *li;
            *li = (*li)->prox;
            free(no);
        }
        free(li);
    }
}
```



Lista Dinâmica: Desalocar lista

30

```
//Arquivo ListaDinamica.c  
  
void libera_lista(Lista* li) {  
    → if(li != NULL) {  
        Elem* no;  
        while ((*li) != NULL) {  
            no = *li;  
            *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

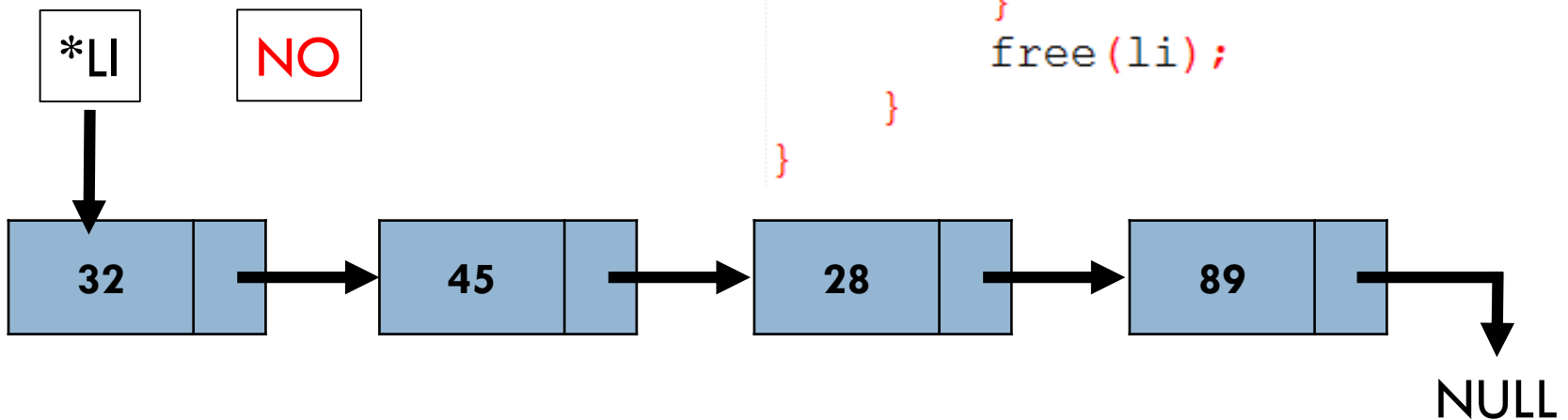


Lista Dinâmica: Desalocar lista

31

//Arquivo ListaDinamica.c

```
void libera_lista(Lista* li){  
    if(li != NULL){  
        → Elem* no;  
        while ((*li) != NULL){  
            no = *li;  
            *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

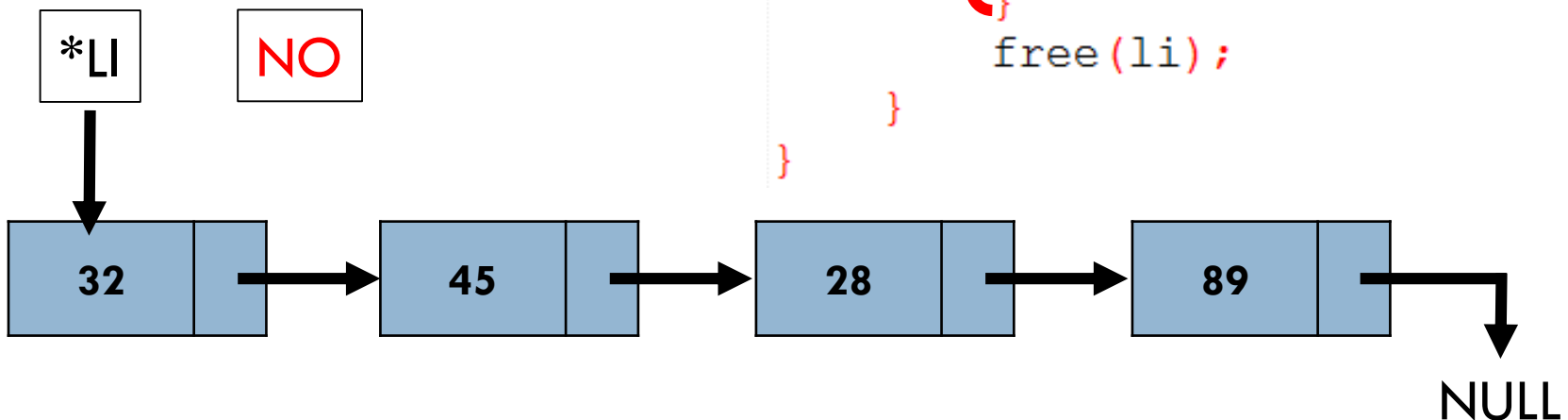


Lista Dinâmica: Desalocar lista

32

```
//Arquivo ListaDinamica.c
```

```
void libera_lista(Lista* li){  
    if(li != NULL){  
        Elem* no;  
        → while ((*li) != NULL) {  
            no = *li;  
            *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

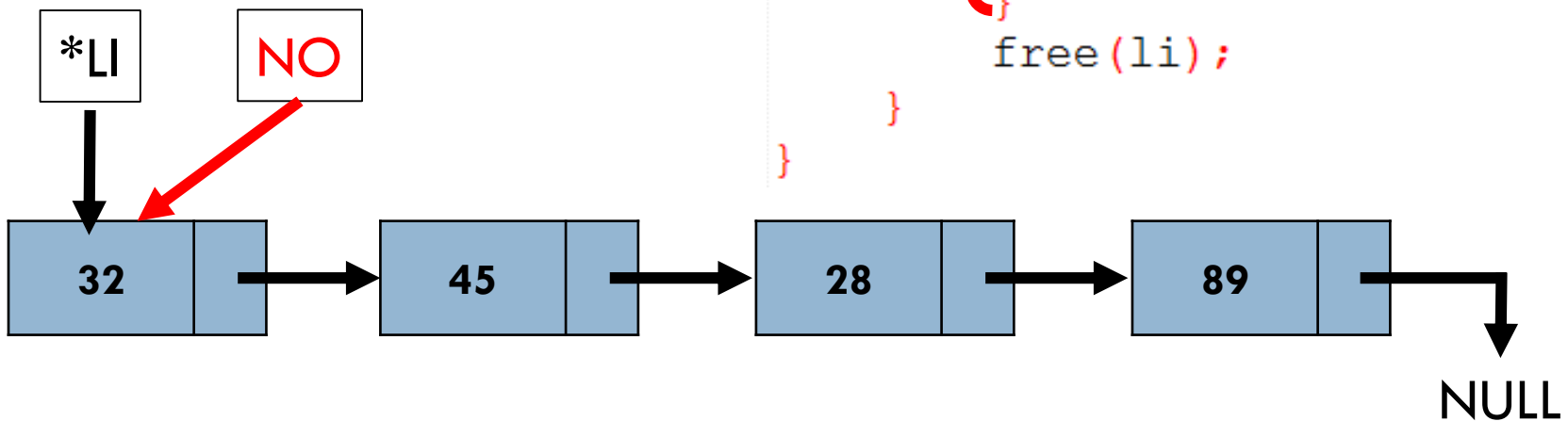


Lista Dinâmica: Desalocar lista

33

```
//Arquivo ListaDinamica.c
```

```
void libera_lista(Lista* li){  
    if(li != NULL){  
        Elem* no;  
        while ((*li) != NULL){  
            no = *li;  
            *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

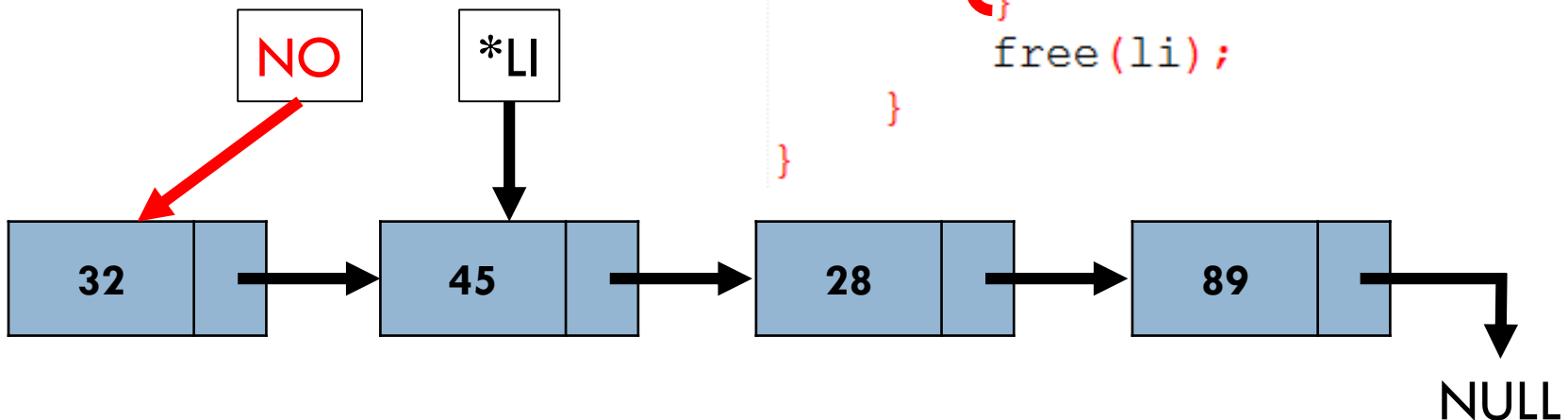


Lista Dinâmica: Desalocar lista

34

```
//Arquivo ListaDinamica.c
```

```
void libera_lista(Lista* li){  
    if(li != NULL){  
        Elem* no;  
        while ((*li) != NULL){  
            no = *li;  
            → *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

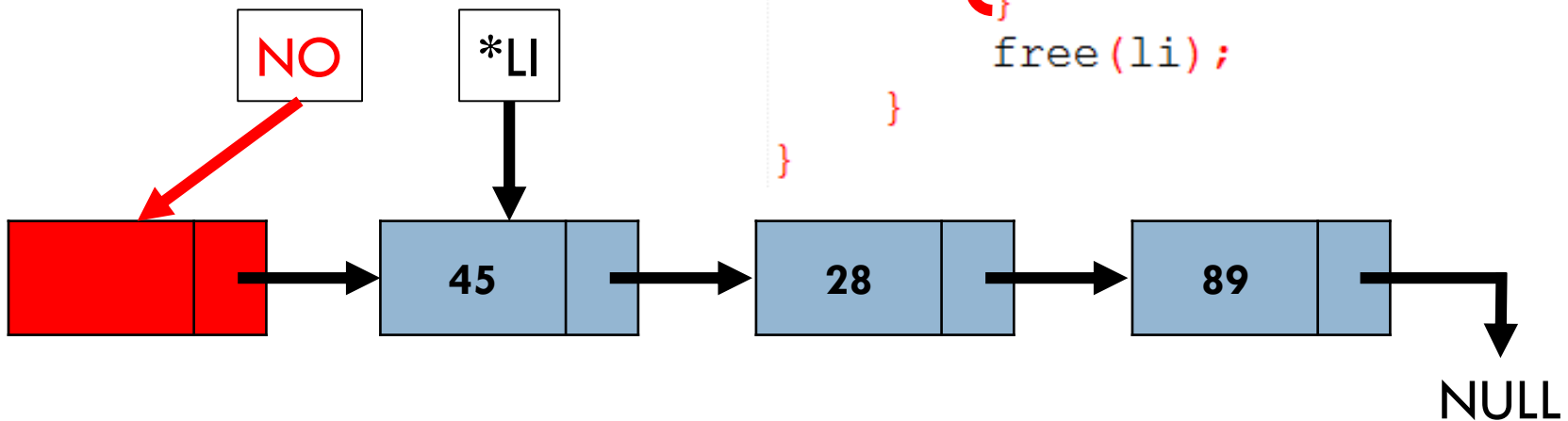


Lista Dinâmica: Desalocar lista

35

```
//Arquivo ListaDinamica.c
```

```
void libera_lista(Lista* li){  
    if(li != NULL){  
        Elem* no;  
        while ((*li) != NULL){  
            no = *li;  
            *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

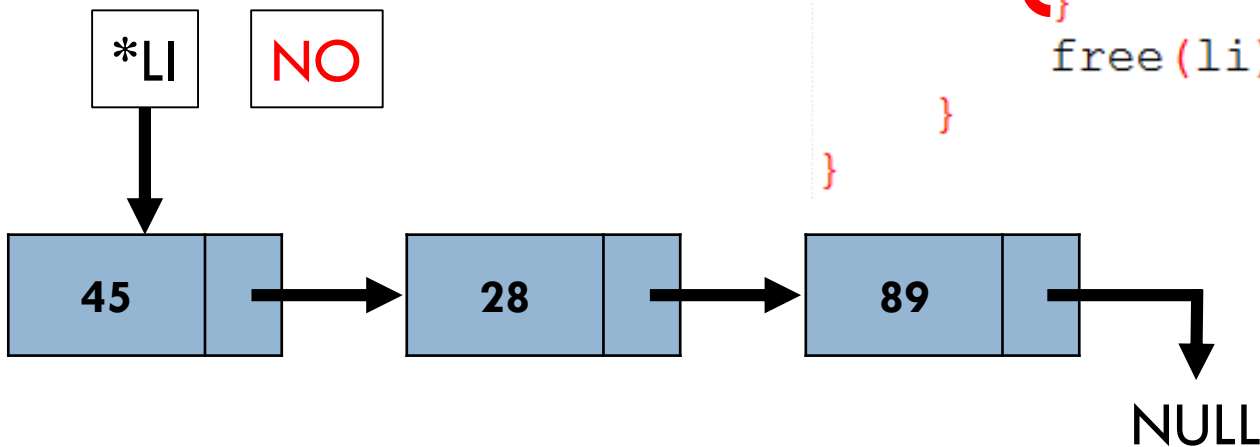


Lista Dinâmica: Desalocar lista

36

//Arquivo ListaDinamica.c

```
void libera_lista(Lista* li){  
    if(li != NULL){  
        Elem* no;  
        → while ((*li) != NULL) {  
            no = *li;  
            *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

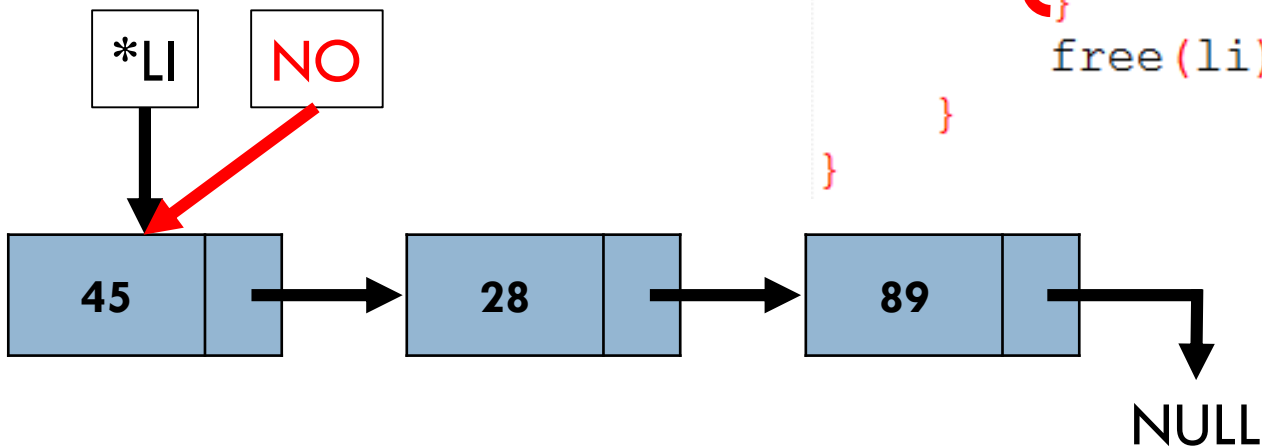


Lista Dinâmica: Desalocar lista

37

```
//Arquivo ListaDinamica.c
```

```
void libera_lista(Lista* li){  
    if(li != NULL){  
        Elem* no;  
        while ((*li) != NULL){  
            → no = *li;  
            *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

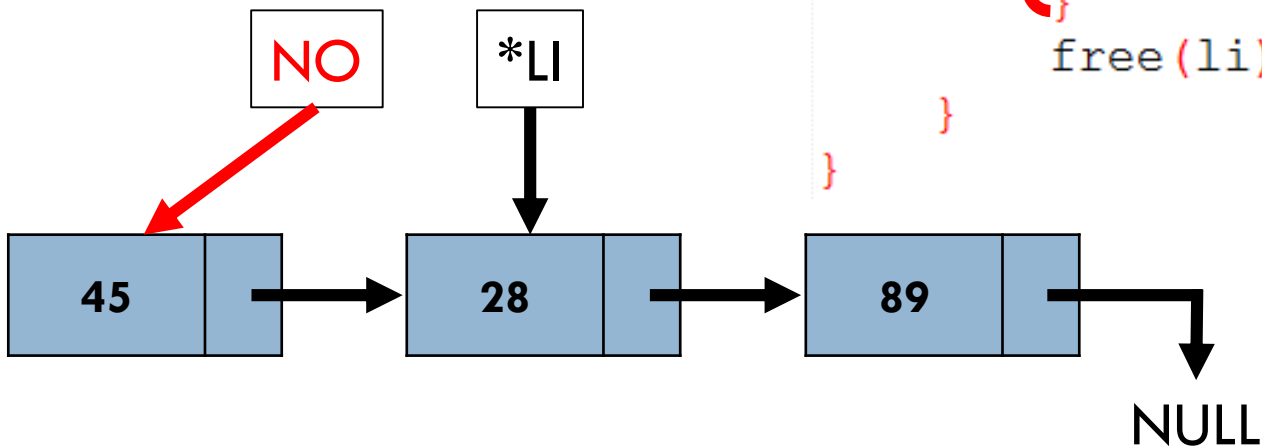


Lista Dinâmica: Desalocar lista

38

```
//Arquivo ListaDinamica.c
```

```
void libera_lista(Lista* li){  
    if(li != NULL){  
        Elem* no;  
        while ((*li) != NULL){  
            no = *li;  
            *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

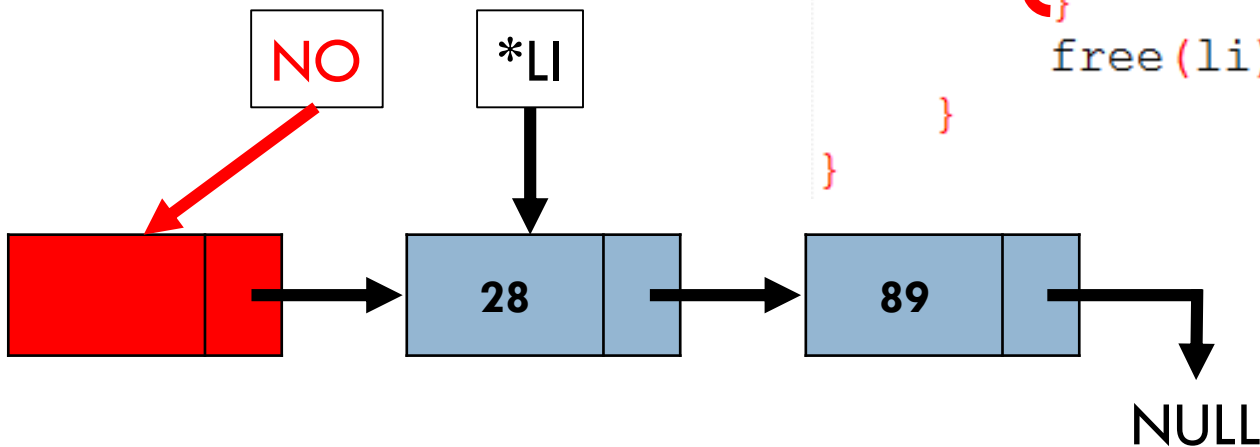


Lista Dinâmica: Desalocar lista

39

```
//Arquivo ListaDinamica.c
```

```
void libera_lista(Lista* li){  
    if(li != NULL){  
        Elem* no;  
        while ((*li) != NULL){  
            no = *li;  
            *li = (*li)->prox;  
            → free(no);  
        }  
        free(li);  
    }  
}
```

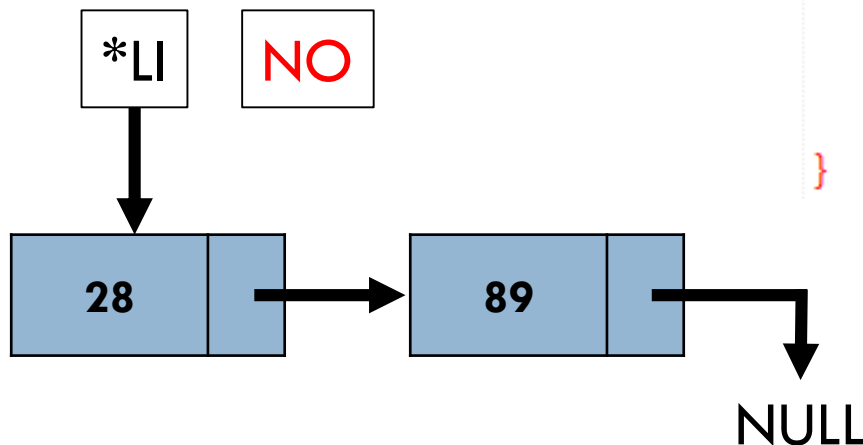


Lista Dinâmica: Desalocar lista

40

```
//Arquivo ListaDinamica.c
```

```
void libera_lista(Lista* li) {  
    if(li != NULL) {  
        Elem* no;  
        → while ((*li) != NULL) {  
            no = *li;  
            *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

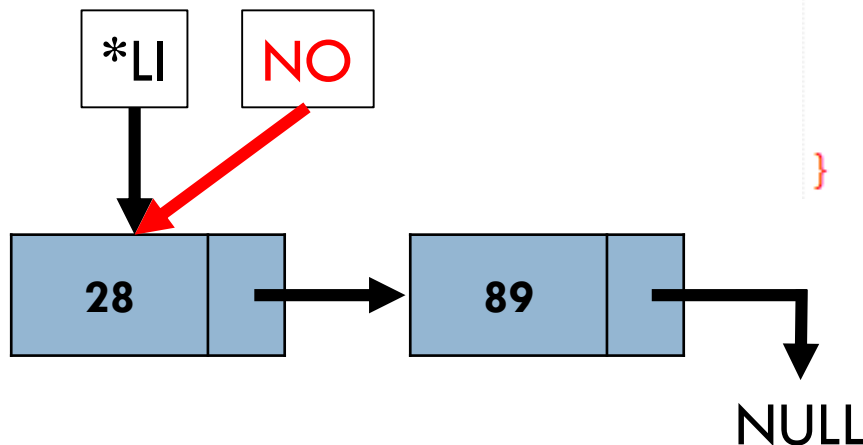


Lista Dinâmica: Desalocar lista

41

```
//Arquivo ListaDinamica.c
```

```
void libera_lista(Lista* li){  
    if(li != NULL){  
        Elem* no;  
        while ((*li) != NULL){  
            → no = *li;  
            *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

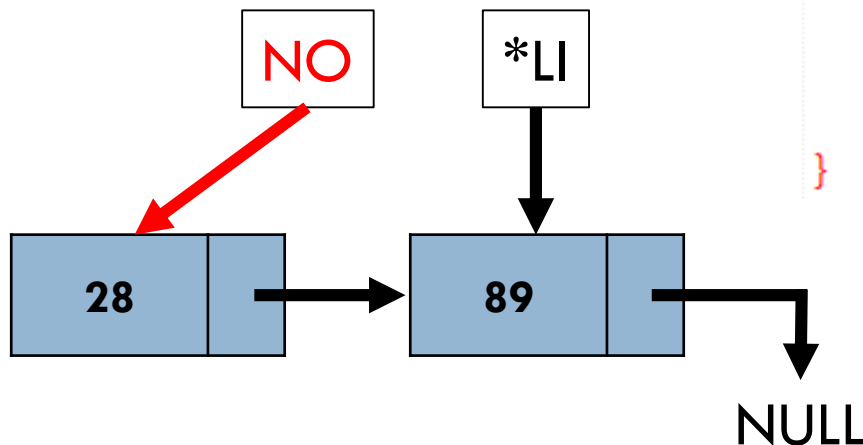


Lista Dinâmica: Desalocar lista

42

```
//Arquivo ListaDinamica.c
```

```
void libera_lista(Lista* li){  
    if(li != NULL){  
        Elem* no;  
        while ((*li) != NULL){  
            no = *li;  
            → *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

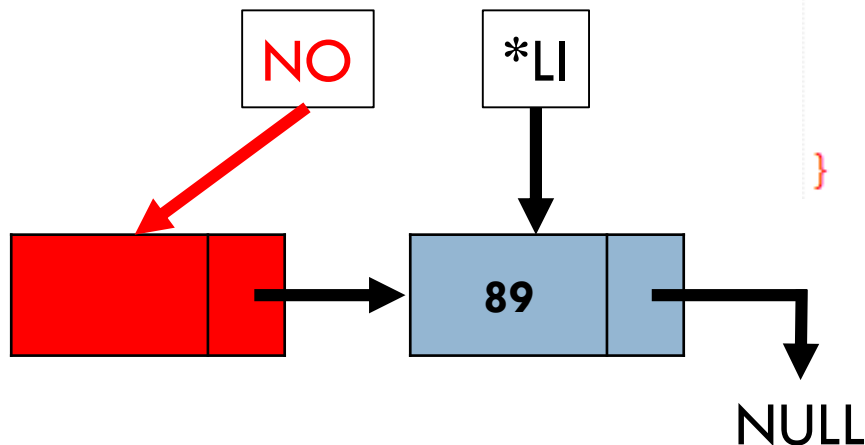


Lista Dinâmica: Desalocar lista

43

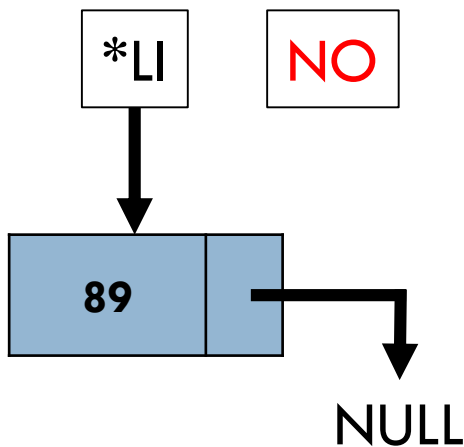
```
//Arquivo ListaDinamica.c
```

```
void libera_lista(Lista* li){  
    if(li != NULL){  
        Elem* no;  
        {  
            while ((*li) != NULL){  
                no = *li;  
                *li = (*li)->prox;  
                → free(no);  
            }  
        }  
        free(li);  
    }  
}
```



Lista Dinâmica: Desalocar lista

44

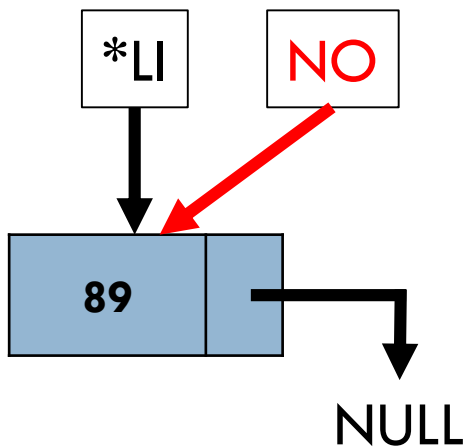


```
//Arquivo ListaDinamica.c

void libera_lista(Lista* li){
    if(li != NULL){
        Elem* no;
        → while ((*li) != NULL) {
            no = *li;
            *li = (*li)->prox;
            free(no);
        }
        free(li);
    }
}
```

Lista Dinâmica: Desalocar lista

45

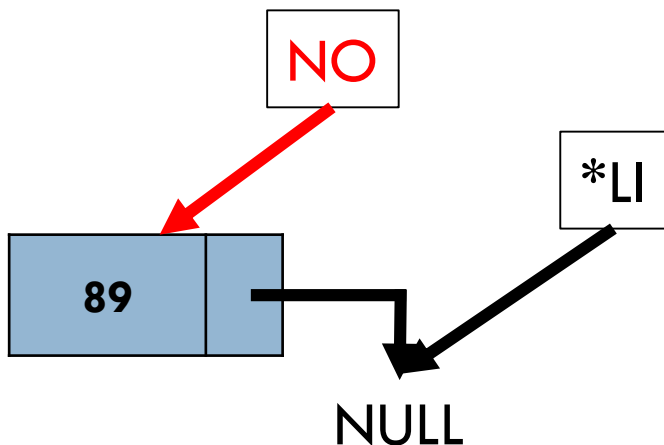


```
//Arquivo ListaDinamica.c
```

```
void libera_lista(Lista* li){  
    if(li != NULL){  
        Elem* no;  
        while ((*li) != NULL){  
            → no = *li;  
            *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

Lista Dinâmica: Desalocar lista

46

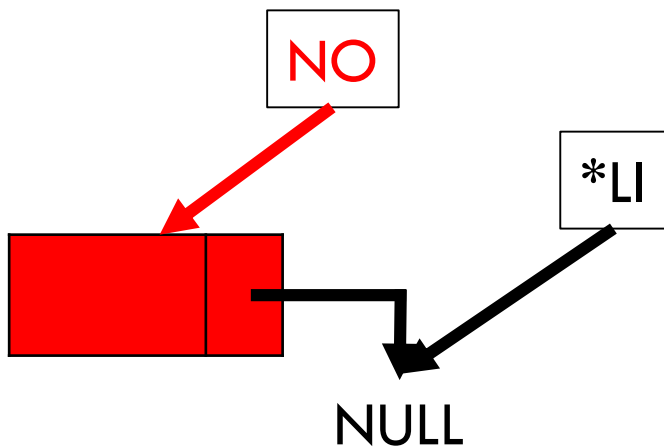


```
//Arquivo ListaDinamica.c
```

```
void libera_lista(Lista* li){  
    if(li != NULL){  
        Elem* no;  
        while ((*li) != NULL){  
            no = *li;  
            → *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

Lista Dinâmica: Desalocar lista

47

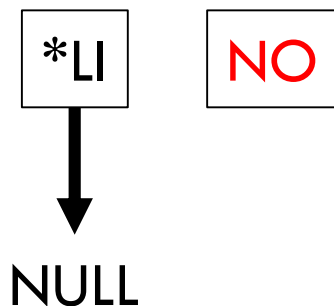


```
//Arquivo ListaDinamica.c

void libera_lista(Lista* li){
    if(li != NULL){
        Elem* no;
        {
            while ((*li) != NULL){
                no = *li;
                *li = (*li)->prox;
                → free(no);
            }
        }
        free(li);
    }
}
```

Lista Dinâmica: Desalocar lista

48

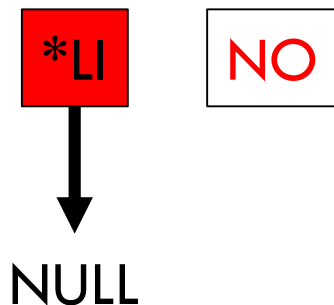


```
//Arquivo ListaDinamica.c

void libera_lista(Lista* li) {
    if(li != NULL) {
        Elem* no;
        → while ((*li) != NULL) {
            no = *li;
            *li = (*li)->prox;
            free(no);
        }
        free(li);
    }
}
```


Lista Dinâmica: Desalocar lista

49

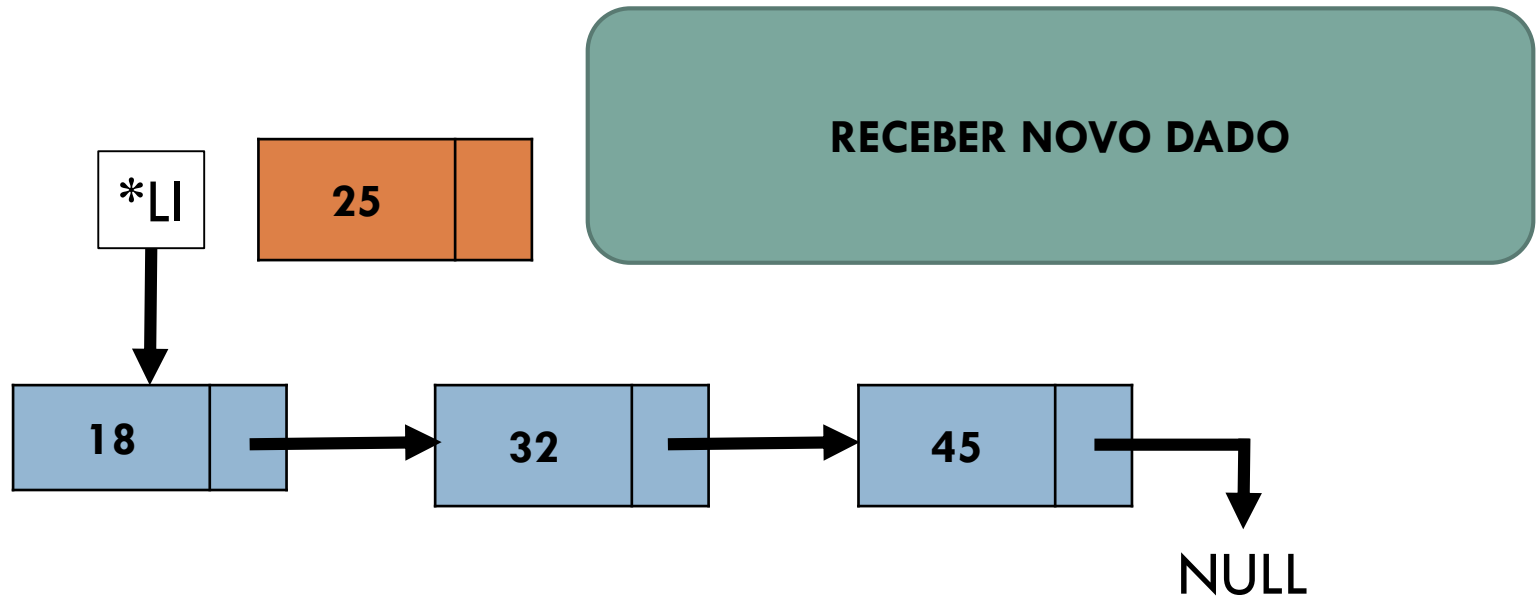


```
//Arquivo ListaDinamica.c

void libera_lista(Lista* li){
    if(li != NULL){
        Elem* no;
        while((*li) != NULL){
            no = *li;
            *li = (*li)->prox;
            free(no);
        }
        free(li);
    }
}
```

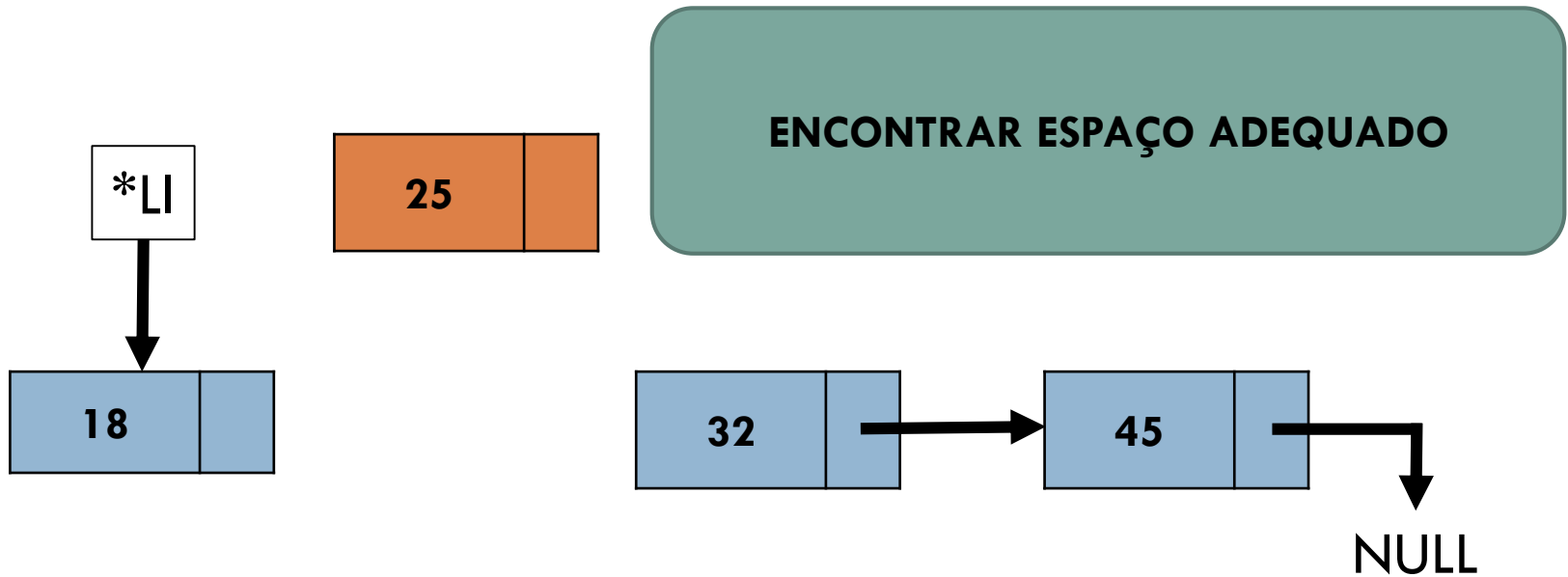
Lista Dinâmica: Inserção Ordenada

50



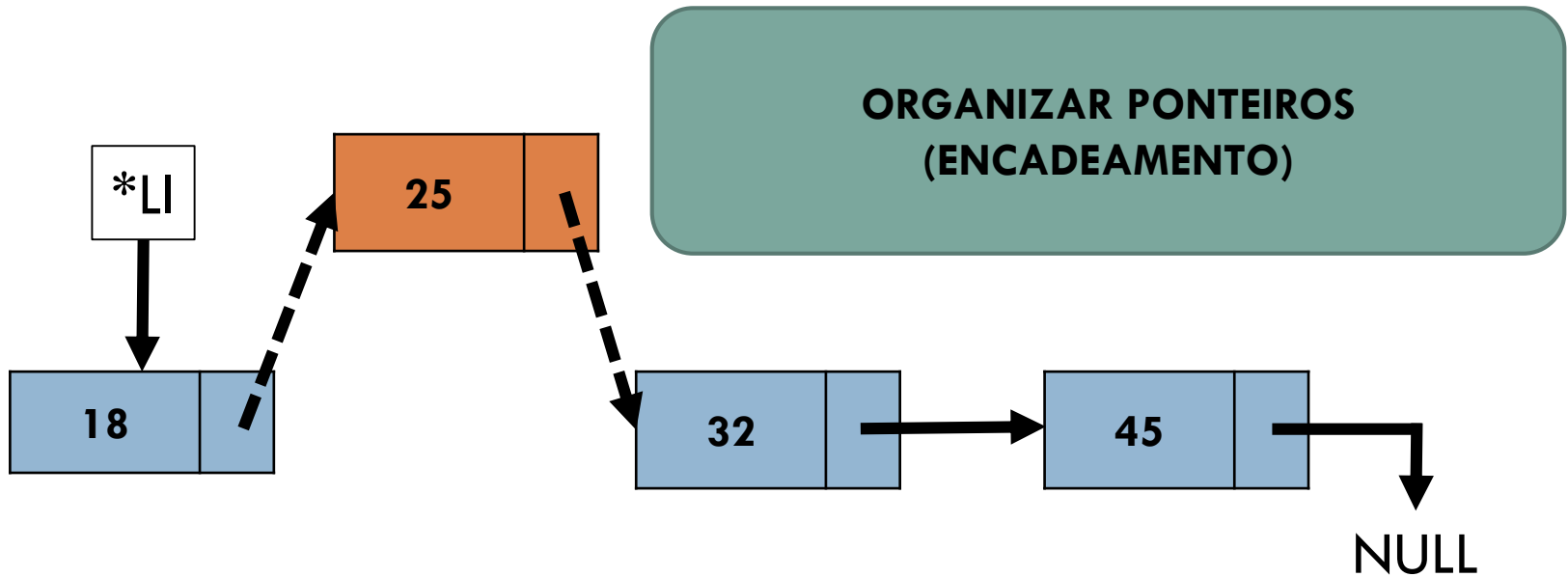
Lista Dinâmica: Inserção Ordenada

51



Lista Dinâmica: Inserção Ordenada

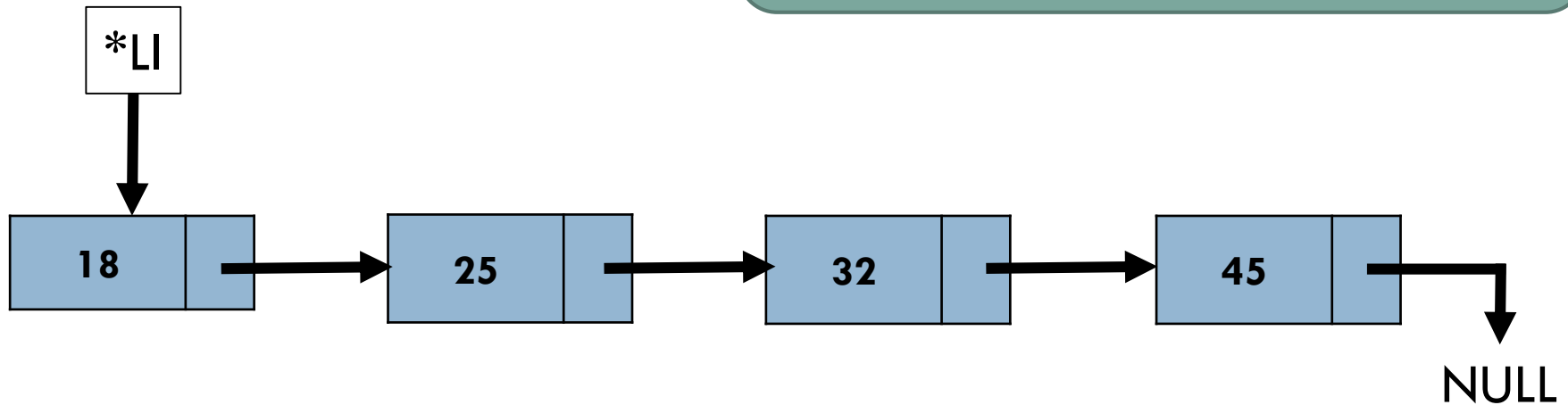
52



Lista Dinâmica: Inserção Ordenada

53

DADO INSERIDO MANTENDO ORDEM



Lista Dinâmica: Inserção Ordenada

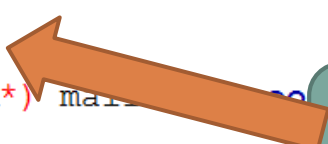
54

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if((*li) == NULL){//lista vazia: insere início
        no->prox = NULL;
        *li = no;
        return 1;
    }
    else{
        Elem *ant, *atual = *li;
        while(atual != NULL && atual->dados.matricula < al.matricula){
            ant = atual;
            atual = atual->prox;
        }
        if(atual == *li){//insere início
            no->prox = (*li);
            *li = no;
        }else{
            no->prox = atual;
            ant->prox = no;
        }
        return 1;
    }
}
```

Lista Dinâmica: Inserção Ordenada

55

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if((*li) == NULL){//lista vazia: insere início
        no->prox = NULL;
        *li = no;
        return 1;
    }
    else{
        Elem *ant, *atual = *li;
        while(atual != NULL && atual->dados.matricula < al.matricula){
            ant = atual;
            atual = atual->prox;
        }
        if(atual == *li){//insere início
            no->prox = (*li);
            *li = no;
        }else{
            no->prox = atual;
            ant->prox = no;
        }
    }
    return 1;
}
```



VERIFICAR SE A LISTA EXISTE

Lista Dinâmica: Inserção Ordenada

56

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if((*li) == NULL){//lista vazia:
        no->prox = NULL;
        *li = no;
        return 1;
    }
    else{
        Elem *ant, *atual = *li;
        while(atual != NULL && atual->dados.matricula < al.matricula){
            ant = atual;
            atual = atual->prox;
        }
        if(atual == *li){//insere início
            no->prox = (*li);
            *li = no;
        }else{
            no->prox = atual;
            ant->prox = no;
        }
        return 1;
    }
}
```



**ALOCAR ELEMENTO QUE
RECEBERÁ OS DADOS**

Lista Dinâmica: Inserção Ordenada

57

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if((*li) == NULL){//lista vazia: insere início
        no->prox = NULL;
        *li = no;
        return 1;
    }
    else{
        Elem *ant, *atual = *li;
        while(atual != NULL && atual->dados < al.dados){
            ant = atual;
            atual = atual->prox;
        }
        if(atual == *li){//insere início
            no->prox = (*li);
            *li = no;
        }
        else{
            no->prox = atual;
            ant->prox = no;
        }
    }
    return 1;
}
```



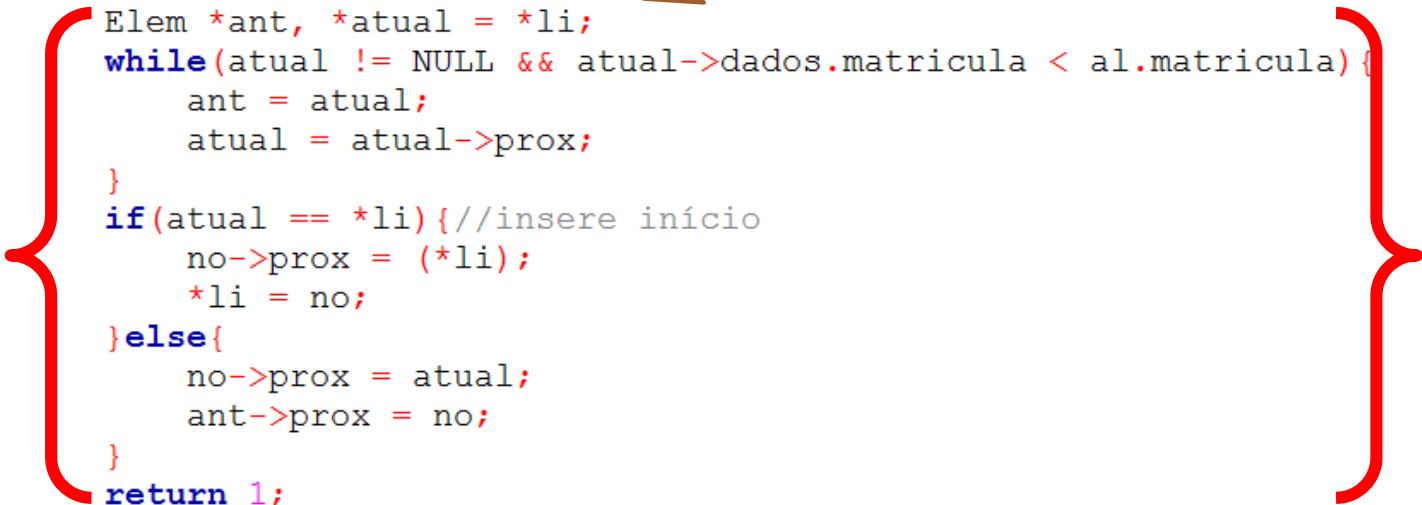
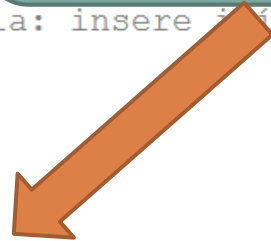
CASO ESPECIAL: LISA VAZIA,
RECEBE O PRIMEIRO ELEMENTO
NA LISTA.

Lista Dinâmica: Inserção Ordenada

58

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if((*li) == NULL){//lista vazia: insere no início
        no->prox = NULL;
        *li = no;
        return 1;
    }
    else{
        Elem *ant, *atual = *li;
        while(atual != NULL && atual->dados.matricula < al.matricula){
            ant = atual;
            atual = atual->prox;
        }
        if(atual == *li){//insere início
            no->prox = (*li);
            *li = no;
        }else{
            no->prox = atual;
            ant->prox = no;
        }
        return 1;
    }
}
```

- PROCURAR LOCAL IDEAL;
- REORGANIZA PONTEIROS PARA INSERIR.



Lista Dinâmica: Inserção Ordenada

59

AL: 32

*LI



NULL

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```

Lista Dinâmica: Inserção Ordenada

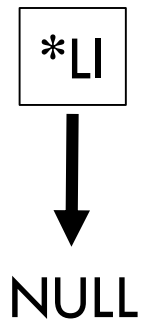
60

AL: 32

```
//Arquivo ListaDinamica.c  
int insere_lista_ordenada(Lista* li, struct aluno al){  
    → if(li == NULL)  
        return 0;
```

```
    Elem *no = (Elem*) malloc(sizeof(Elem));
```

- LISTA == NULL : NÃO ALOCADA.
- *LISTA == NULL : VAZIA.

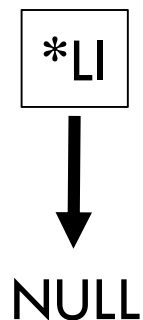


Lista Dinâmica: Inserção Ordenada

61

AL: 32

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    → Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```

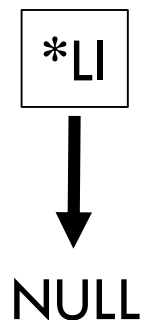


Lista Dinâmica: Inserção Ordenada

62

AL: 32

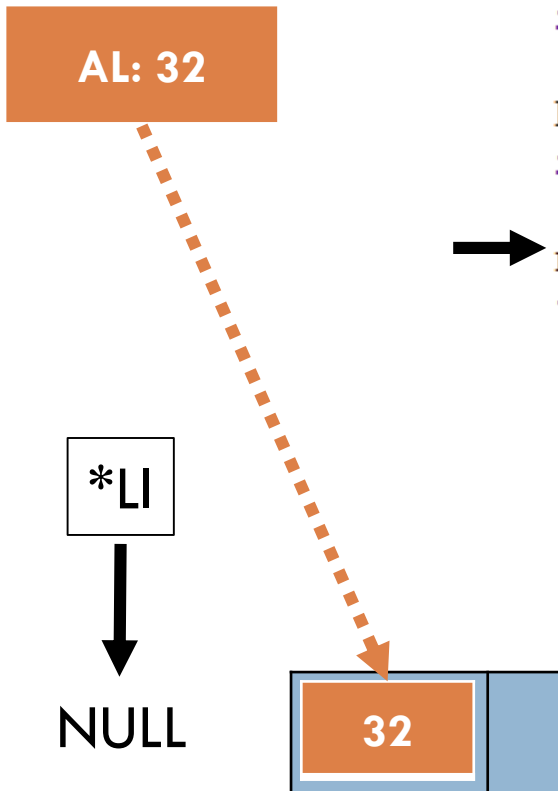
```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    → if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```



Lista Dinâmica: Inserção Ordenada

63

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    → no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```



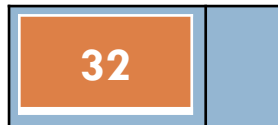
Lista Dinâmica: Inserção Ordenada

64

AL: 32

```
no->dados = al;  
→ if ((*li) == NULL) { //lista vazia: insere início  
    no->prox = NULL;  
    *li = no;  
    return 1;  
}  
else {
```

*LI
↓
NULL

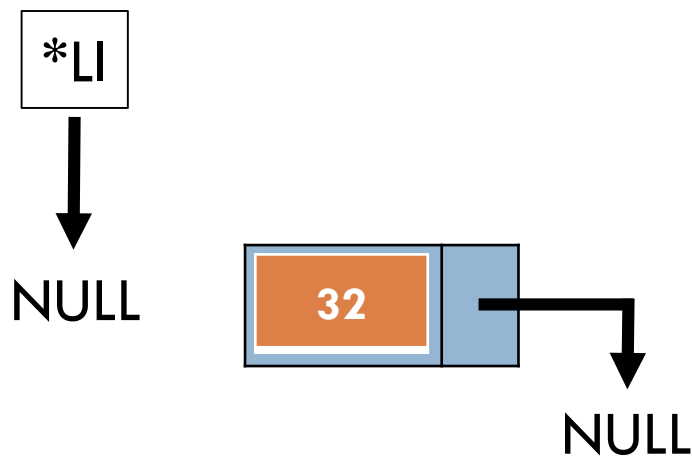


Lista Dinâmica: Inserção Ordenada

65

AL: 32

```
no->dados = al;  
if((*li) == NULL){//lista vazia: insere início  
    no->prox = NULL;  
    *li = no;  
    return 1;  
}  
else{
```

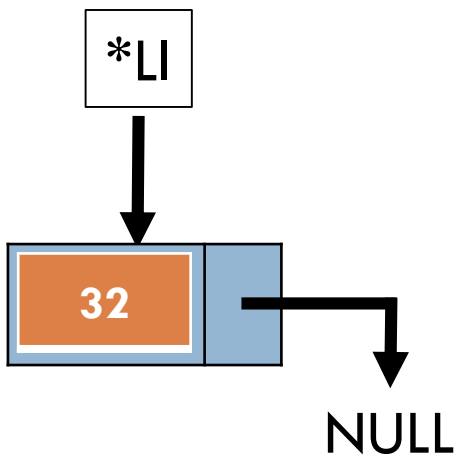


Lista Dinâmica: Inserção Ordenada

66

AL: 32

```
no->dados = al;  
if ((*li) == NULL) { // lista vazia: insere início  
    no->prox = NULL;  
    → *li = no;  
    return 1;  
}  
else {
```



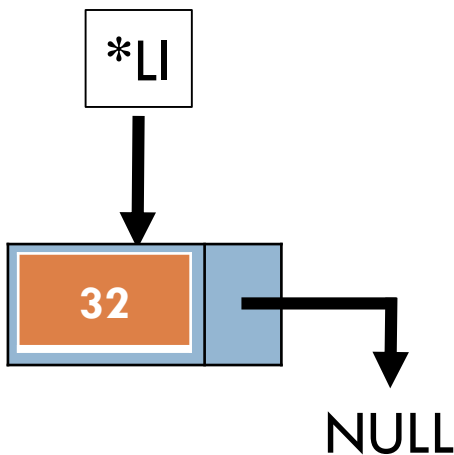
Lista Dinâmica: Inserção Ordenada

67

AL: 32

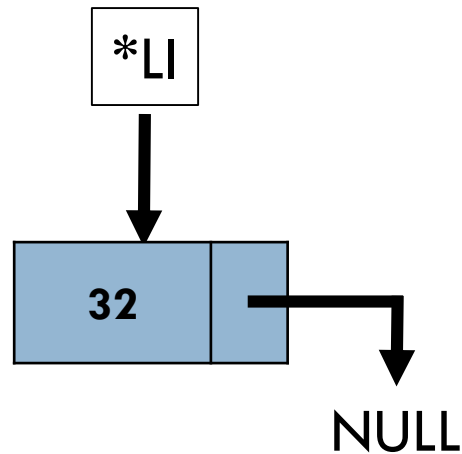
```
no->dados = al;  
if ((*li) == NULL) { // lista vazia: insere início  
    no->prox = NULL;  
    *li = no;  
    → return 1;  
}  
else {
```

- RETORNA OK;
- FINAL DA INSERÇÃO DO ELEMENTO.



Lista Dinâmica: Inserção Ordenada

68

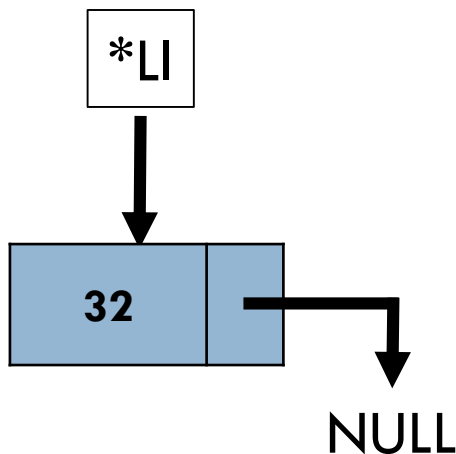


Lista Dinâmica: Inserção Ordenada

69

AL: 45

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```

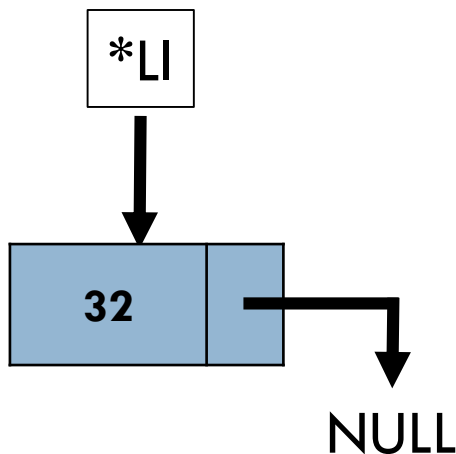


Lista Dinâmica: Inserção Ordenada

70

AL: 45

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    → if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```

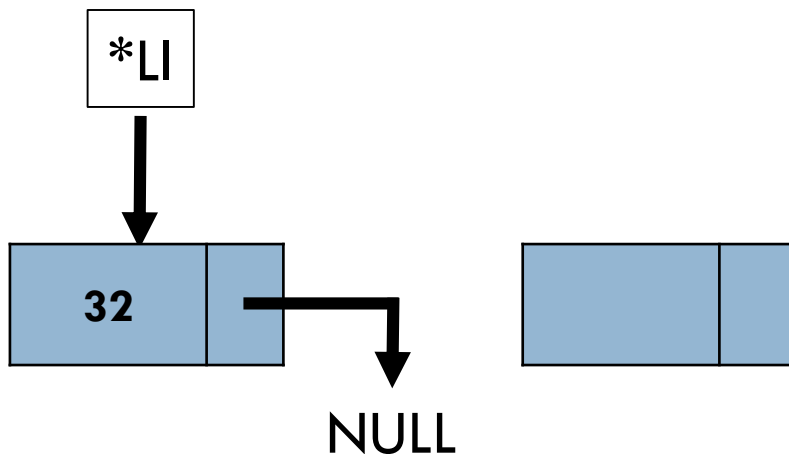


Lista Dinâmica: Inserção Ordenada

71

AL: 45

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    → Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```

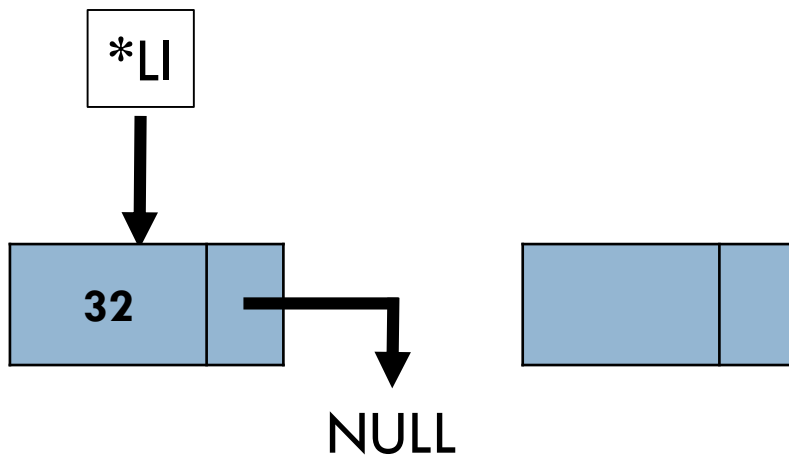


Lista Dinâmica: Inserção Ordenada

72

AL: 45

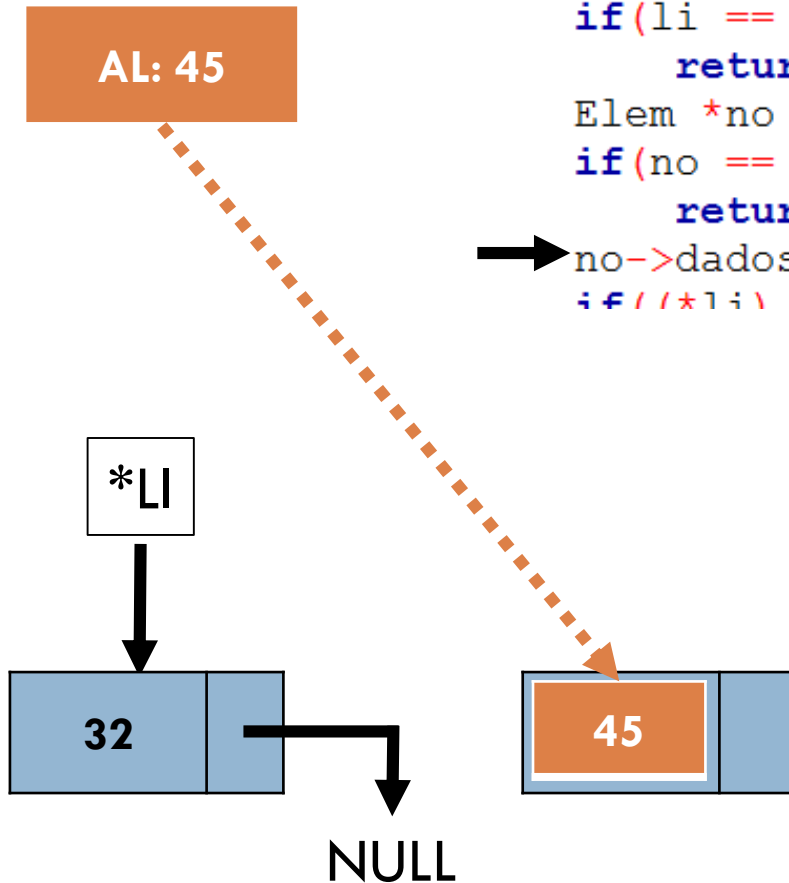
```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    → if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```



Lista Dinâmica: Inserção Ordenada

73

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    → no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```

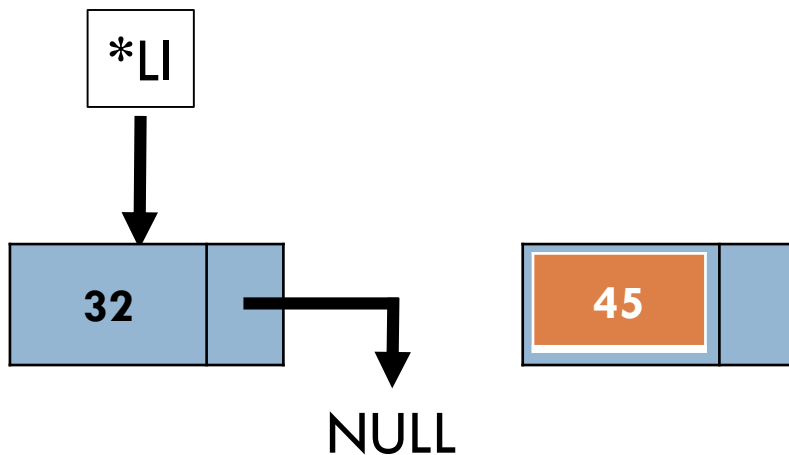


Lista Dinâmica: Inserção Ordenada

74

AL: 45

```
no->dados = al;  
→ if ((*li) == NULL) { //lista vazia: insere início  
    no->prox = NULL;  
    *li = no;  
    return 1;  
}  
else {
```

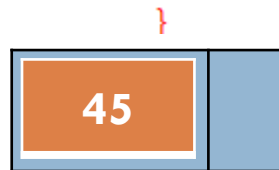
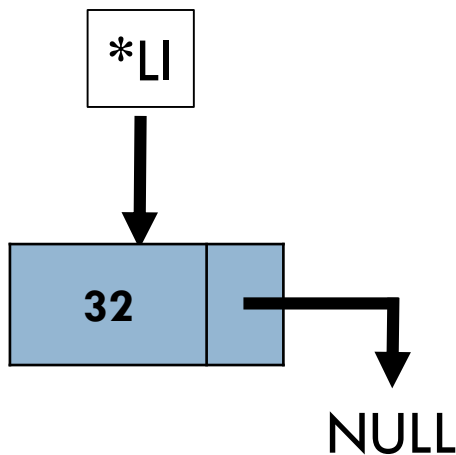


Lista Dinâmica: Inserção Ordenada

75

AL: 45

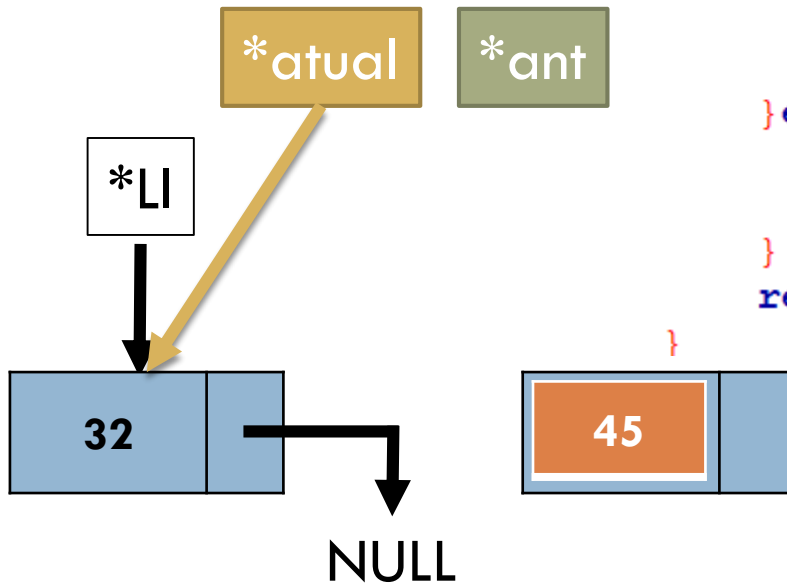
```
→ else{  
    Elem *ant, *atual = *li;  
    while(atual != NULL &&  
        atual->dados.matricula < al.matricula){  
  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){//insere início  
        no->prox = (*li);  
        *li = no;  
    }else{  
        no->prox = atual;  
        ant->prox = no;  
    }  
    return 1;  
}
```



Lista Dinâmica: Inserção Ordenada

76

AL: 45



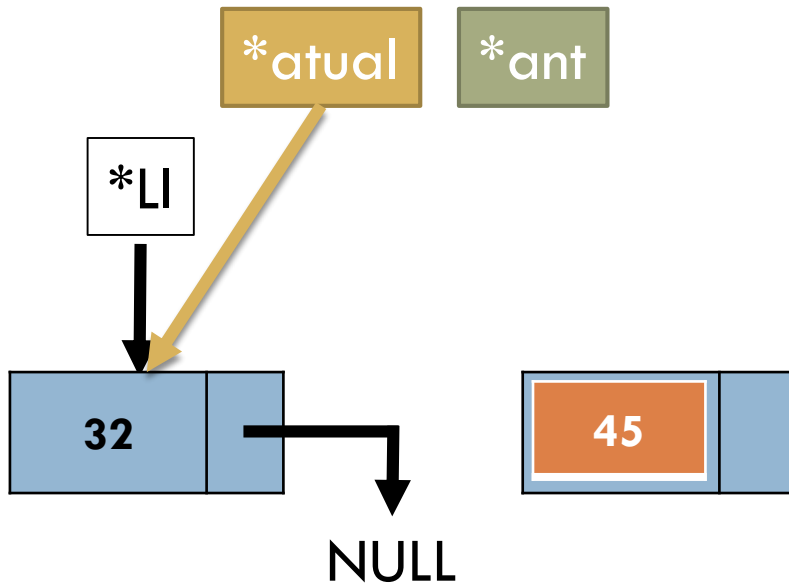
```
else{  
    → Elem *ant, *atual = *li;  
    while(atual != NULL &&  
        atual->dados.matricula < al.matricula){  
  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){//insere início  
        no->prox = (*li);  
        *li = no;  
    }else{  
        no->prox = atual;  
        ant->prox = no;  
    }  
    return 1;  
}
```

Lista Dinâmica: Inserção Ordenada

77

AL: 45

```
else{  
    Elem *ant, *atual = *li;  
    → while(atual != NULL &&  
           atual->dados.matricula < al.matricula){  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){ //insere início
```

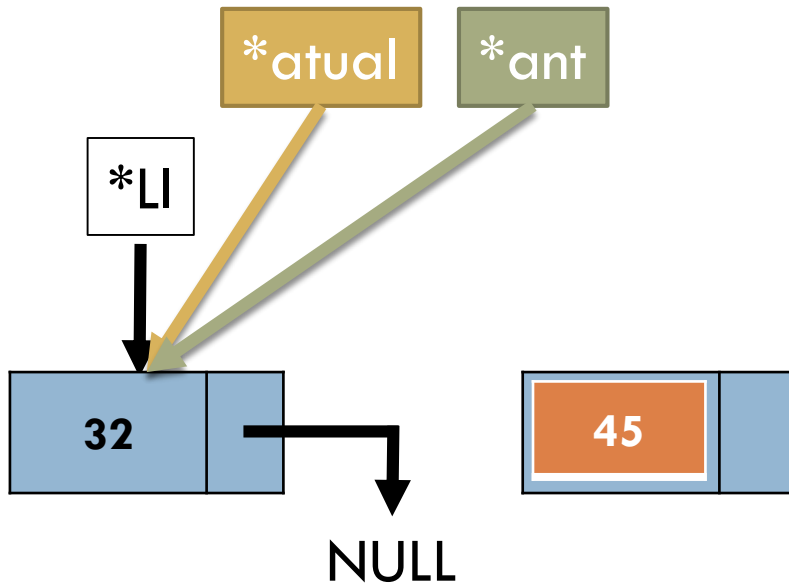


Lista Dinâmica: Inserção Ordenada

78

AL: 45

```
else{  
    Elem *ant, *atual = *li;  
    while(atual != NULL &&  
        atual->dados.matricula < al.matricula){  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){ //insere início
```

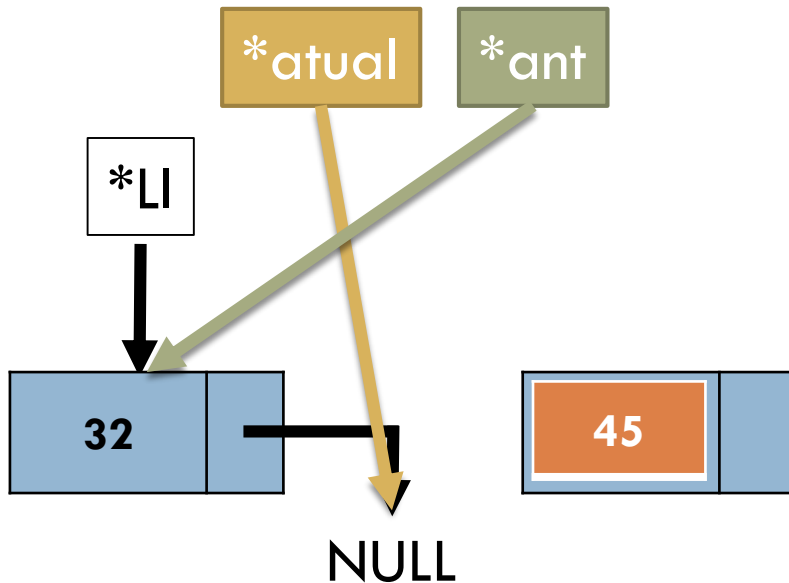


Lista Dinâmica: Inserção Ordenada

79

AL: 45

```
else{  
    Elem *ant, *atual = *li;  
    while(atual != NULL &&  
        atual->dados.matricula < al.matricula){  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){ //insere início
```

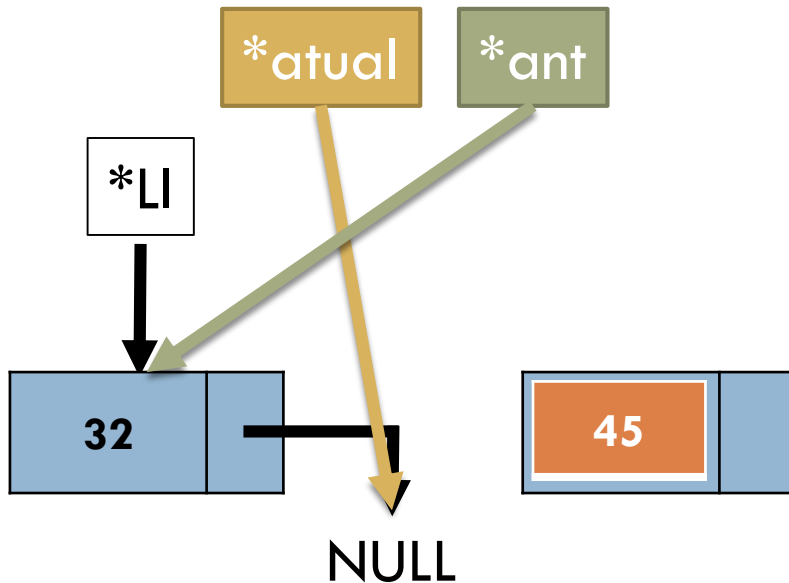


Lista Dinâmica: Inserção Ordenada

80

AL: 45

```
else{  
    Elem *ant, *atual = *li;  
    → while(atual != NULL &&  
           atual->dados.matricula < al.matricula){  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){ //insere início
```



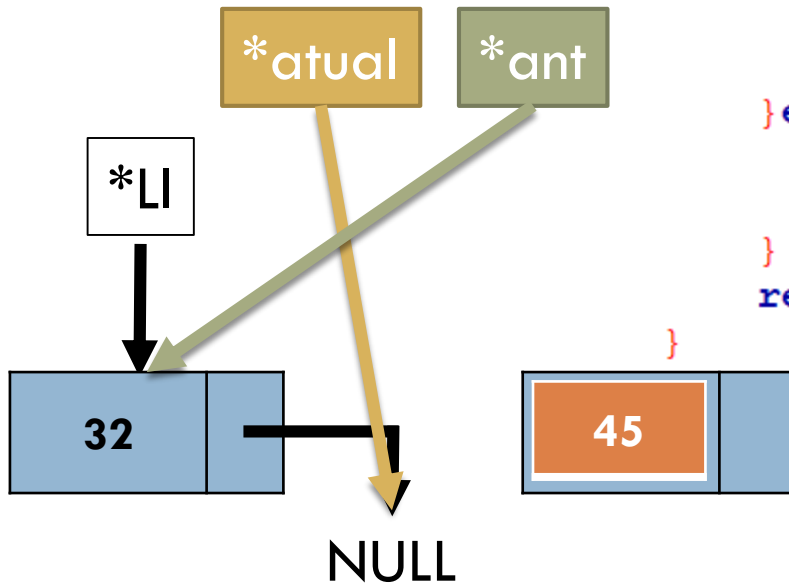
Lista Dinâmica: Inserção Ordenada

81

AL: 45

```
else{
    Elem *ant, *atual = *li;
    while(atual != NULL &&
        atual->dados.matricula < al.matricula){

        ant = atual;
        atual = atual->prox;
    }
    → if(atual == *li){//insere início
        no->prox = (*li);
        *li = no;
    }else{
        no->prox = atual;
        ant->prox = no;
    }
    return 1;
}
```



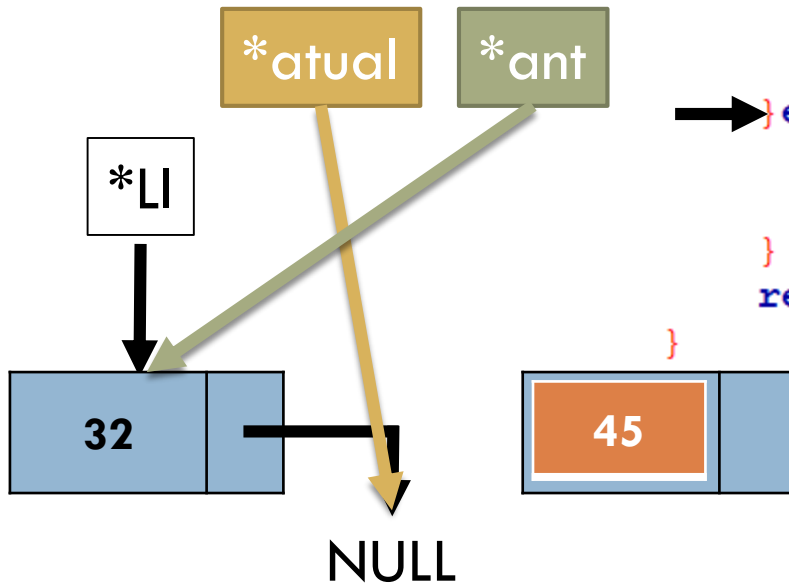
Lista Dinâmica: Inserção Ordenada

82

AL: 45

```
else{
    Elem *ant, *atual = *li;
    while(atual != NULL &&
        atual->dados.matricula < al.matricula){

        ant = atual;
        atual = atual->prox;
    }
    if(atual == *li){//insere início
        no->prox = (*li);
        *li = no;
    }else{
        no->prox = atual;
        ant->prox = no;
    }
    return 1;
}
```



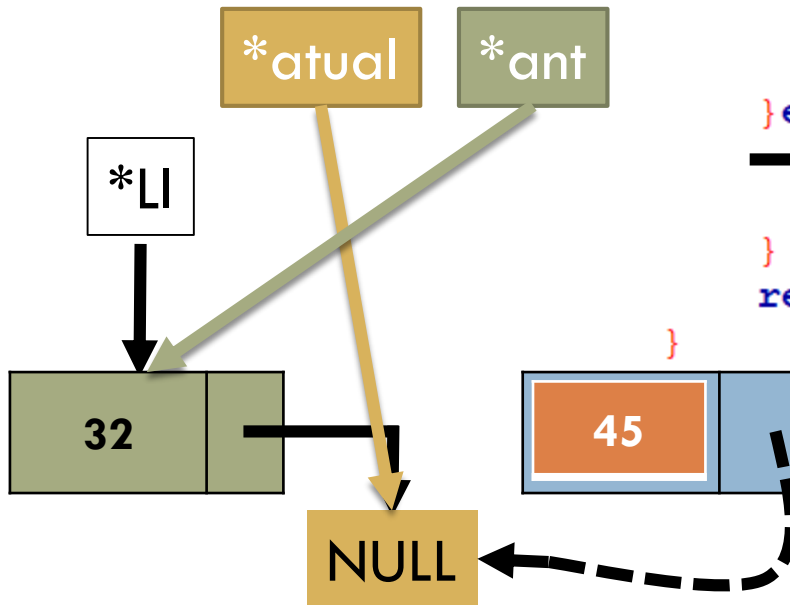
Lista Dinâmica: Inserção Ordenada

83

AL: 45

```
else{
    Elem *ant, *atual = *li;
    while(atual != NULL &&
        atual->dados.matricula < al.matricula){

        ant = atual;
        atual = atual->prox;
    }
    if(atual == *li){//insere início
        no->prox = (*li);
        *li = no;
    }else{
        → no->prox = atual;
        ant->prox = no;
    }
    return 1;
}
```



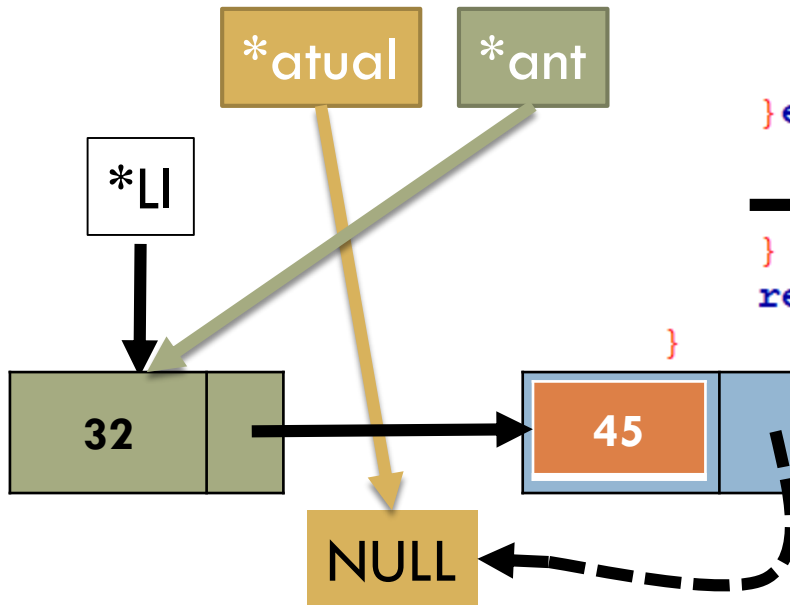
Lista Dinâmica: Inserção Ordenada

84

AL: 45

```
else{
    Elem *ant, *atual = *li;
    while(atual != NULL &&
        atual->dados.matricula < al.matricula){

        ant = atual;
        atual = atual->prox;
    }
    if(atual == *li){//insere início
        no->prox = (*li);
        *li = no;
    }else{
        no->prox = atual;
        ant->prox = no;
    }
    return 1;
}
```



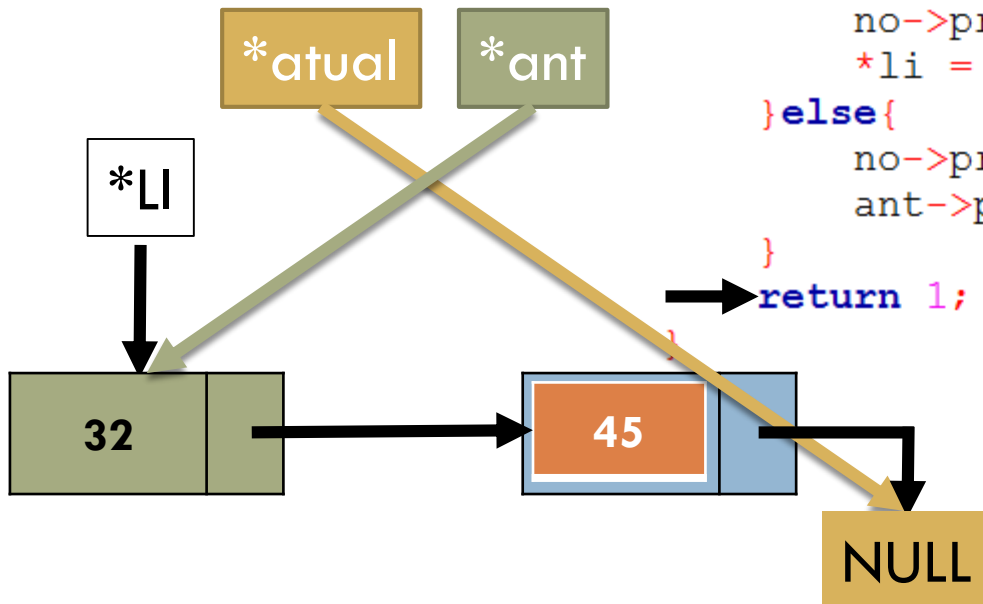
Lista Dinâmica: Inserção Ordenada

85

AL: 45

```
else{  
    Elem *ant = *atual - *li;  
    while
```

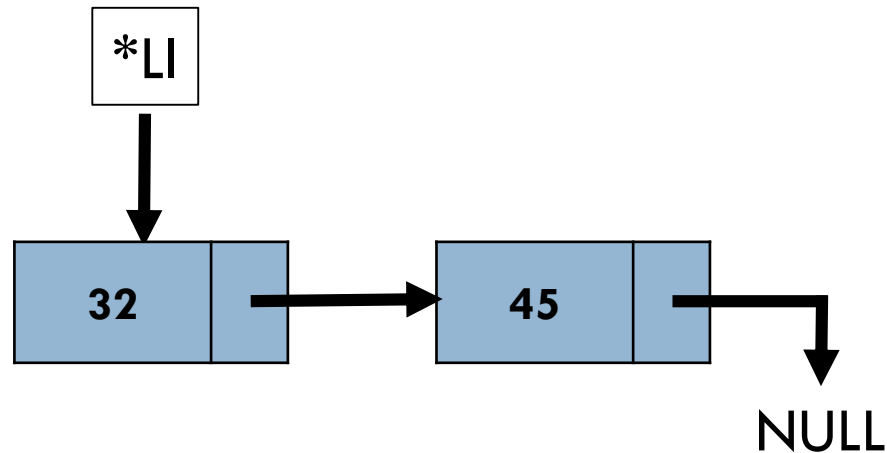
- RETORNA OK;
- FINAL DA INSERÇÃO DO ELEMENTO.



```
}  
if(atual == *li){//insere início  
    no->prox = (*li);  
    *li = no;  
}  
else{  
    no->prox = atual;  
    ant->prox = no;  
}  
return 1;
```

Lista Dinâmica: Inserção Ordenada

86

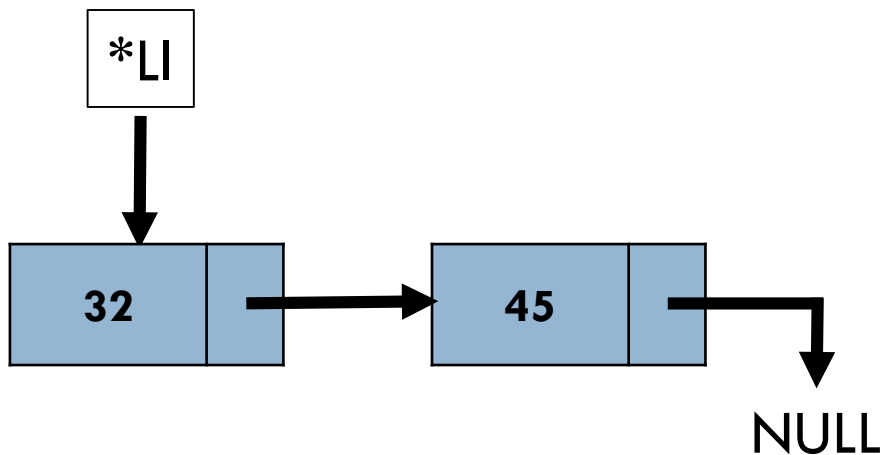


Lista Dinâmica: Inserção Ordenada

87

AL: 18

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```

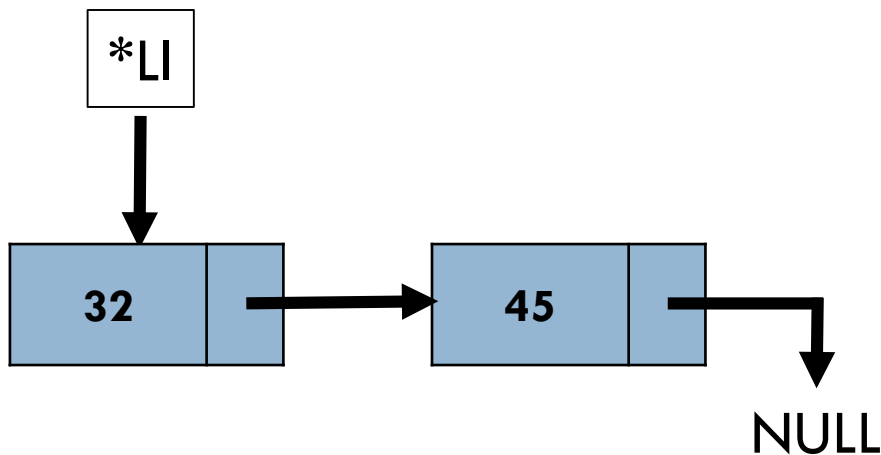


Lista Dinâmica: Inserção Ordenada

88

AL: 18

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    → if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```

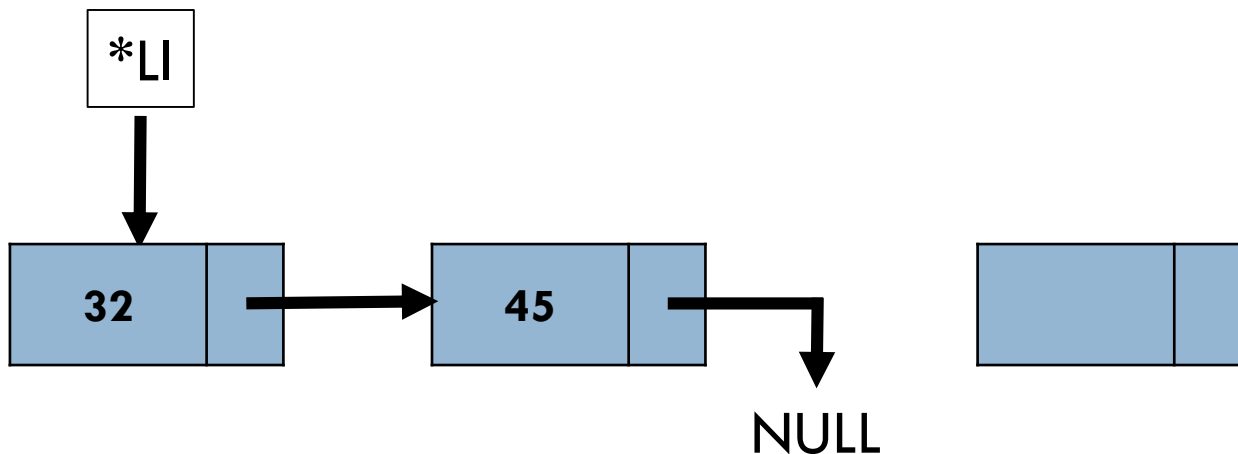


Lista Dinâmica: Inserção Ordenada

89

AL: 18

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    → Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```

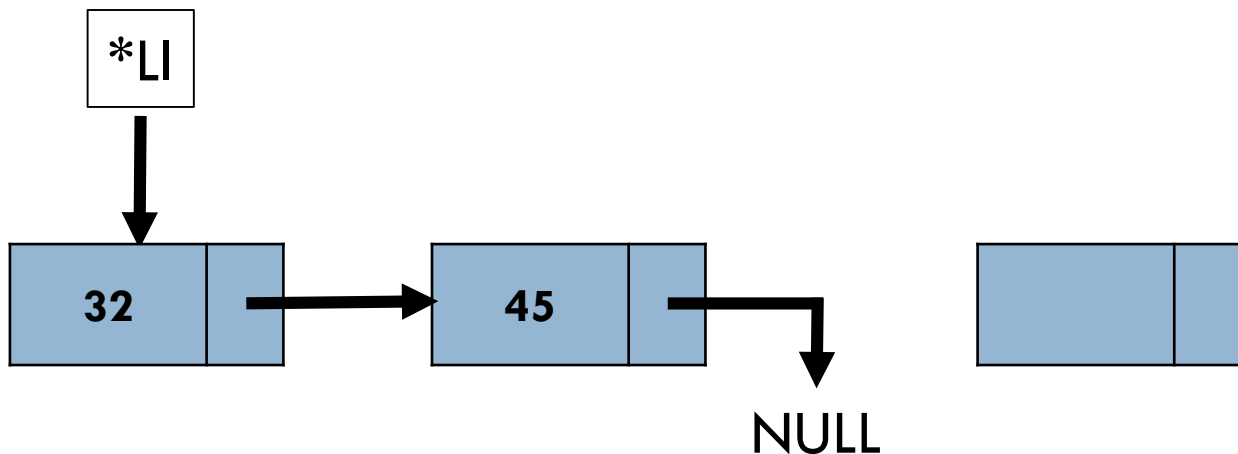


Lista Dinâmica: Inserção Ordenada

90

AL: 18

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    → if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```

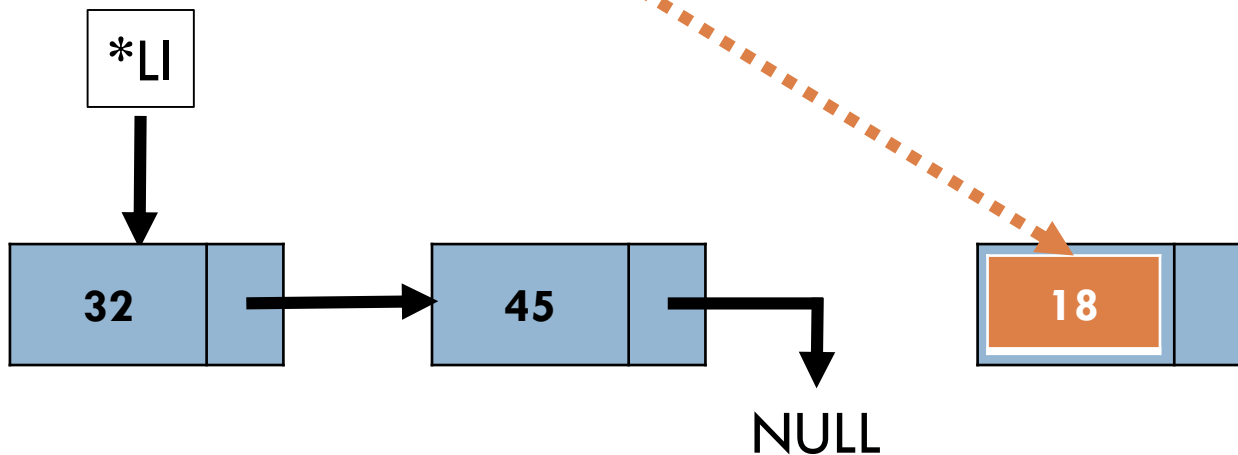


Lista Dinâmica: Inserção Ordenada

91

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if ((*li) == NULL) //lista vazia: insere início
```

AL: 18

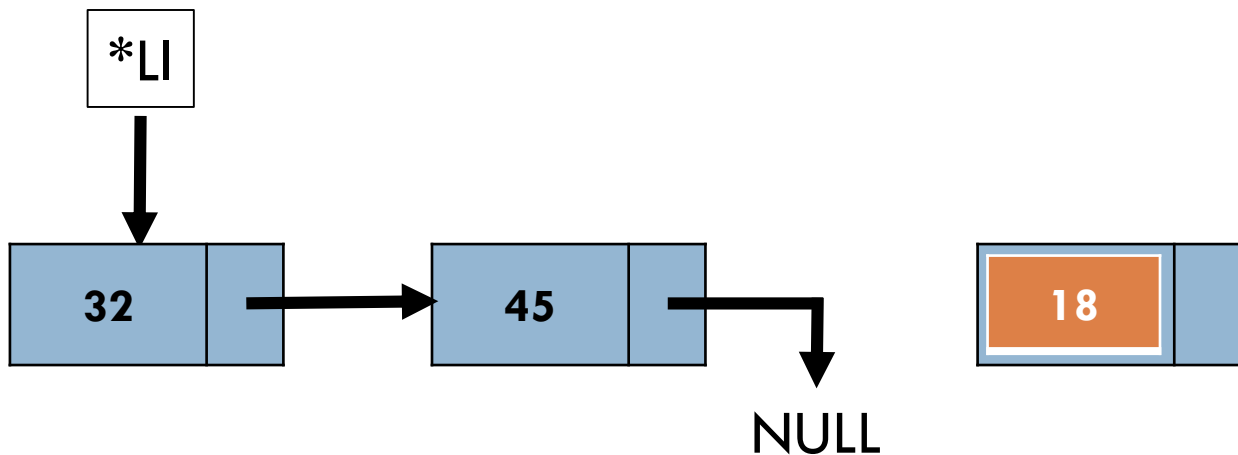


Lista Dinâmica: Inserção Ordenada

92

AL: 18

```
no->dados = al;  
→ if ((*li) == NULL) { //lista vazia: insere início  
    no->prox = NULL;  
    *li = no;  
    return 1;  
}  
else {
```

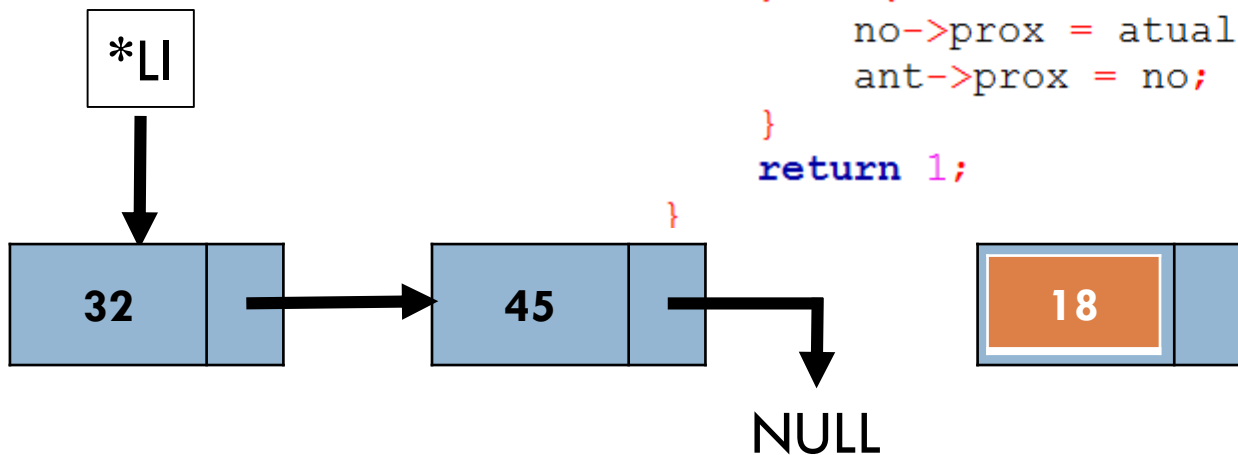


Lista Dinâmica: Inserção Ordenada

93

AL: 18

```
→ else{  
    Elem *ant, *atual = *li;  
    while(atual != NULL &&  
        atual->dados.matricula < al.matricula){  
  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){//insere início  
        no->prox = (*li);  
        *li = no;  
    }else{  
        no->prox = atual;  
        ant->prox = no;  
    }  
    return 1;  
}
```

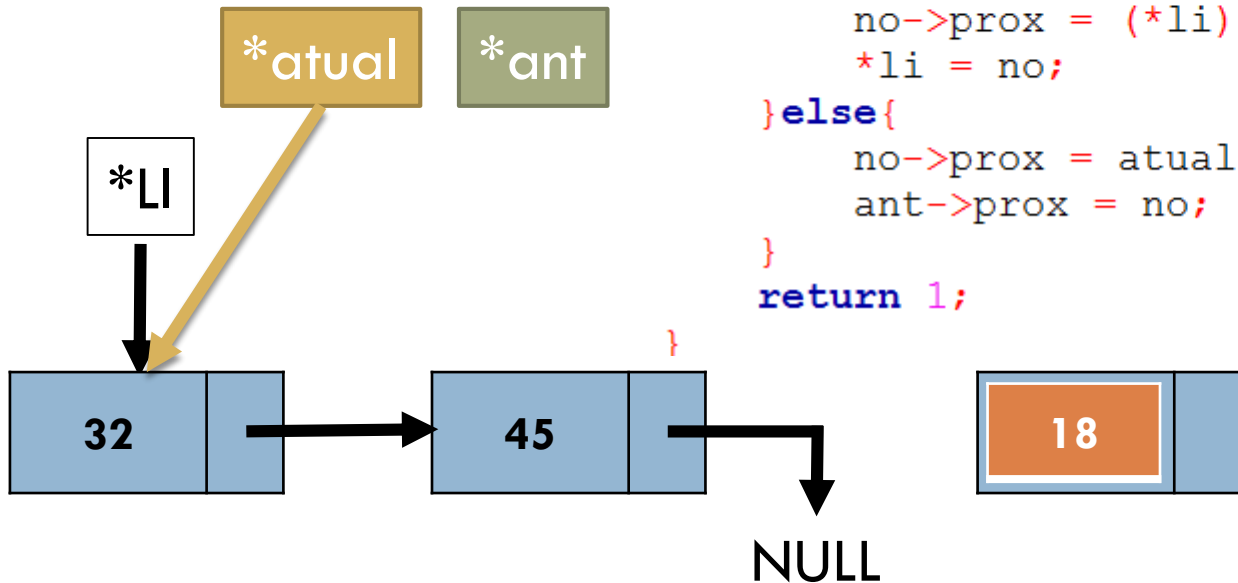


Lista Dinâmica: Inserção Ordenada

94

AL: 18

```
else{  
    → Elem *ant, *atual = *li;  
    while(atual != NULL &&  
        atual->dados.matricula < al.matricula){  
  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){//insere início  
        no->prox = (*li);  
        *li = no;  
    }else{  
        no->prox = atual;  
        ant->prox = no;  
    }  
    return 1;  
}
```

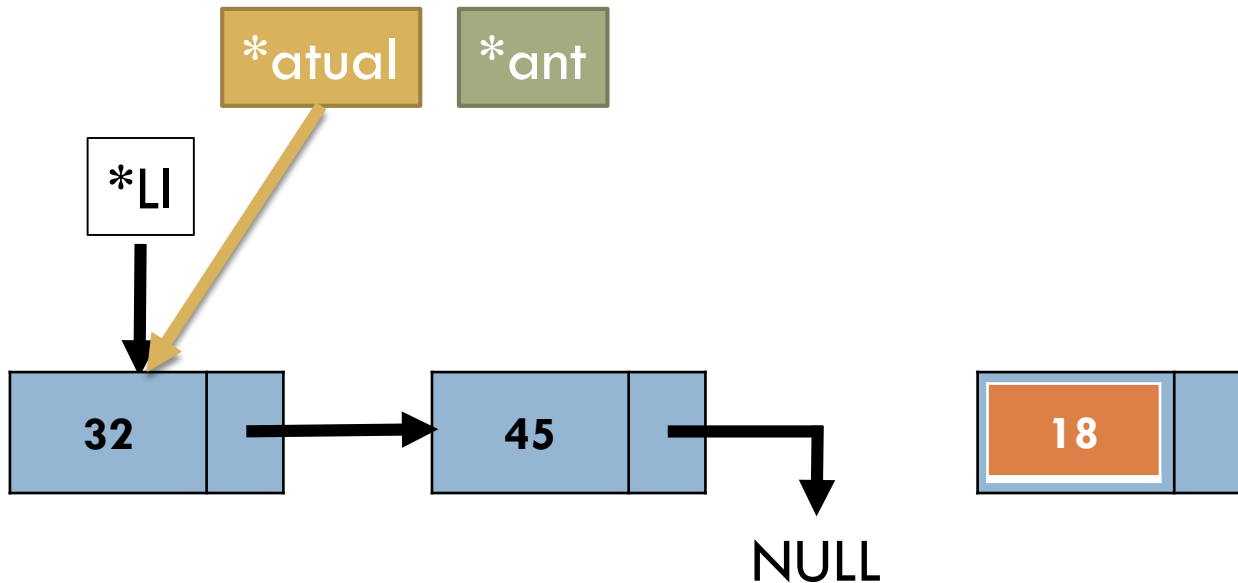


Lista Dinâmica: Inserção Ordenada

95

AL: 18

```
else{  
    Elem *ant, *atual = *li;  
    → while(atual != NULL &&  
           atual->dados.matricula < al.matricula){  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){ //insere início
```



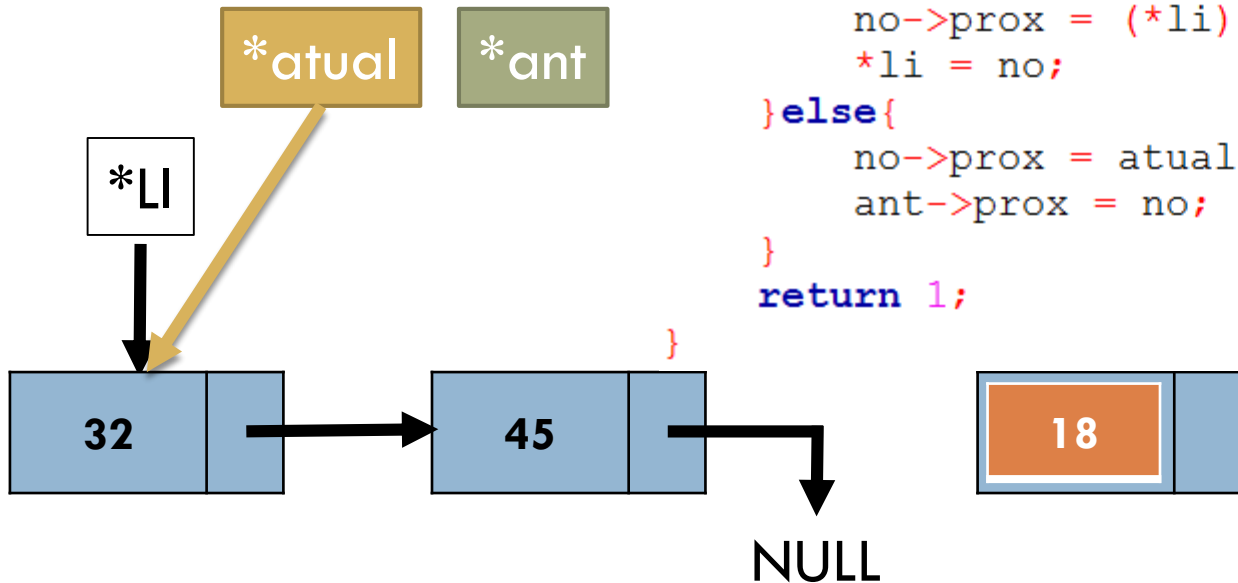
Lista Dinâmica: Inserção Ordenada

96

AL: 18

```
else{
    Elem *ant, *atual = *li;
    while(atual != NULL &&
        atual->dados.matricula < al.matricula){

        ant = atual;
        atual = atual->prox;
    }
    → if(atual == *li){//insere início
        no->prox = (*li);
        *li = no;
    }else{
        no->prox = atual;
        ant->prox = no;
    }
    return 1;
}
```



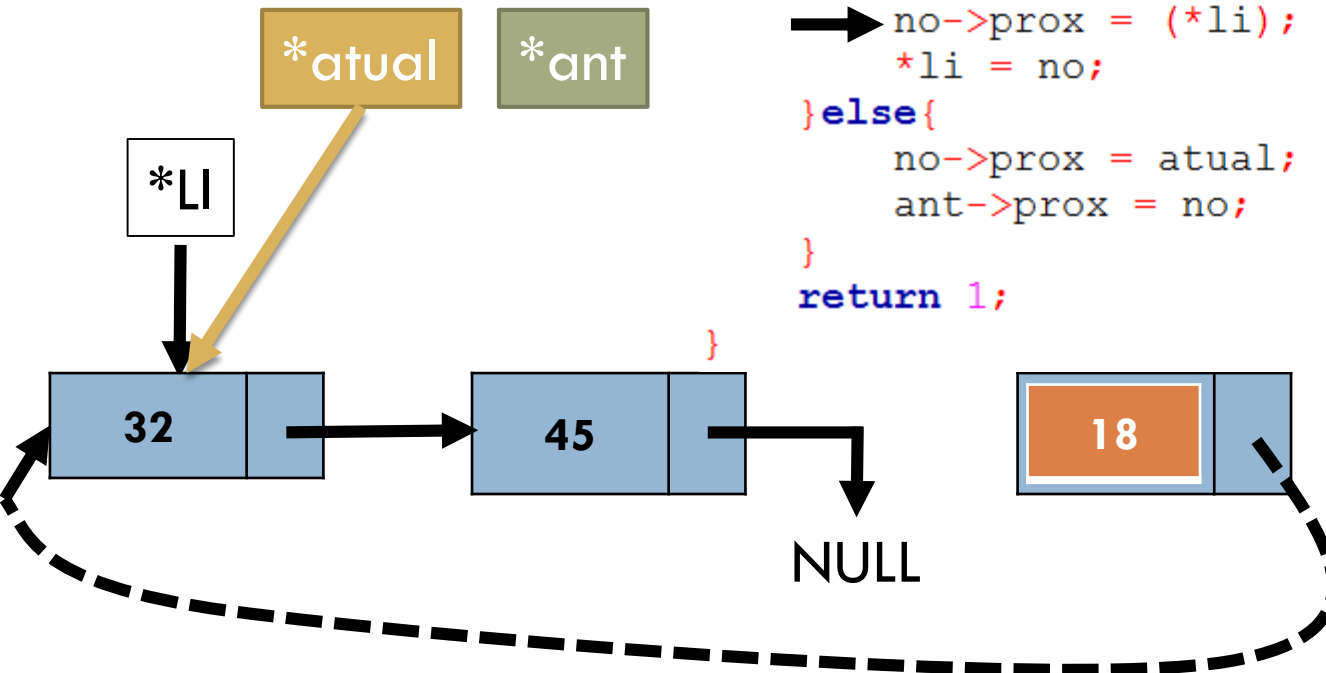
Lista Dinâmica: Inserção Ordenada

97

AL: 18

```
else{
    Elem *ant, *atual = *li;
    while(atual != NULL &&
        atual->dados.matricula < al.matricula){

        ant = atual;
        atual = atual->prox;
    }
    if(atual == *li){//insere início
        → no->prox = (*li);
        *li = no;
    }else{
        no->prox = atual;
        ant->prox = no;
    }
    return 1;
}
```



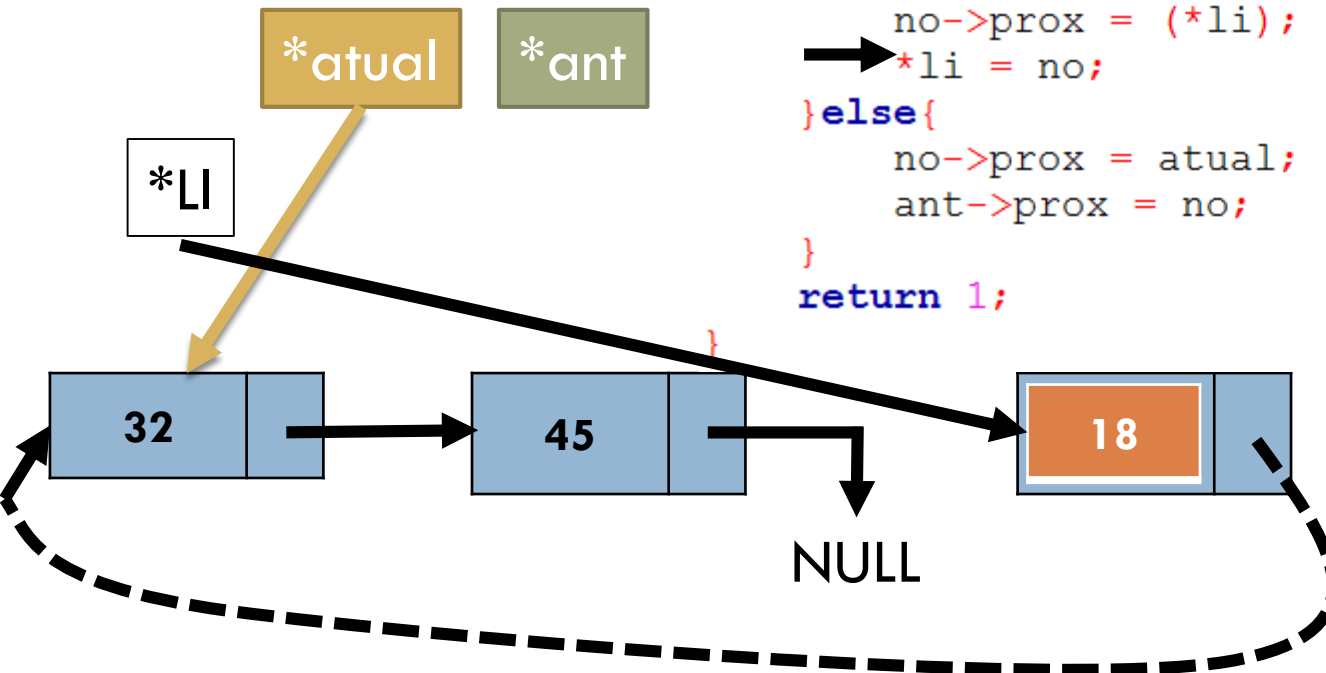
Lista Dinâmica: Inserção Ordenada

98

AL: 18

```
else{
    Elem *ant, *atual = *li;
    while(atual != NULL &&
        atual->dados.matricula < al.matricula){

        ant = atual;
        atual = atual->prox;
    }
    if(atual == *li){//insere início
        no->prox = (*li);
        → *li = no;
    }else{
        no->prox = atual;
        ant->prox = no;
    }
    return 1;
}
```



Lista Dinâmica: Inserção Ordenada

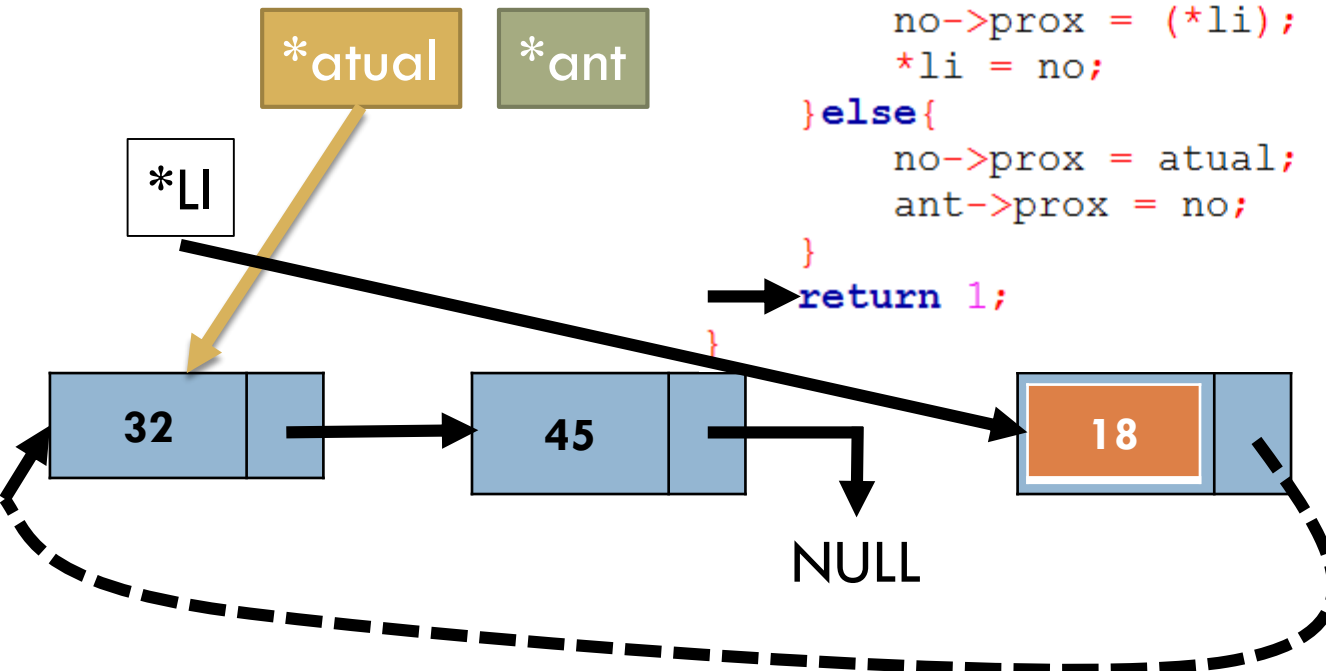
99

AL: 18

```
else{  
    Elem *ant, *atual = *li;  
    while
```

- RETORNA OK;
- FINAL DA INSERÇÃO DO ELEMENTO.

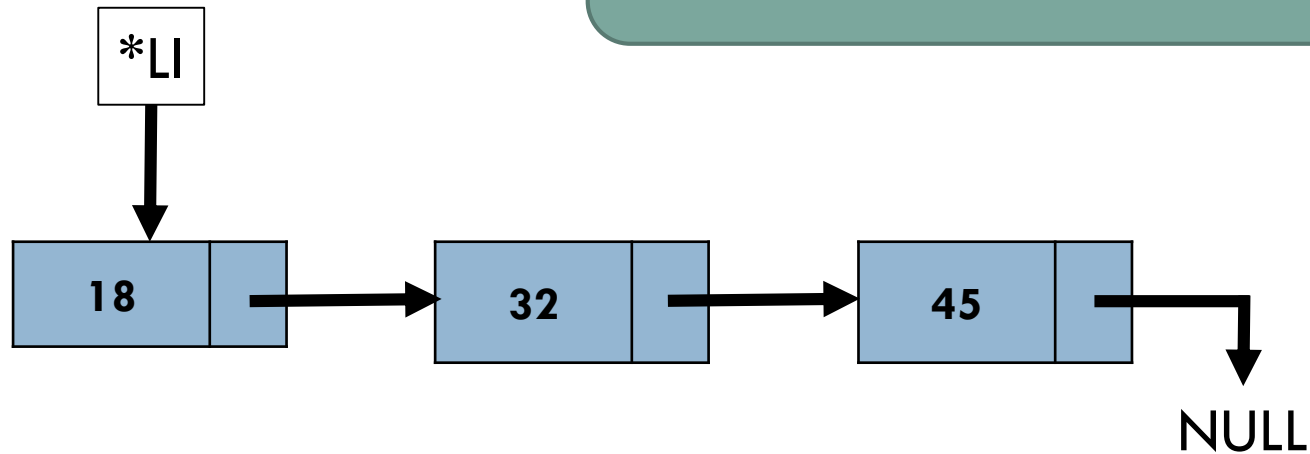
```
}  
if(atual == *li){//insere início  
    no->prox = (*li);  
    *li = no;  
}  
else{  
    no->prox = atual;  
    ant->prox = no;  
}  
return 1;  
}
```



Lista Dinâmica: Inserção Ordenada

100

REORGANIZANDO A IMAGEM

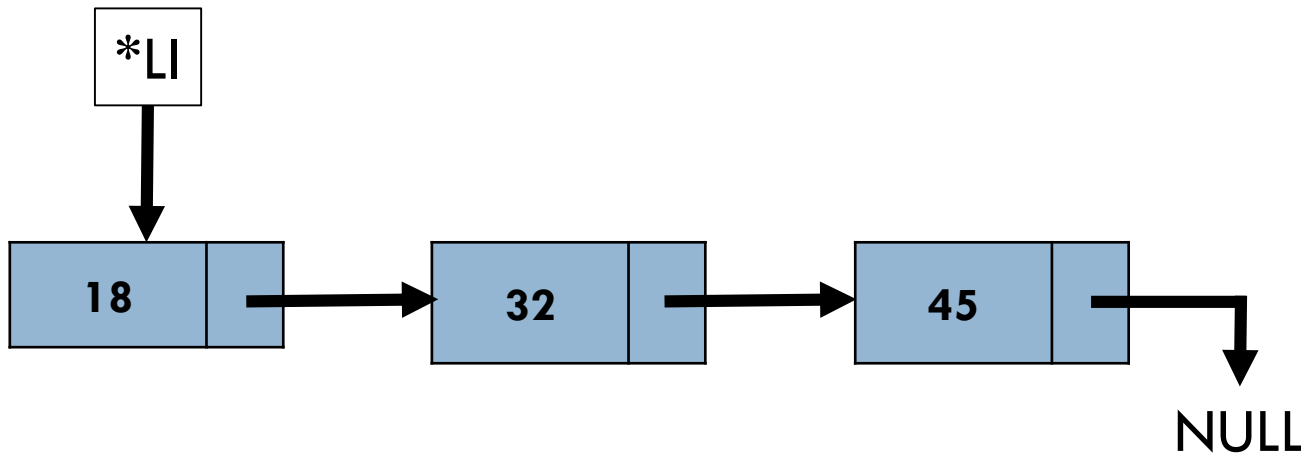


Lista Dinâmica: Inserção Ordenada

101

AL: 25

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    → if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```

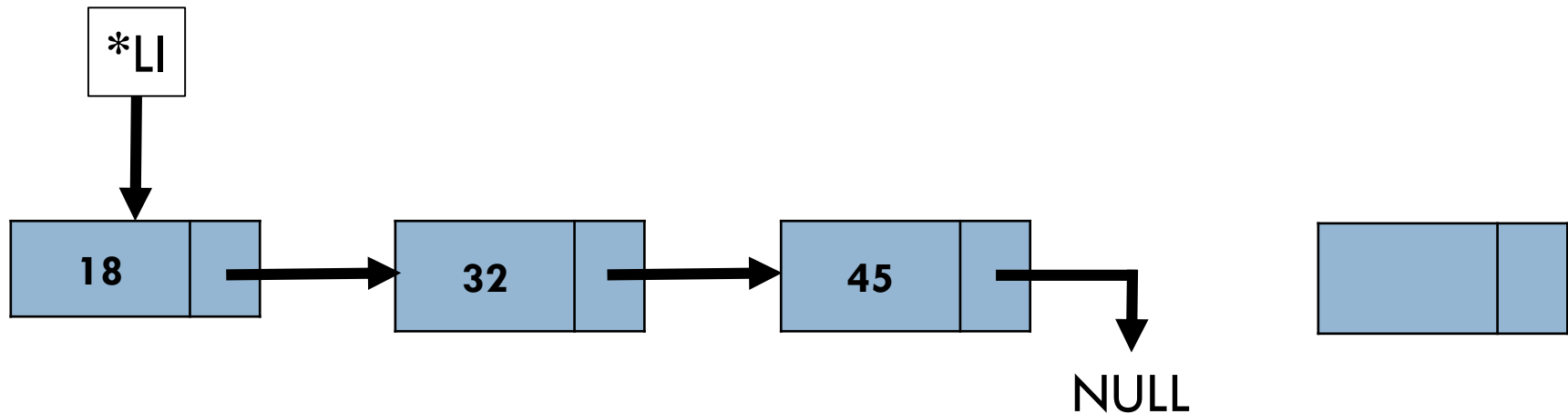


Lista Dinâmica: Inserção Ordenada

102

AL: 25

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    → Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```

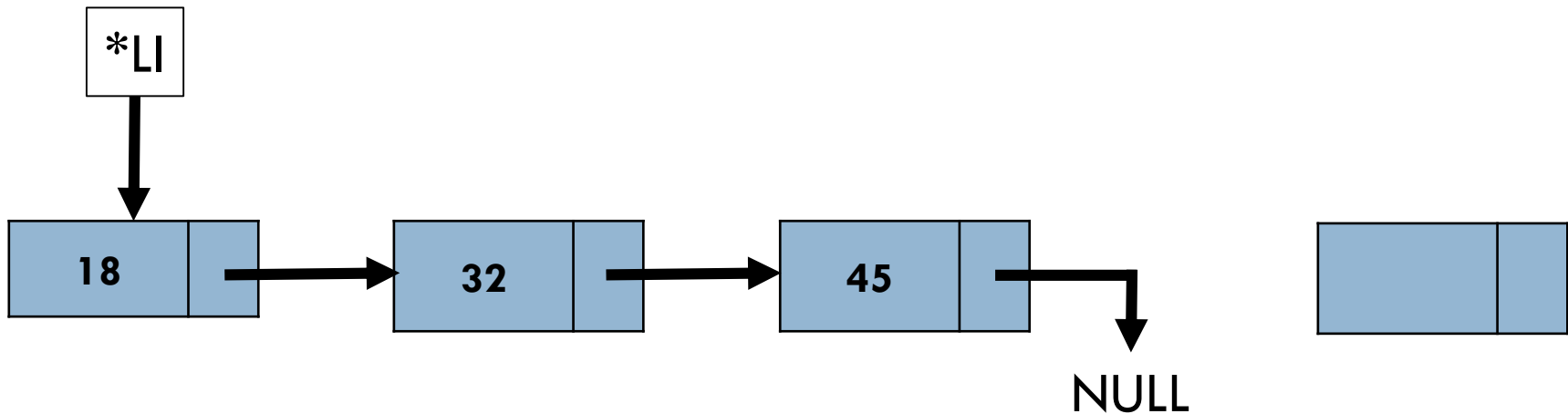


Lista Dinâmica: Inserção Ordenada

103

AL: 25

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    → if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```

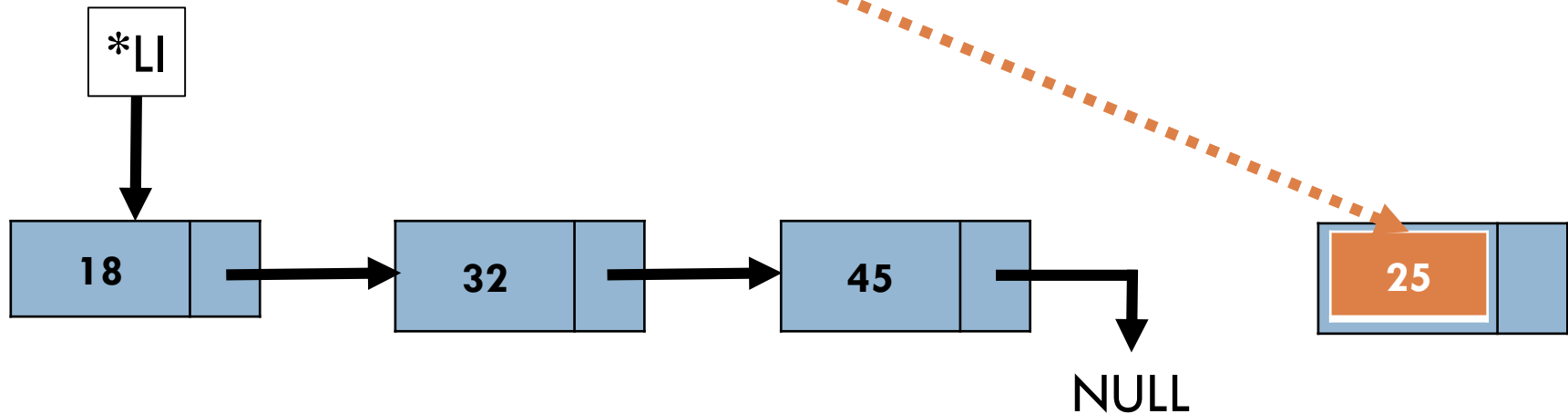


Lista Dinâmica: Inserção Ordenada

104

```
//Arquivo ListaDinamica.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    if(*li == NULL) //lista vazia: insere início
```

AL: 25

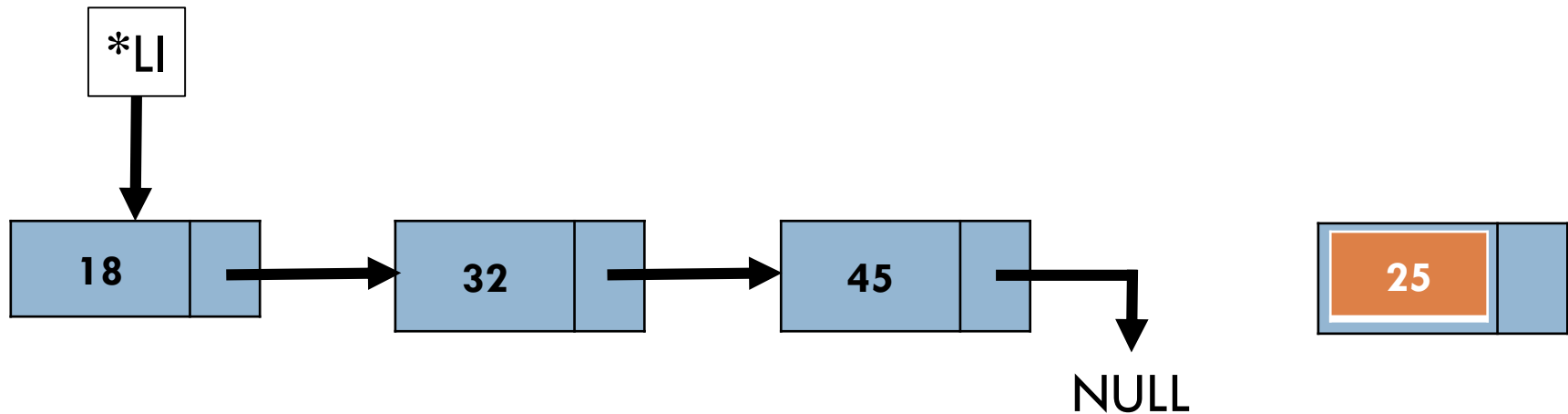


Lista Dinâmica: Inserção Ordenada

105

AL: 25

```
no->dados = al;  
→ if ((*li) == NULL) { //lista vazia: insere início  
    no->prox = NULL;  
    *li = no;  
    return 1;  
}  
else {
```

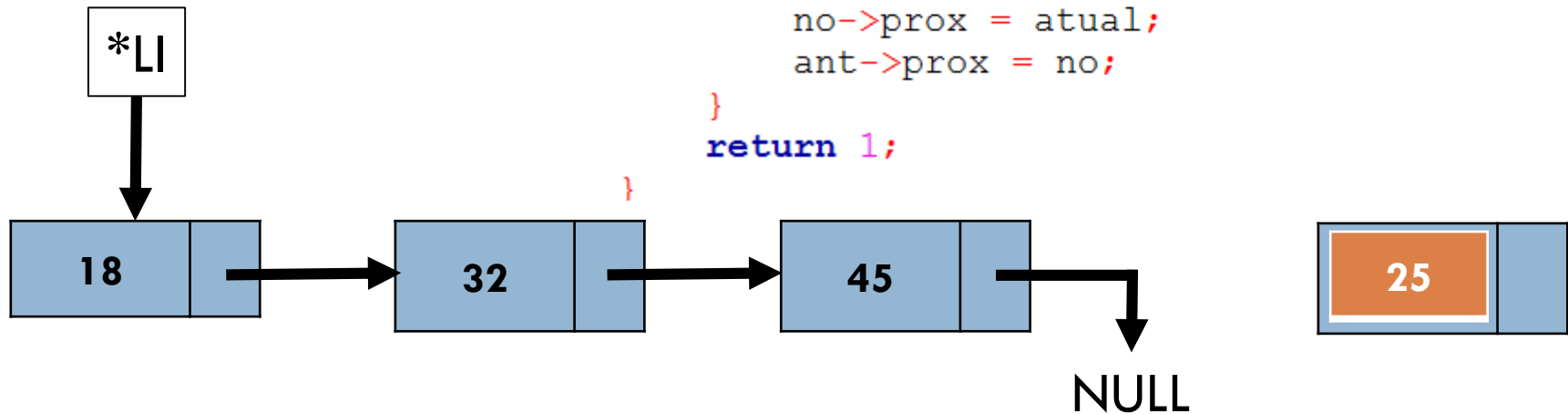


Lista Dinâmica: Inserção Ordenada

106

AL: 25

```
→ else{  
    Elem *ant, *atual = *li;  
    while(atual != NULL &&  
        atual->dados.matricula < al.matricula){  
  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){//insere início  
        no->prox = (*li);  
        *li = no;  
    }else{  
        no->prox = atual;  
        ant->prox = no;  
    }  
    return 1;  
}
```

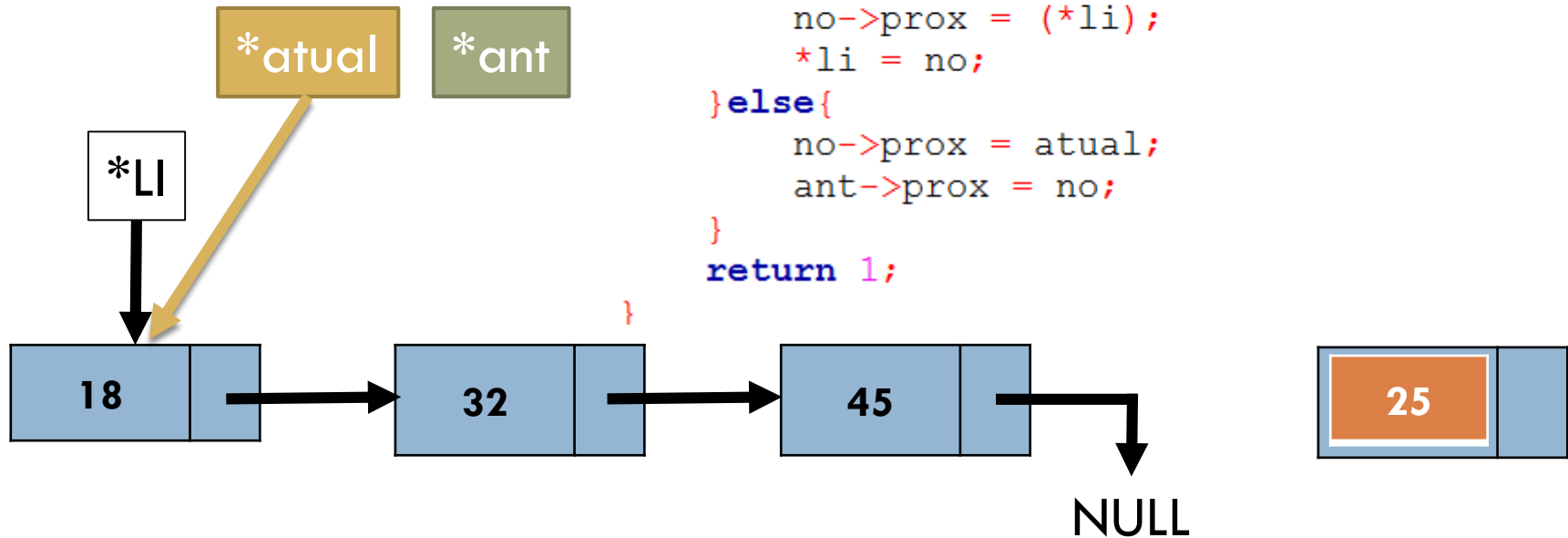


Lista Dinâmica: Inserção Ordenada

107

AL: 25

```
else{  
    → Elem *ant, *atual = *li;  
    while(atual != NULL &&  
        atual->dados.matricula < al.matricula){  
  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){//insere início  
        no->prox = (*li);  
        *li = no;  
    }else{  
        no->prox = atual;  
        ant->prox = no;  
    }  
    return 1;  
}
```

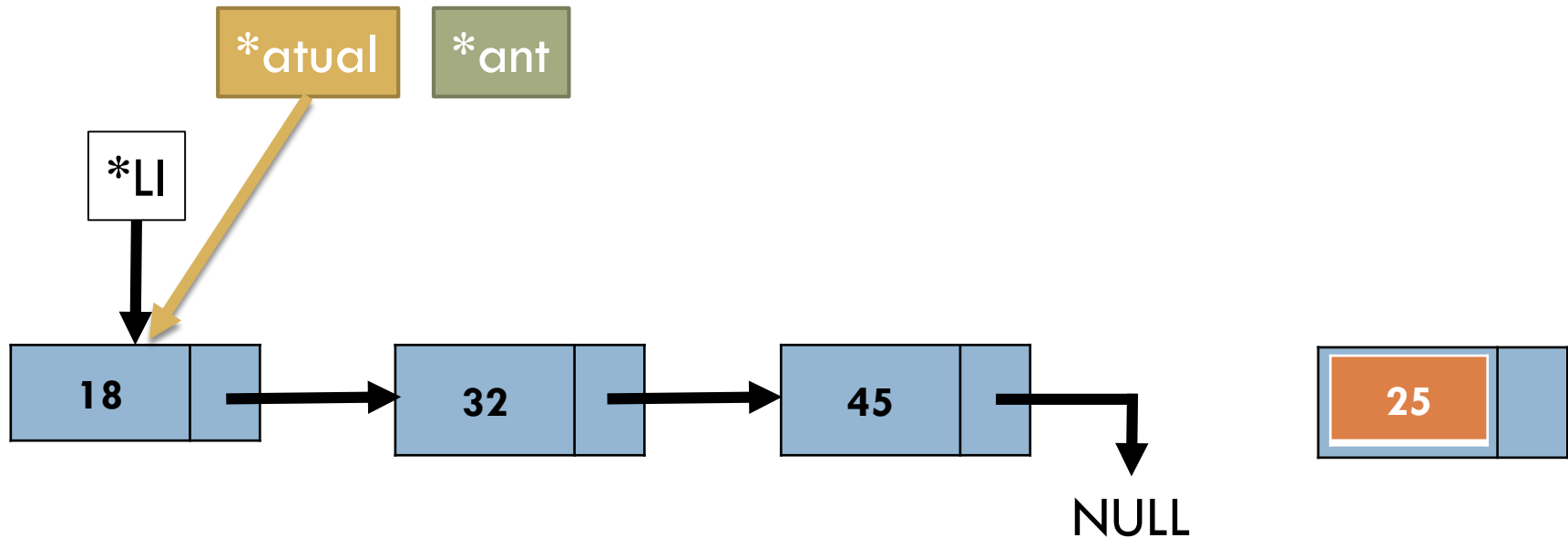


Lista Dinâmica: Inserção Ordenada

108

AL: 25

```
else{  
    Elem *ant, *atual = *li;  
    → while(atual != NULL &&  
           atual->dados.matricula < al.matricula){  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){ //insere início
```

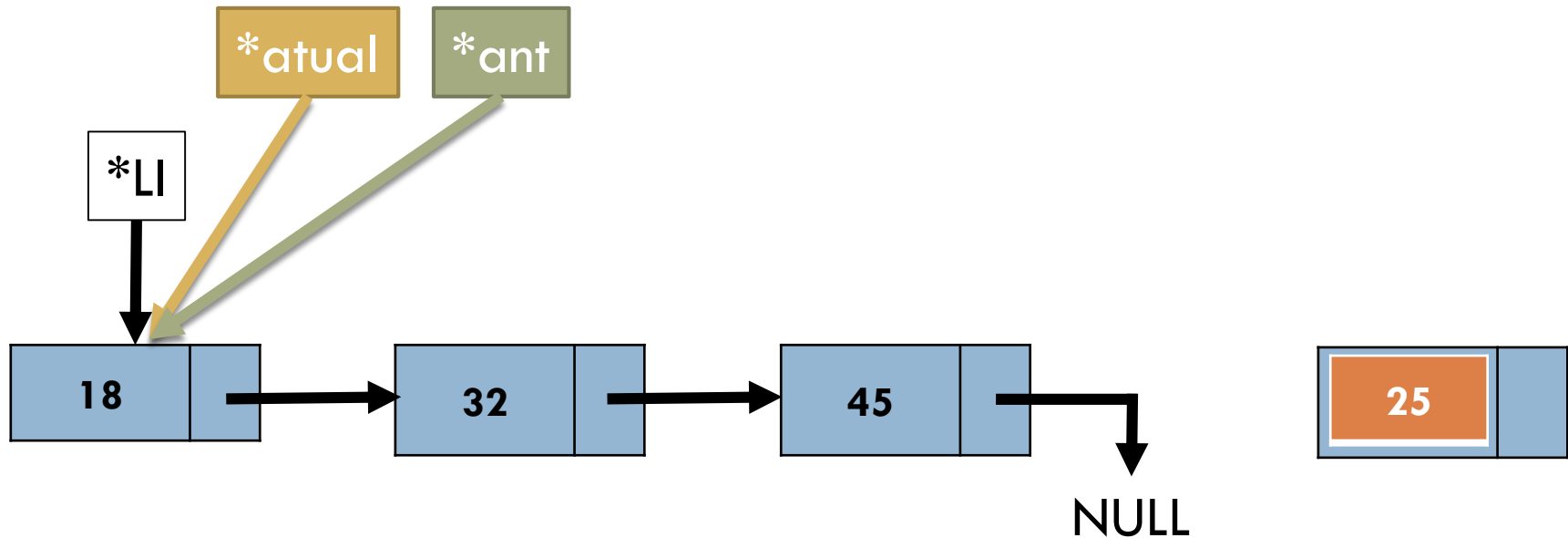


Lista Dinâmica: Inserção Ordenada

109

AL: 25

```
else{  
    Elem *ant, *atual = *li;  
    while(atual != NULL &&  
        atual->dados.matricula < al.matricula){  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){ //insere início
```

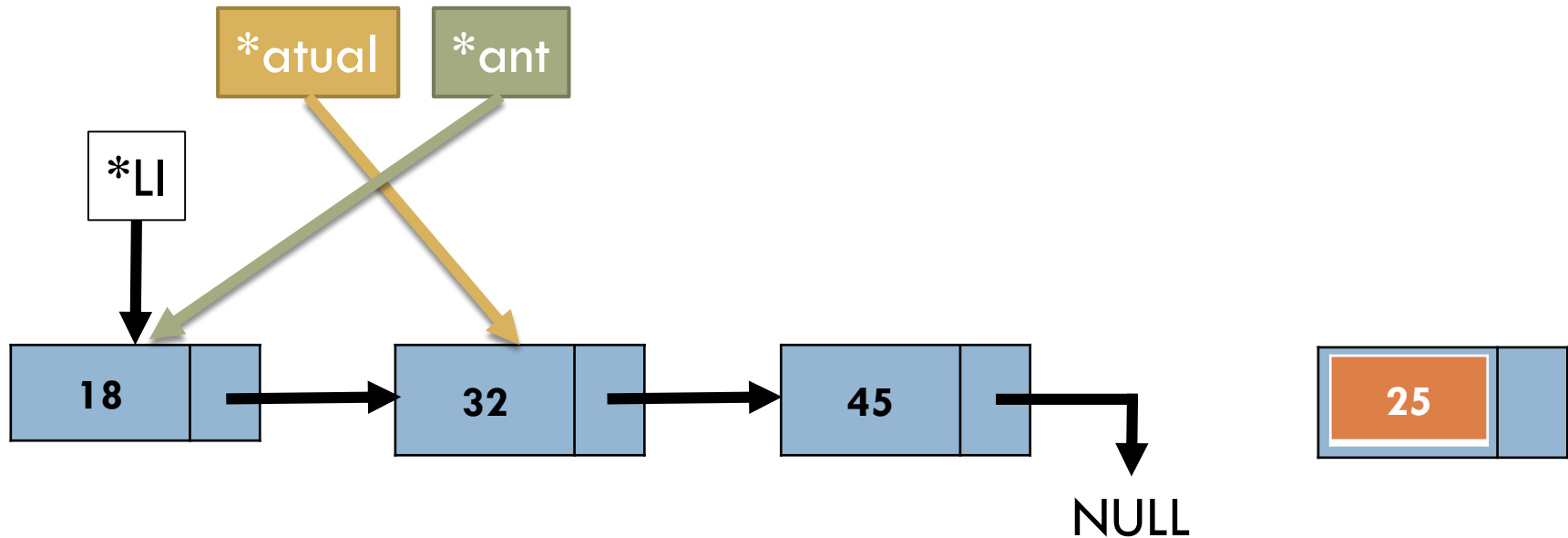


Lista Dinâmica: Inserção Ordenada

110

AL: 25

```
else{  
    Elem *ant, *atual = *li;  
    while(atual != NULL &&  
        atual->dados.matricula < al.matricula){  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){ //insere início
```

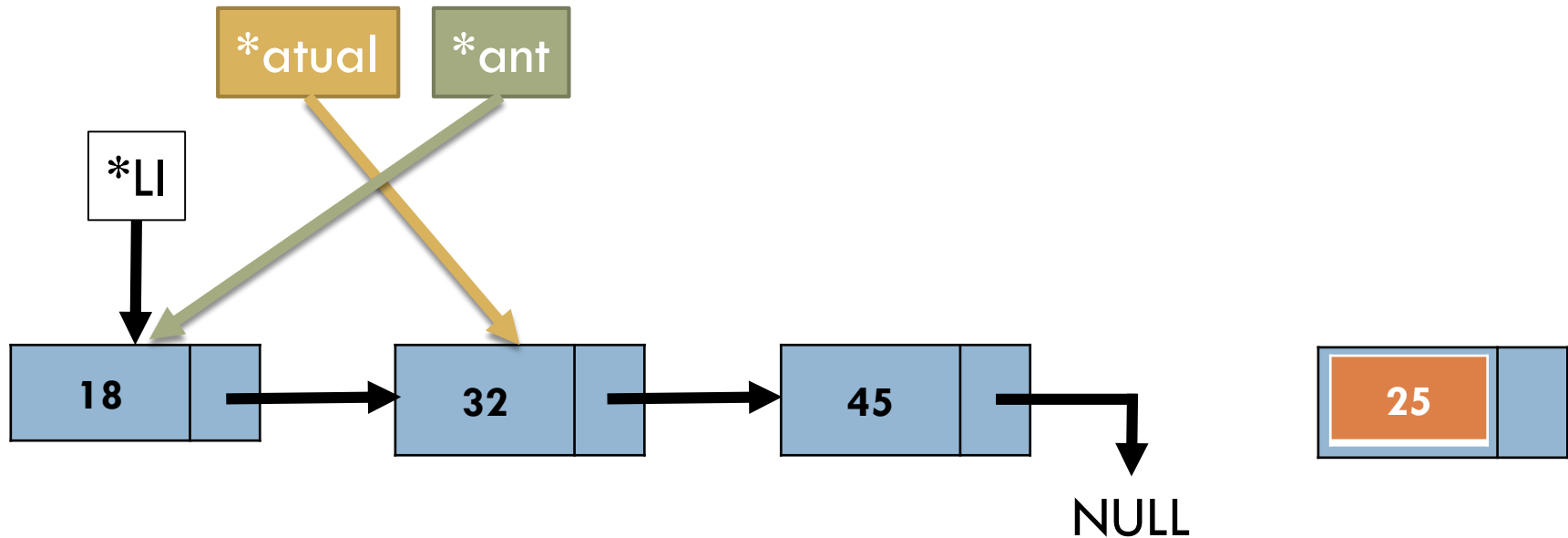


Lista Dinâmica: Inserção Ordenada

111

AL: 25

```
else{  
    Elem *ant, *atual = *li;  
    → while(atual != NULL &&  
           atual->dados.matricula < al.matricula){  
        ant = atual;  
        atual = atual->prox;  
    }  
    if(atual == *li){ //insere início
```



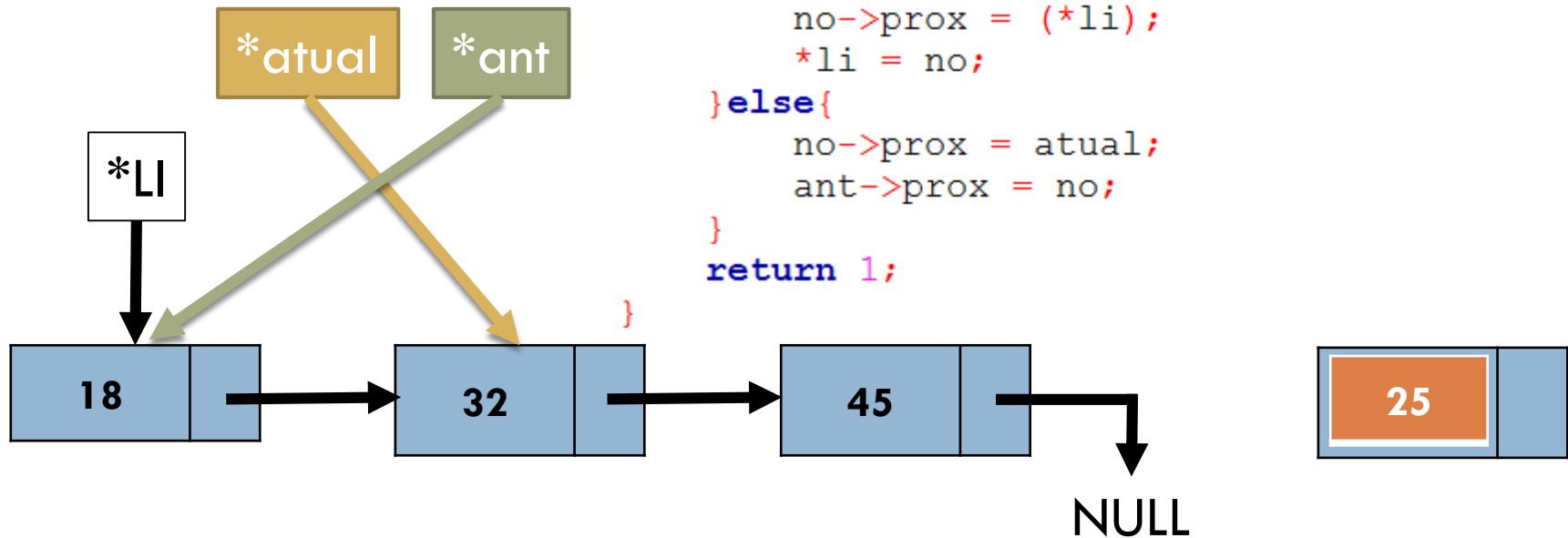
Lista Dinâmica: Inserção Ordenada

112

AL: 25

```
else{
    Elem *ant, *atual = *li;
    while(atual != NULL &&
        atual->dados.matricula < al.matricula){

        ant = atual;
        atual = atual->prox;
    }
    → if(atual == *li){//insere início
        no->prox = (*li);
        *li = no;
    }else{
        no->prox = atual;
        ant->prox = no;
    }
    return 1;
}
```



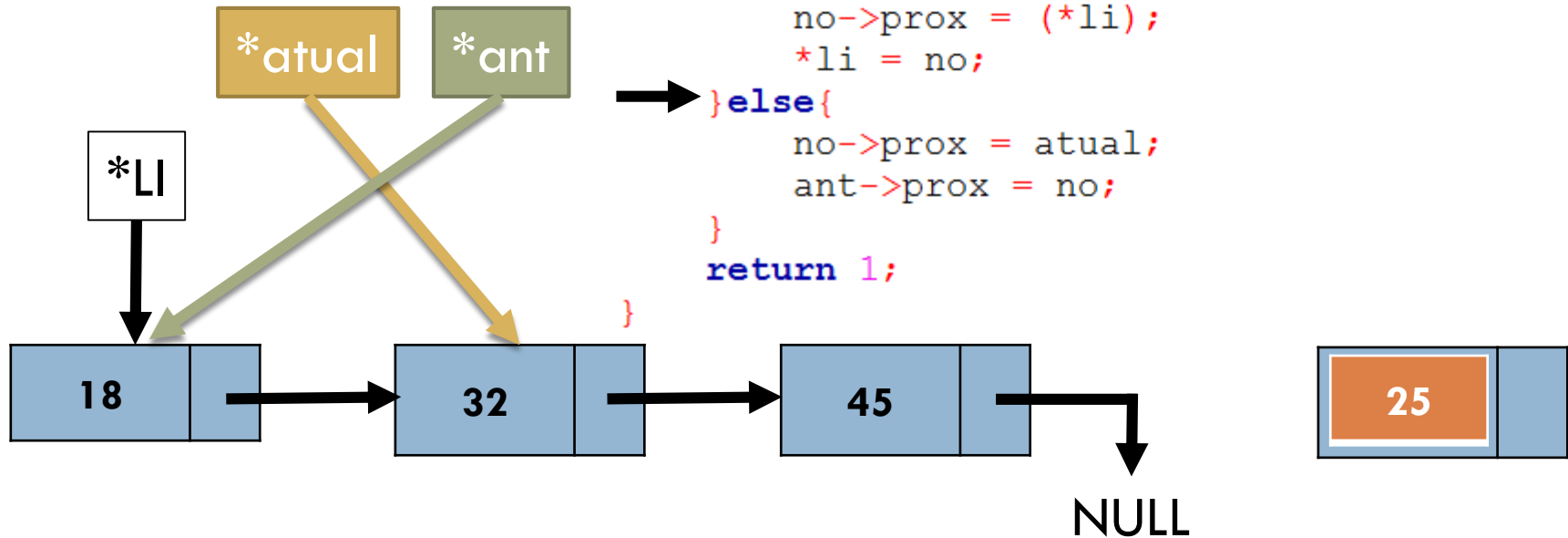
Lista Dinâmica: Inserção Ordenada

113

AL: 25

```
else{
    Elem *ant, *atual = *li;
    while(atual != NULL &&
        atual->dados.matricula < al.matricula){

        ant = atual;
        atual = atual->prox;
    }
    if(atual == *li){//insere início
        no->prox = (*li);
        *li = no;
    }else{
        no->prox = atual;
        ant->prox = no;
    }
    return 1;
}
```



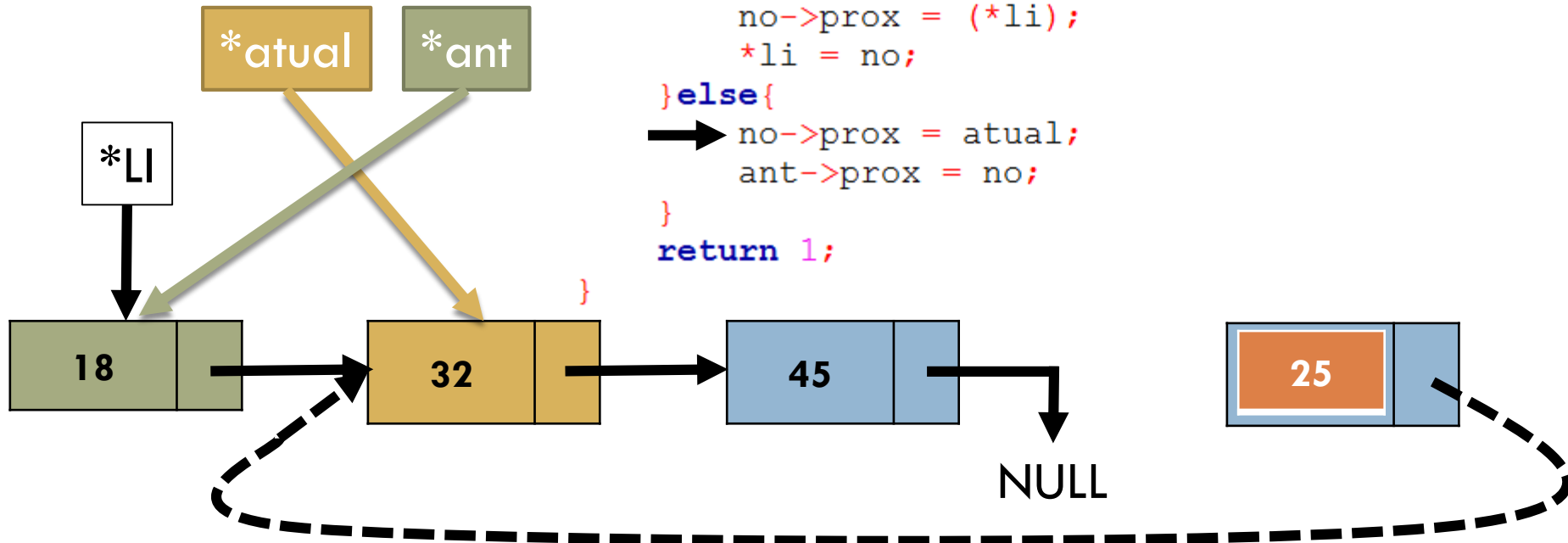
Lista Dinâmica: Inserção Ordenada

114

AL: 25

```
else{
    Elem *ant, *atual = *li;
    while(atual != NULL &&
        atual->dados.matricula < al.matricula){

        ant = atual;
        atual = atual->prox;
    }
    if(atual == *li){//insere início
        no->prox = (*li);
        *li = no;
    }else{
        → no->prox = atual;
        ant->prox = no;
    }
    return 1;
}
```



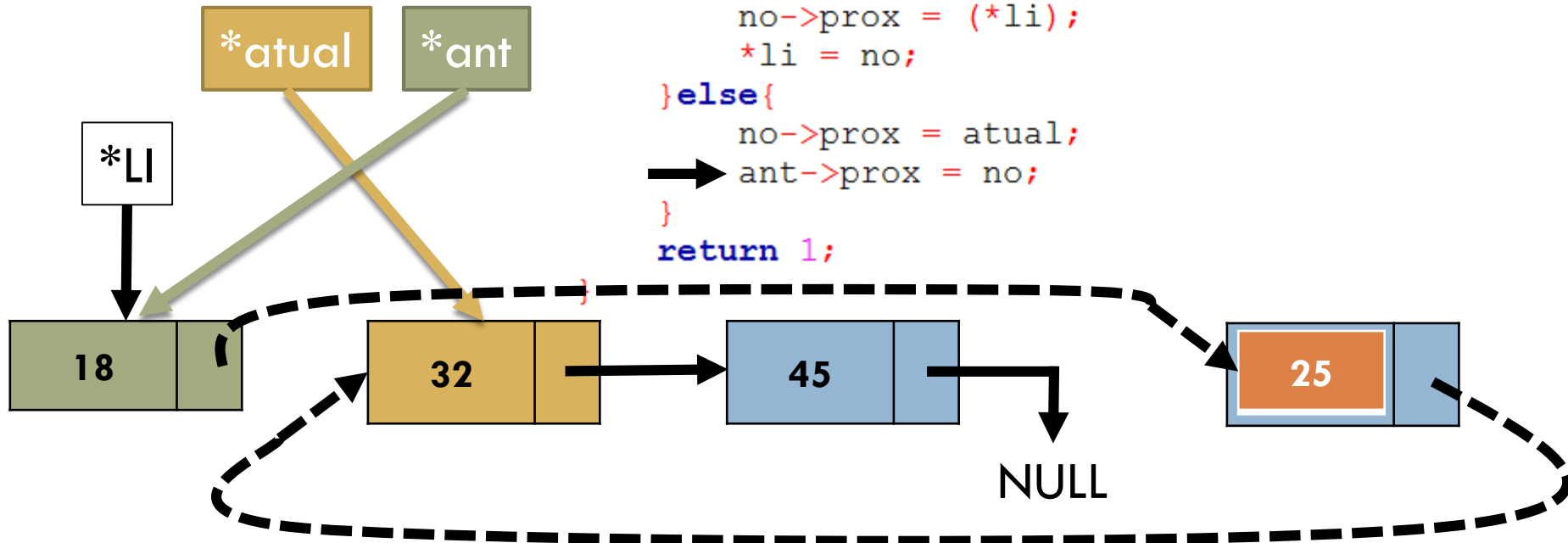
Lista Dinâmica: Inserção Ordenada

115

AL: 25

```
else{
    Elem *ant, *atual = *li;
    while(atual != NULL &&
        atual->dados.matricula < al.matricula){

        ant = atual;
        atual = atual->prox;
    }
    if(atual == *li){//insere início
        no->prox = (*li);
        *li = no;
    }else{
        no->prox = atual;
        ant->prox = no;
    }
    return 1;
}
```



Lista Dinâmica: Inserção Ordenada

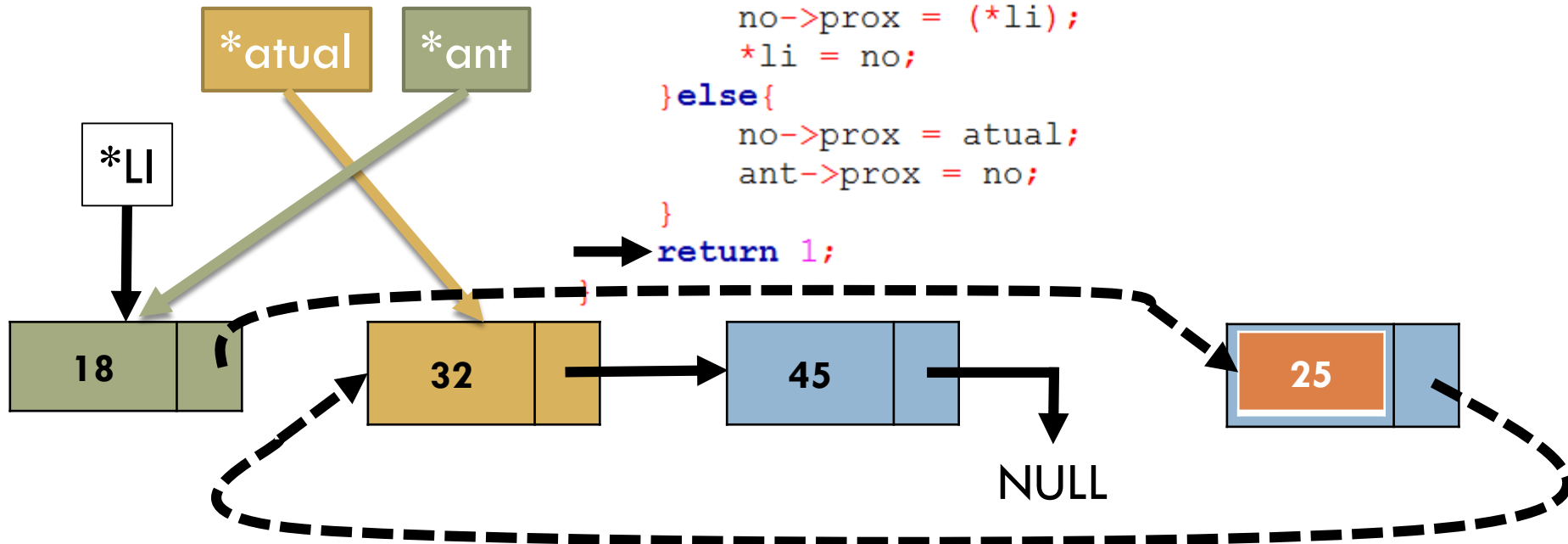
116

AL: 25

```
else{  
    Elem *ant = *atual -> prox;  
    while
```

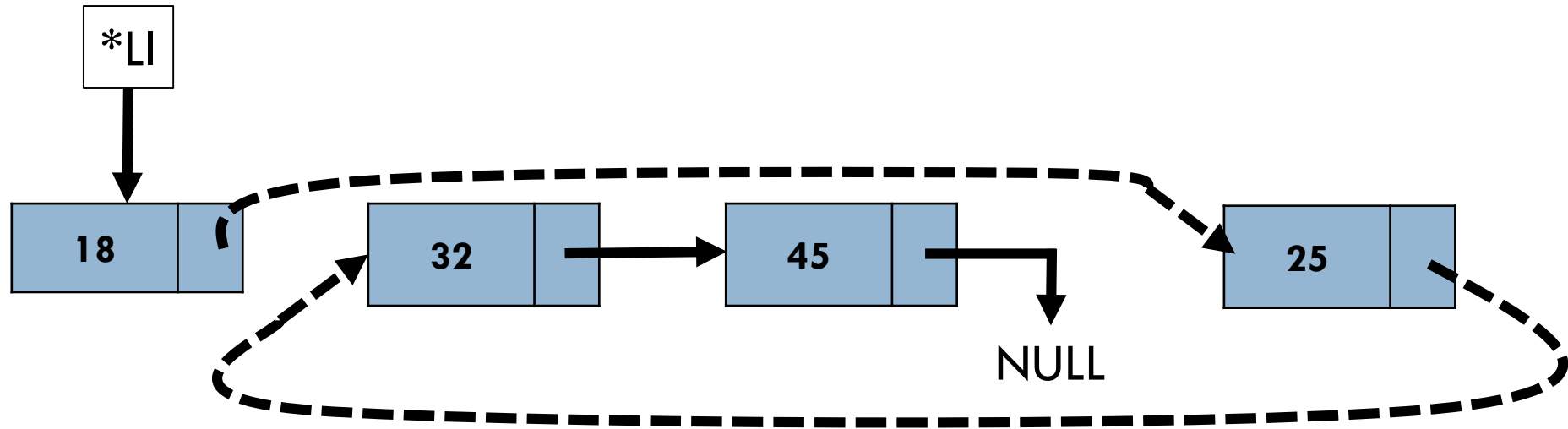
- RETORNA OK;
- FINAL DA INSERÇÃO DO ELEMENTO.

```
}  
if(atual == *li){//insere início  
    no->prox = (*li);  
    *li = no;  
}  
else{  
    no->prox = atual;  
    ant->prox = no;  
}  
return 1;
```



Lista Dinâmica: Inserção Ordenada

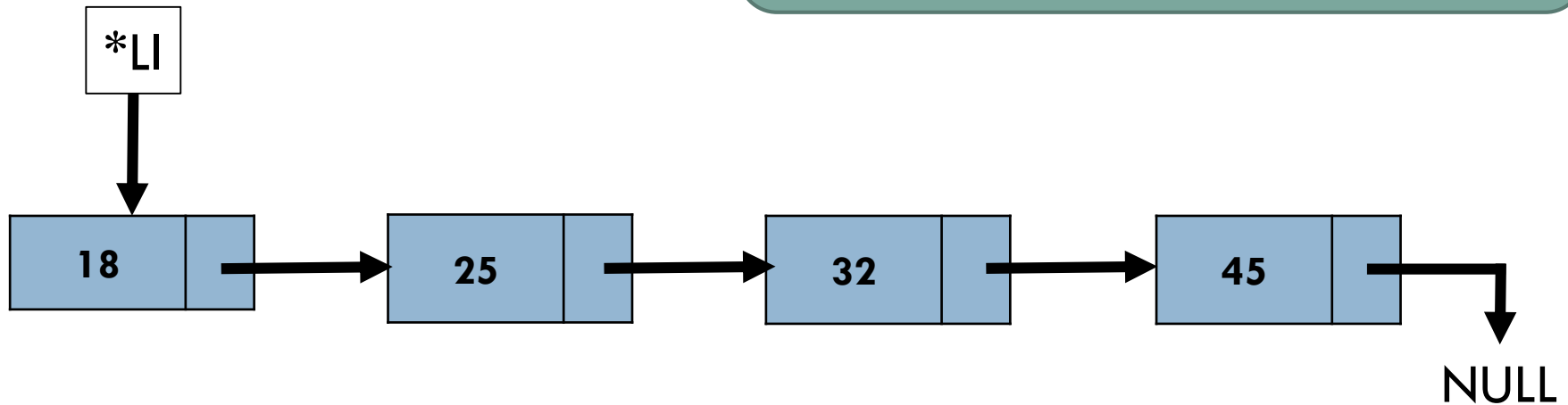
117



Lista Dinâmica: Inserção Ordenada

118

REORGANIZANDO A IMAGEM

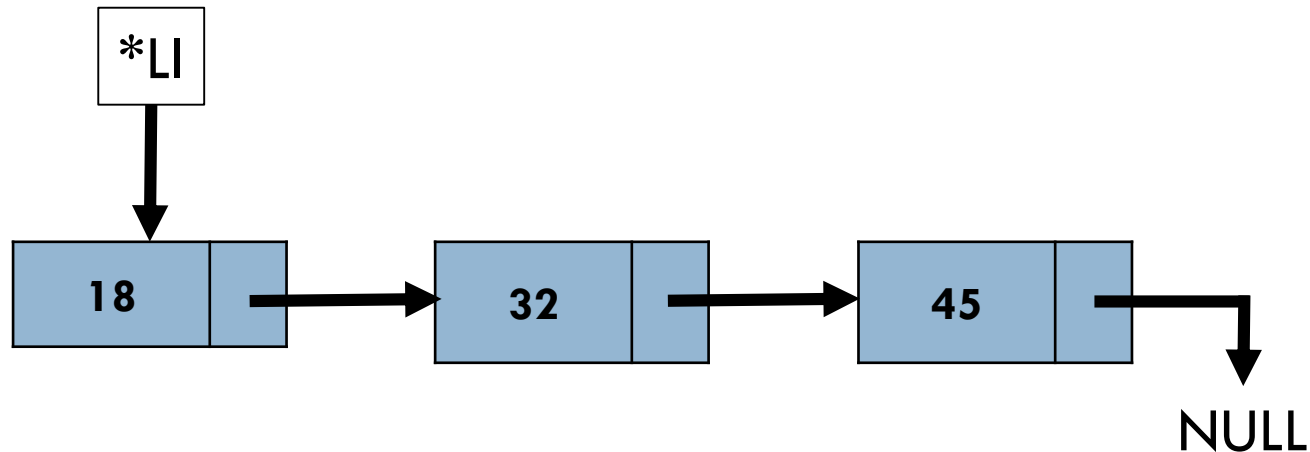


Lista Dinâmica: Remoção

119

32

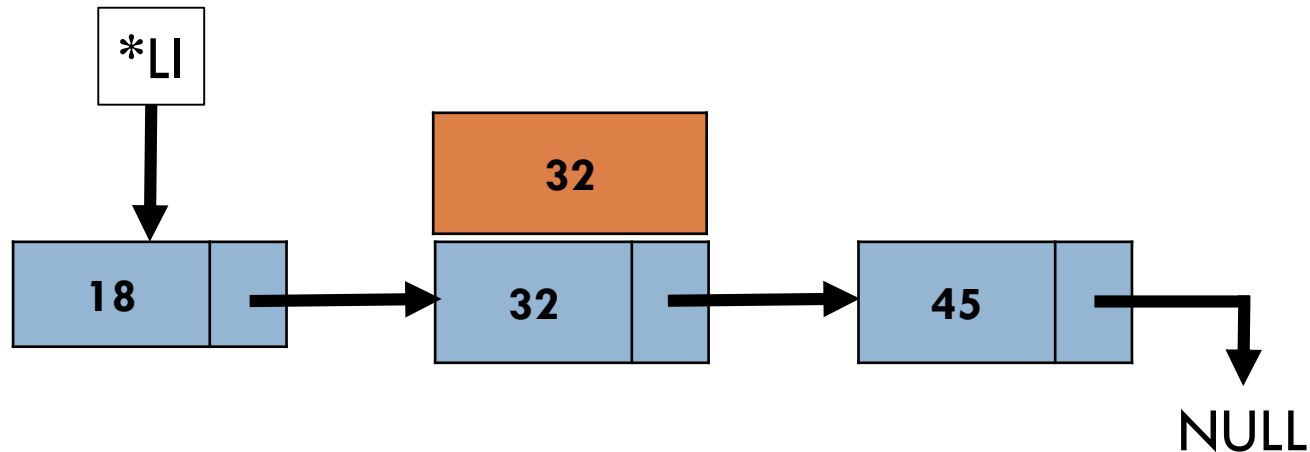
**RECEBER CHAVE DO DADO QUE SERÁ
REMOVIDO DA LISTA**



Lista Dinâmica: Remoção

120

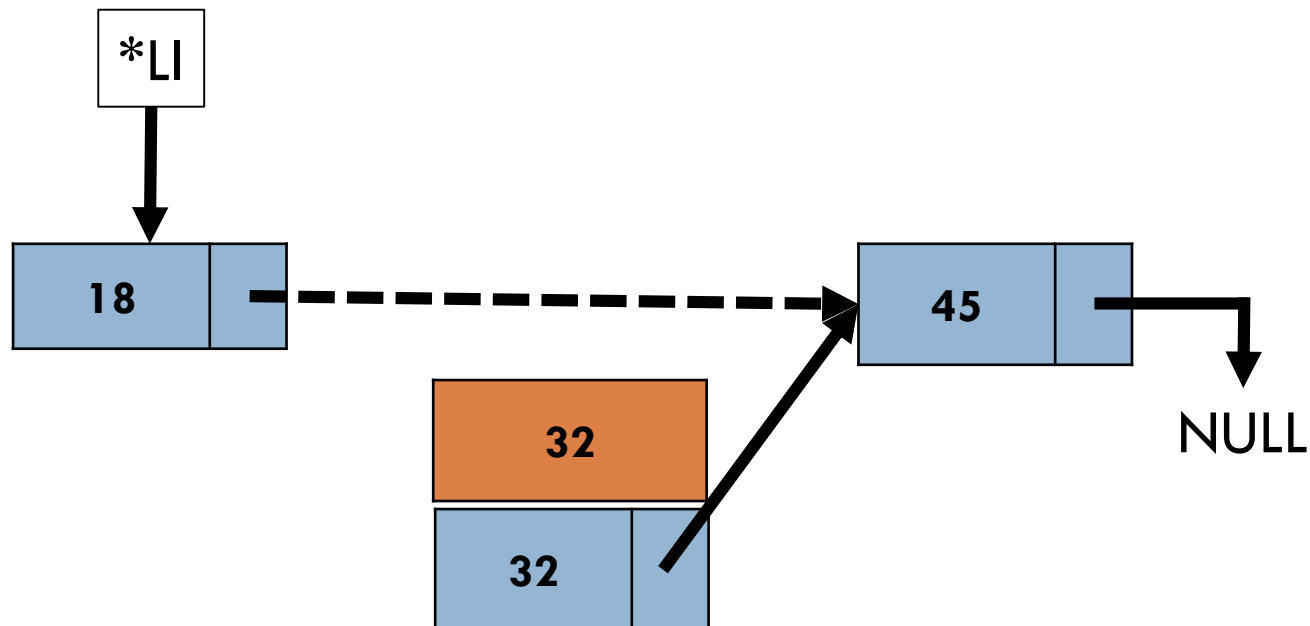
VARRER A LISTA PARA ENCONTRAR A CHAVE INFORMADA



Lista Dinâmica: Remoção

121

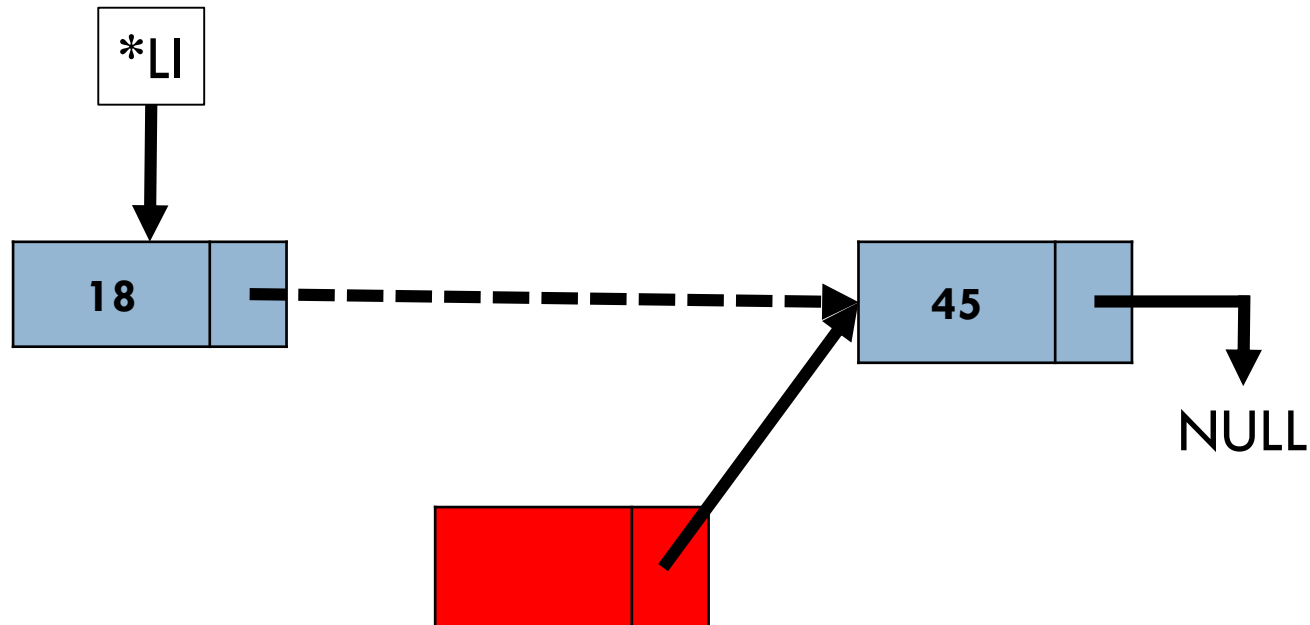
REORGANIZAR PONTEIROS (ANTERIOR E PRÓXIMO) PARA MANTER ENCADEADO



Lista Dinâmica: Remoção

122

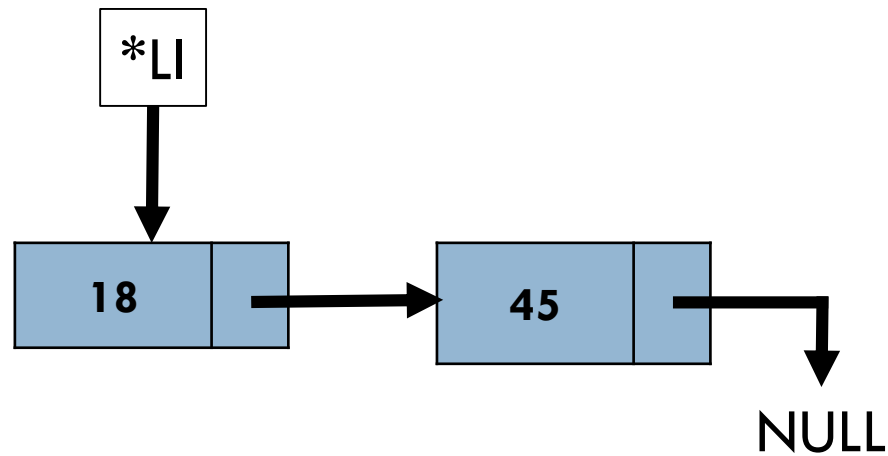
REMOVER O ELEMENTO DA MEMÓRIA



Lista Dinâmica: Remoção

123

LISTA APÓS REMOÇÃO



Lista Dinâmica: Remoção

124

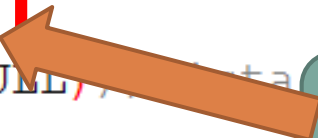
```
int remove_lista(Lista* li, int mat){
    if(li == NULL)
        return 0;
    if((*li) == NULL) //lista vazia
        return 0;
    Elem *ant, *no = *li;
    while(no != NULL && no->dados.matricula != mat){
        ant = no;
        no = no->prox;
    }
    if(no == NULL) //não encontrado
        return 0;

    if(no == *li) //remover o primeiro?
        *li = no->prox;
    else
        ant->prox = no->prox;
    free(no);
    return 1;
}
```

Lista Dinâmica: Remoção

125

```
int remove_lista(Lista* li, int mat){  
    { if(li == NULL) }  
        return 0;  
    if((*li) == NULL) // lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
    if(no == NULL) // não encontrado  
        return 0;  
  
    if(no == *li) // remover o primeiro?  
        *li = no->prox;  
    else  
        ant->prox = no->prox;  
    free(no);  
    return 1;  
}
```



VERIFICAR SE A LISTA EXISTE

Lista Dinâmica: Remoção

126

```
int remove_lista(Lista* li, int mat){
    if(li == NULL)
        return 0;
    if((*li) == NULL) //lista vazia
        return 0;
    Elem *ant, *no = *li;
    while(no != NULL && no->mat != mat)
        ant = no;
        no = no->prox;
    if(no == NULL) //não encontrado
        return 0;
    if(no == *li) //remover o primeiro?
        *li = no->prox;
    else
        ant->prox = no->prox;
    free(no);
    return 1;
}
```



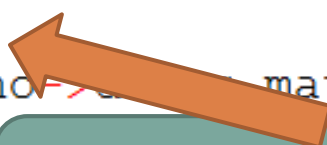
VERIFICAR SE A LISTA ESTÁ VAZIA

Lista Dinâmica: Remoção

127

```
int remove_lista(Lista* li, int mat){
    if(li == NULL)
        return 0;
    if((*li) == NULL) //lista vazia
        return 0;
    Elem *ant, *no = *li;
    while(no != NULL && no->matricula != mat){
        ant = no;
        no = no->prox;
    }
    if(no == NULL) //não encontrado
        return 0;

    if(no == *li) //remover o primeiro?
        *li = no->prox;
    else
        ant->prox = no->prox;
    free(no);
    return 1;
}
```



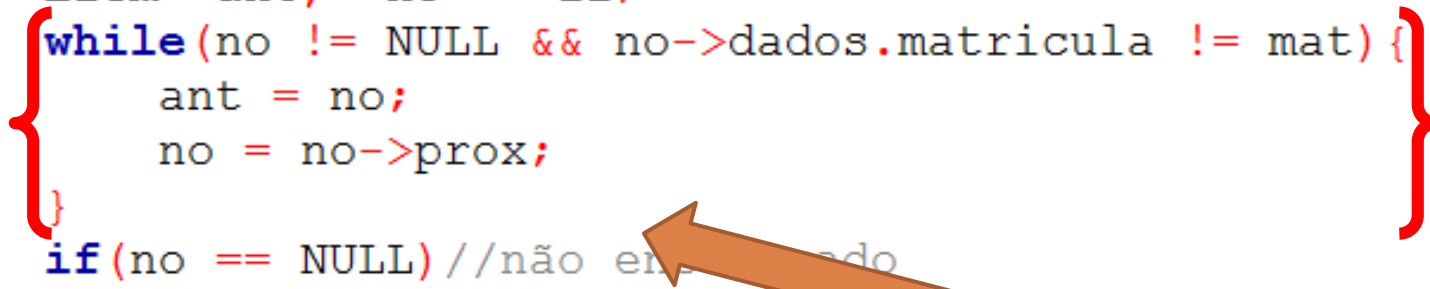
criar auxiliares para manipular ponteiros

Lista Dinâmica: Remoção

128

```
int remove_lista(Lista* li, int mat){
    if(li == NULL)
        return 0;
    if((*li) == NULL) //lista vazia
        return 0;
    Elem *ant, *no = *li;
    {
        while(no != NULL && no->dados.matricula != mat){
            ant = no;
            no = no->prox;
        }
    }
    if(no == NULL) //não encontrado
        return 0;

    if(no == *li) //remove o primeiro elemento
        *li = no->prox;
    else
        ant->prox = no->prox;
    free(no);
    return 1;
}
```

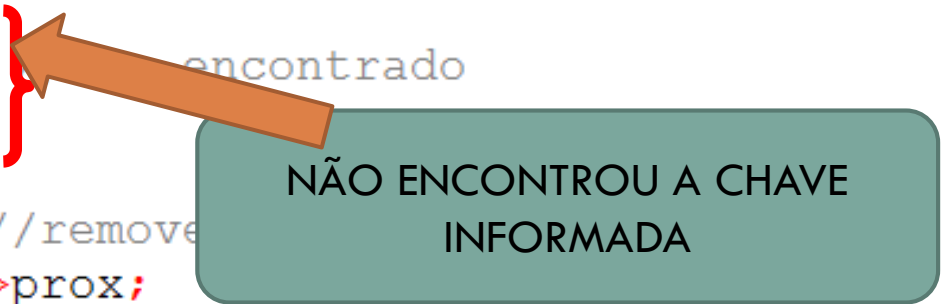


VARRER LISTA PARA ENCONTRAR
CHAVE INFORMADA

Lista Dinâmica: Remoção

129

```
int remove_lista(Lista* li, int mat){
    if(li == NULL)
        return 0;
    if((*li) == NULL) //lista vazia
        return 0;
    Elem *ant, *no = *li;
    while(no != NULL && no->dados.matricula != mat){
        ant = no;
        no = no->prox;
    }
    if(no == NULL) {
        return 0;
    }
    if(no == *li) //remove
        *li = no->prox;
    else
        ant->prox = no->prox;
    free(no);
    return 1;
}
```



encontrado

NÃO ENCONTROU A CHAVE INFORMADA

Lista Dinâmica: Remoção

130

```
int remove_lista(Lista* li, int mat){
    if(li == NULL)
        return 0;
    if((*li) == NULL) //lista vazia
        return 0;
    Elem *ant, *no = *li;
    while(no != NULL && no->dados.matricula != mat){
        ant = no;
        no = no->prox;
    }
    if(no == NULL) //não encontrado
        return 0;
    if(no == *li) //remover o primeiro elemento
        *li = no->prox;
    else
        ant->prox = no->prox;
    free(no);
    return 1;
}
```



EM CASO DE REMOÇÃO DO
PRIMEIRO ELEMENTO

Lista Dinâmica: Remoção

131

```
int remove_lista(Lista* li, int mat){
    if(li == NULL)
        return 0;
    if((*li) == NULL) //lista vazia
        return 0;
    Elem *ant, *no = *li;
    while(no != NULL && no->dados.matricula != mat){
        ant = no;
        no = no->prox;
    }
    if(no == NULL) //não encontrado
        return 0;

    if(no == *li) //remover o primeiro
        *li = no->prox;
    else
        ant->prox = no->prox;
    free(no);
    return 1;
}
```



EM CASO DE REMOÇÃO DE
OUTROS ELEMENTOS

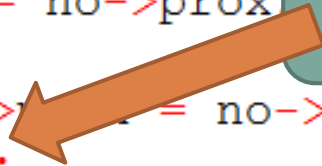
Lista Dinâmica: Remoção

132

```
int remove_lista(Lista* li, int mat){
    if(li == NULL)
        return 0;
    if((*li) == NULL) //lista vazia
        return 0;
    Elem *ant, *no = *li;
    while(no != NULL && no->dados.matricula != mat){
        ant = no;
        no = no->prox;
    }
    if(no == NULL) //não encontrado
        return 0;

    if(no == *li) //remoção do primeiro elemento
        *li = no->prox;
    else
        ant->prox = no->prox;
    free(no);
    return 1;
}
```

DESALOCANDO **ELEMENTO** DA
MEMÓRIA

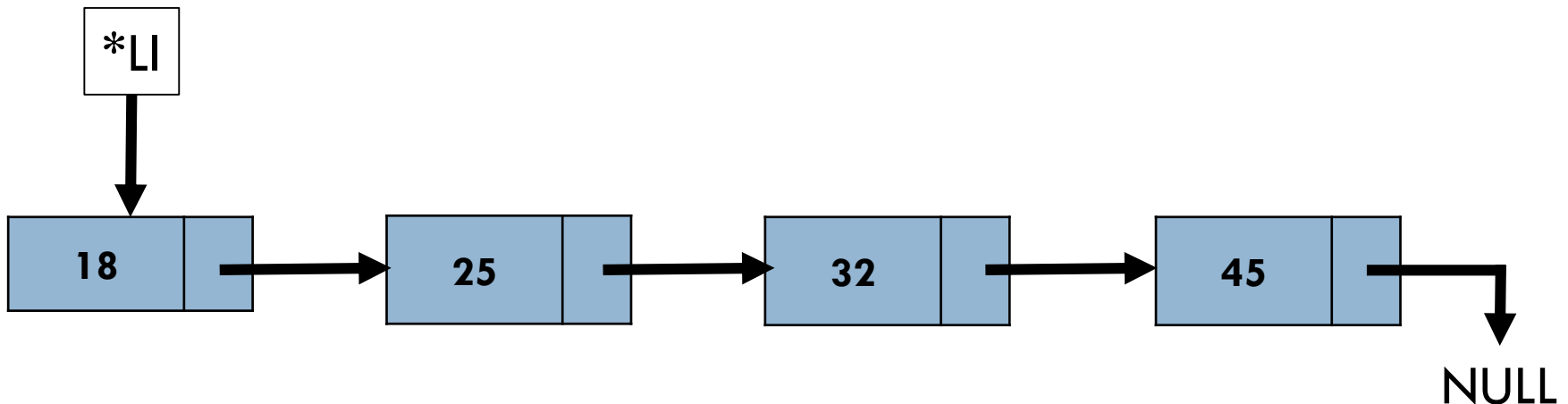


Lista Dinâmica: Remoção

133

32

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){
```

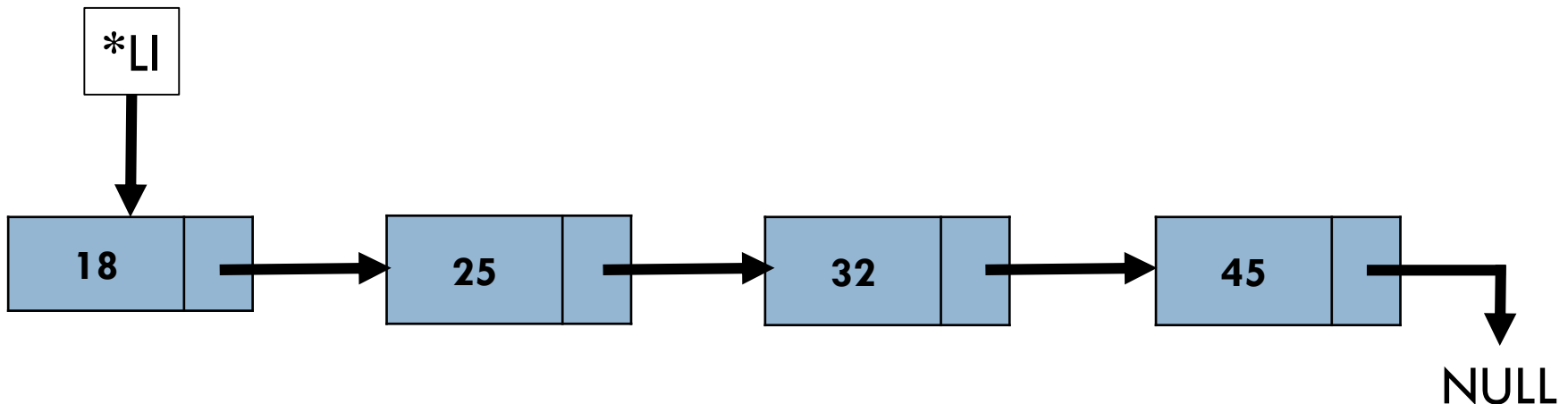


Lista Dinâmica: Remoção

134

32

```
int remove_lista(Lista* li, int mat){  
    → if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){
```

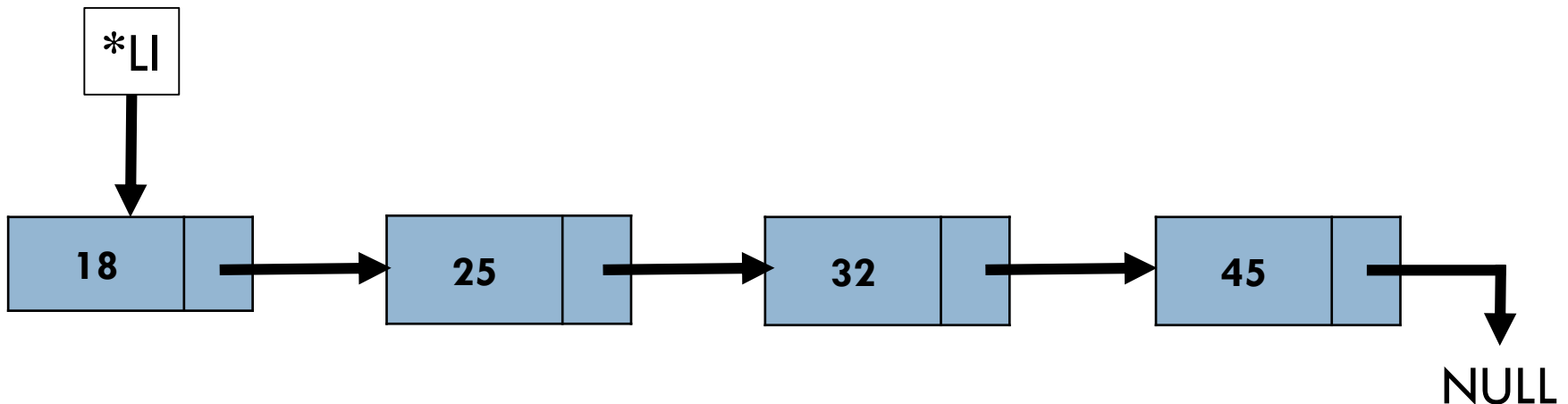


Lista Dinâmica: Remoção

135

32

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    → if((*li) == NULL) //lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){
```

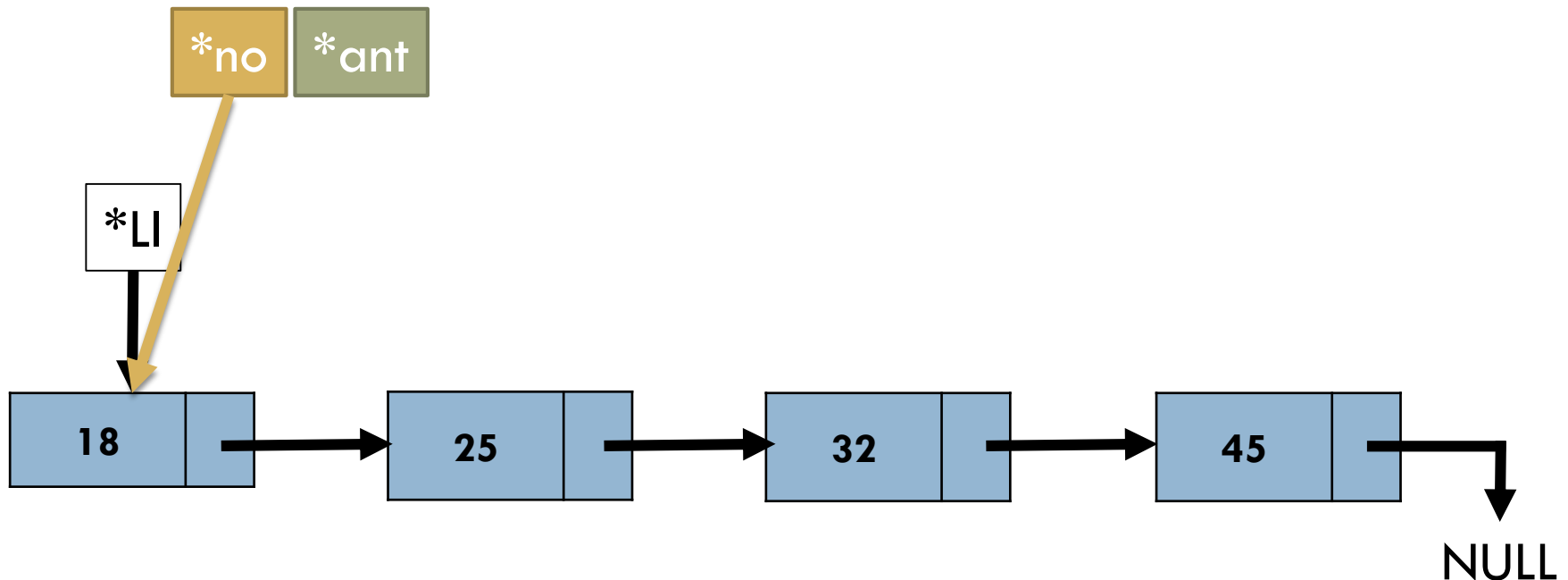


Lista Dinâmica: Remoção

136

32

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    → Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){
```

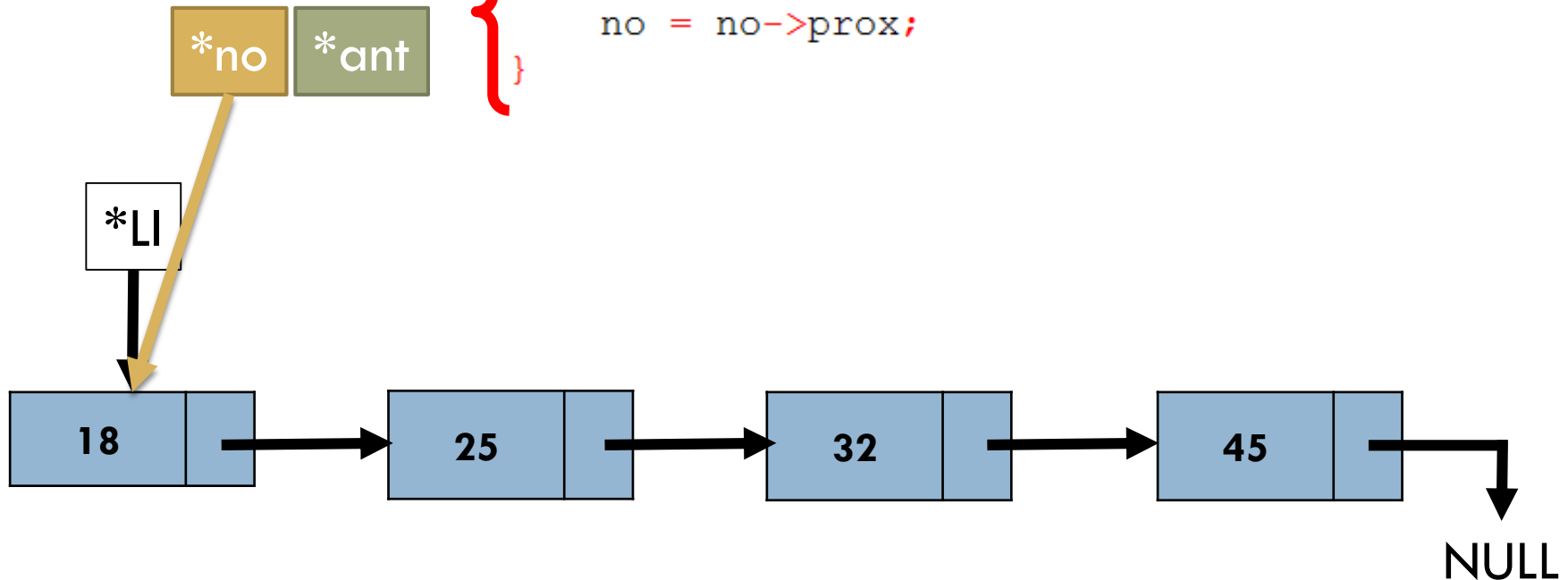


Lista Dinâmica: Remoção

137

32

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    → while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
}
```

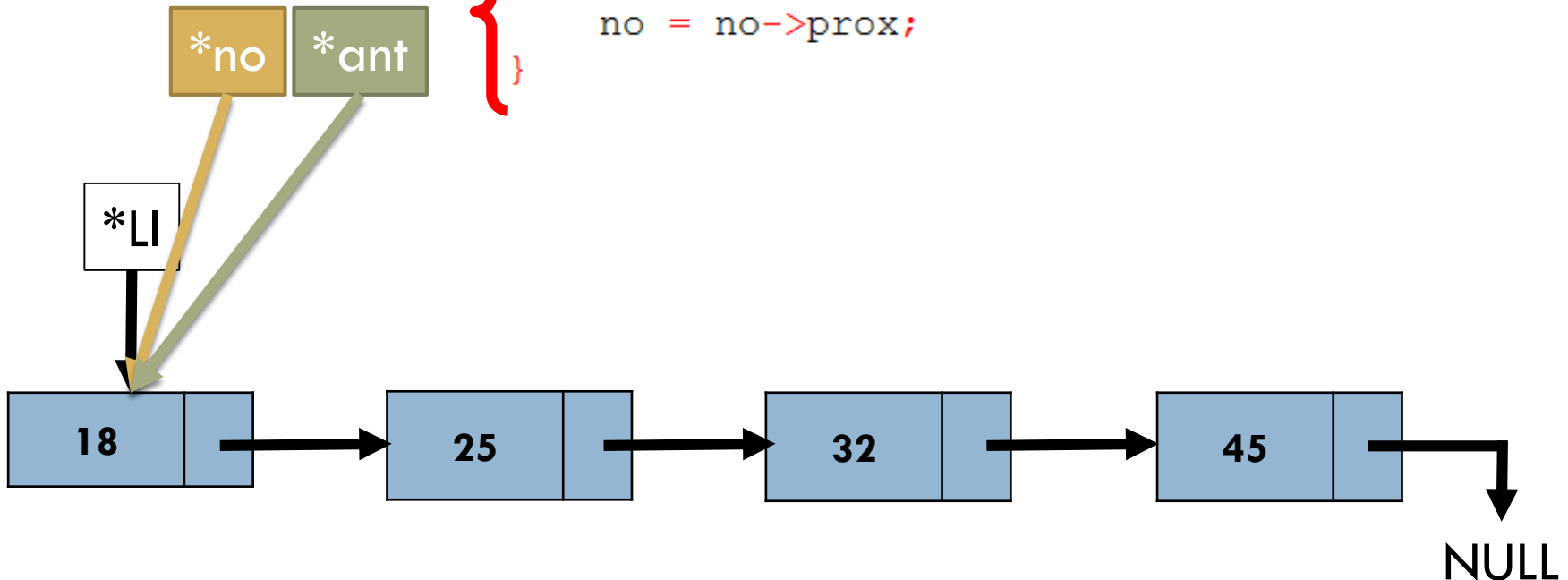


Lista Dinâmica: Remoção

138

32

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
}
```

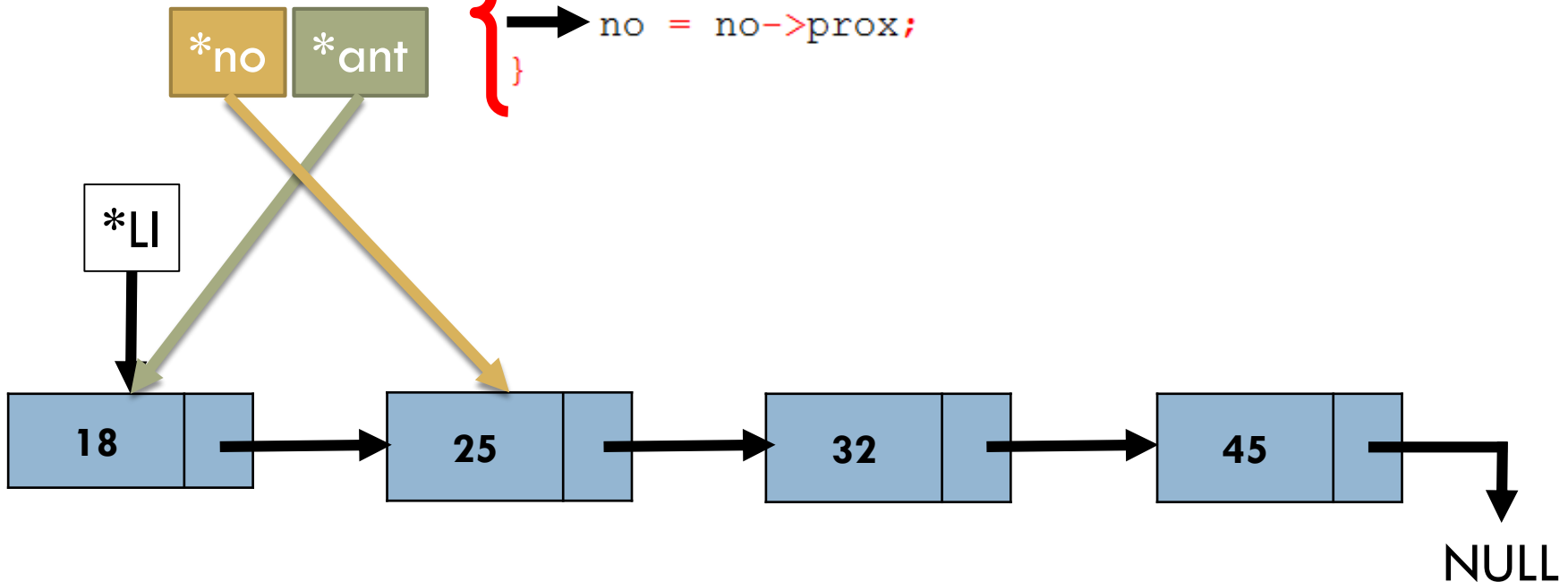


Lista Dinâmica: Remoção

139

32

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
}
```

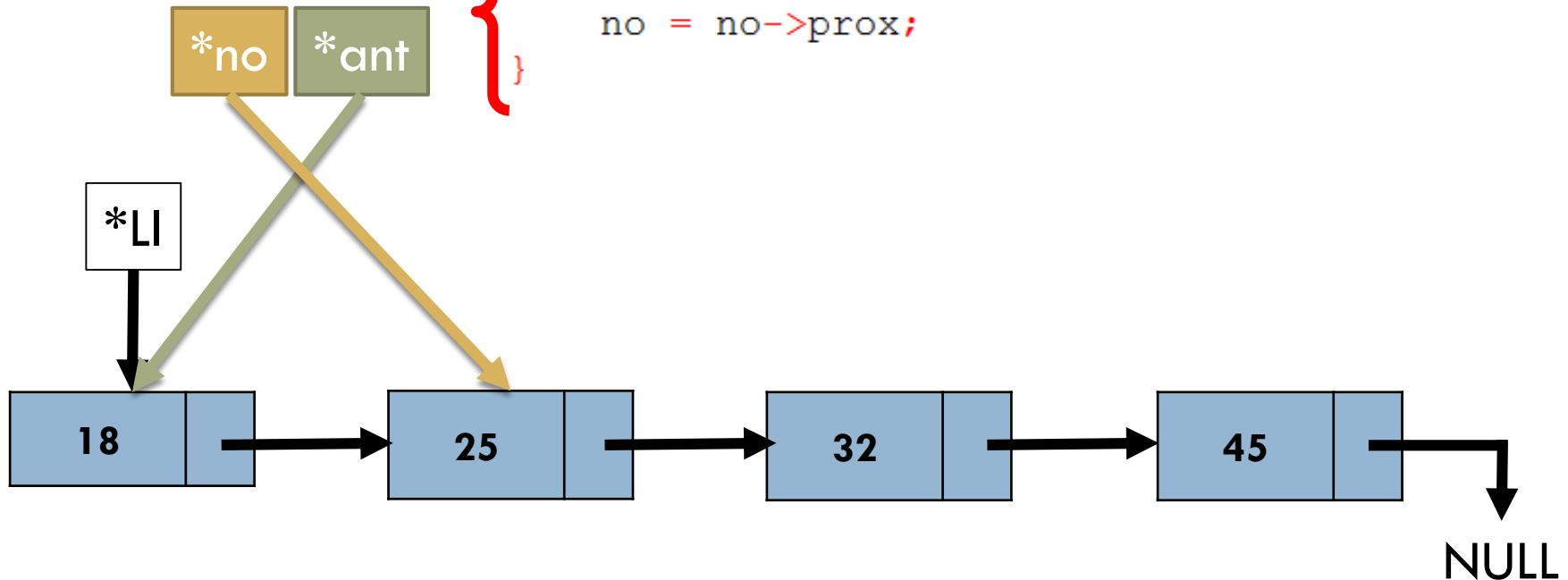


Lista Dinâmica: Remoção

140

32

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
}
```

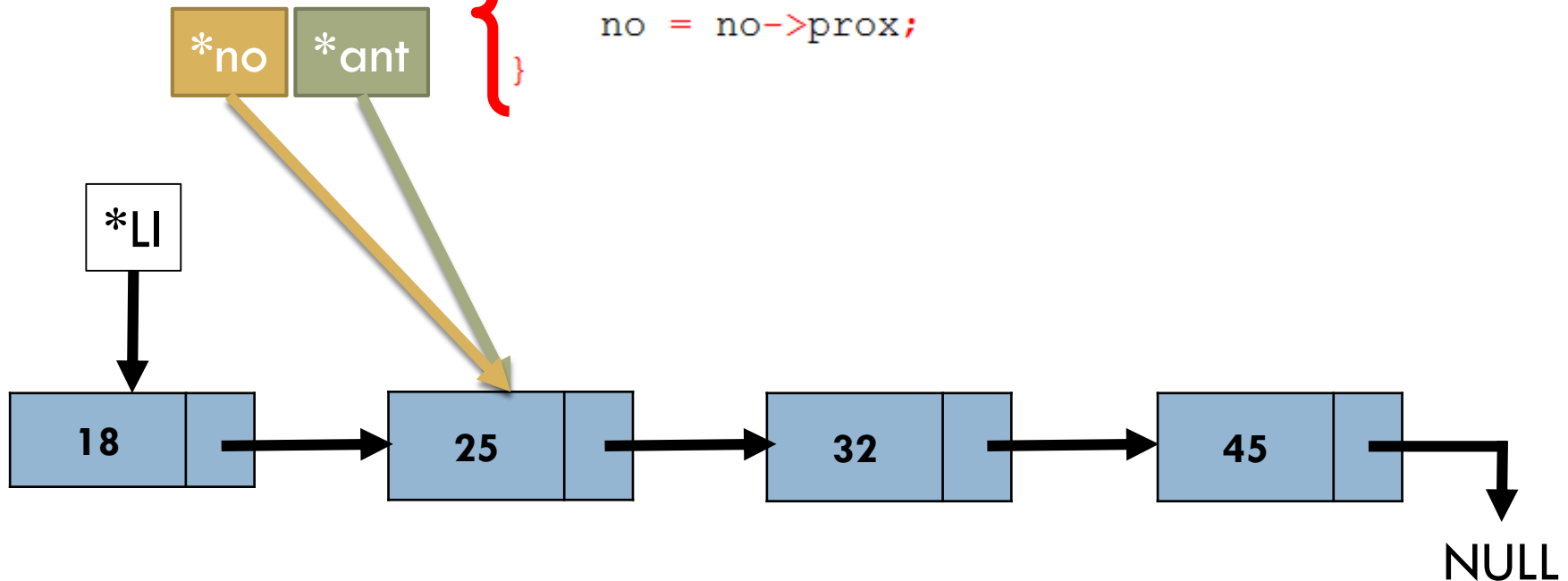


Lista Dinâmica: Remoção

141

32

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
}
```

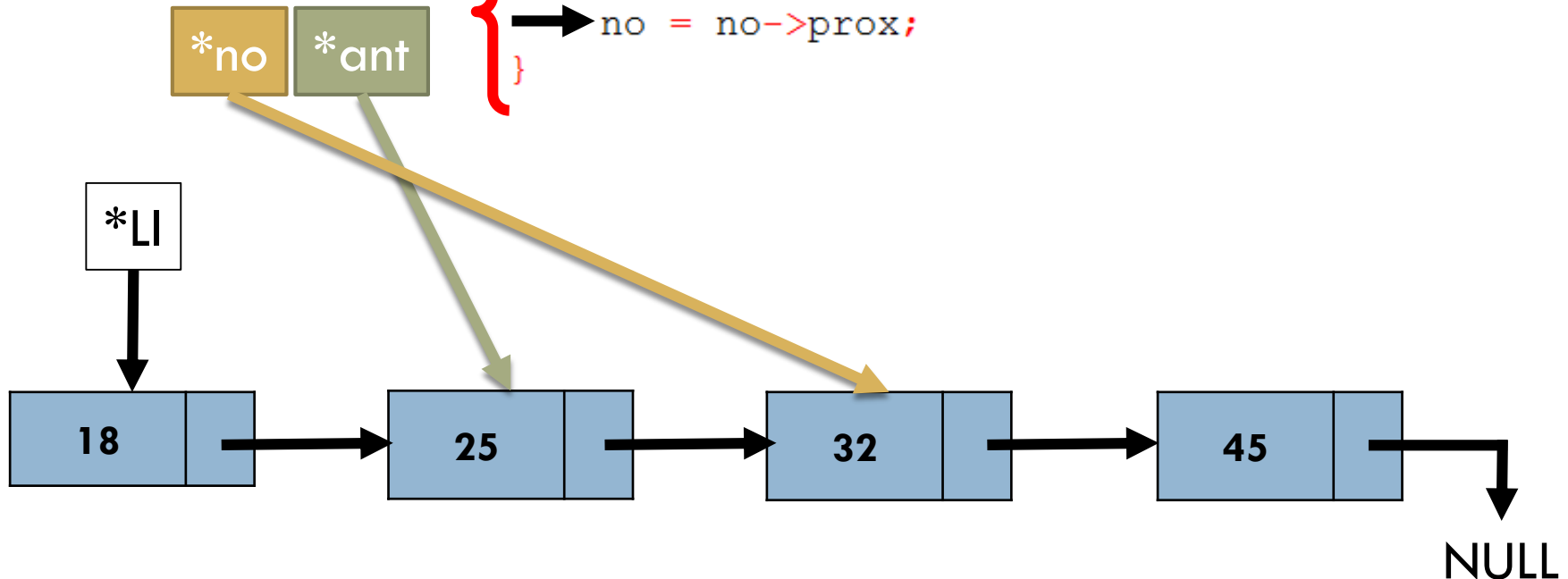


Lista Dinâmica: Remoção

142

32

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
}
```

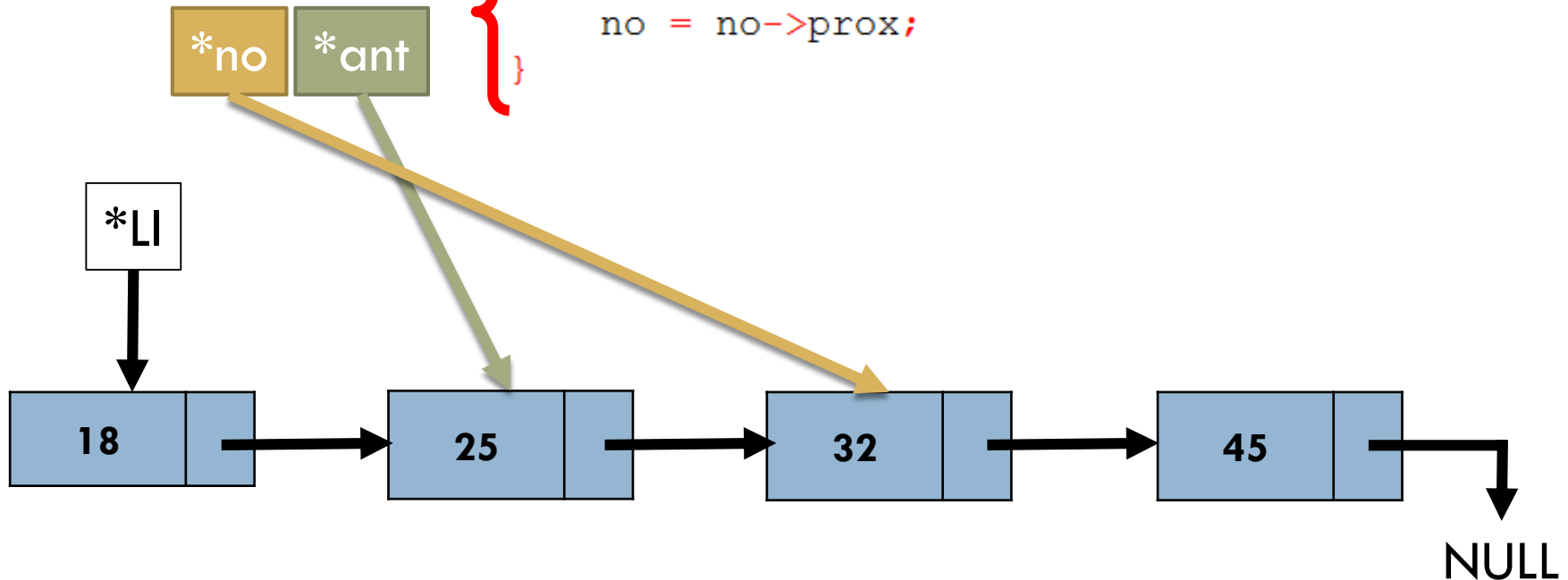


Lista Dinâmica: Remoção

143

32

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
}
```



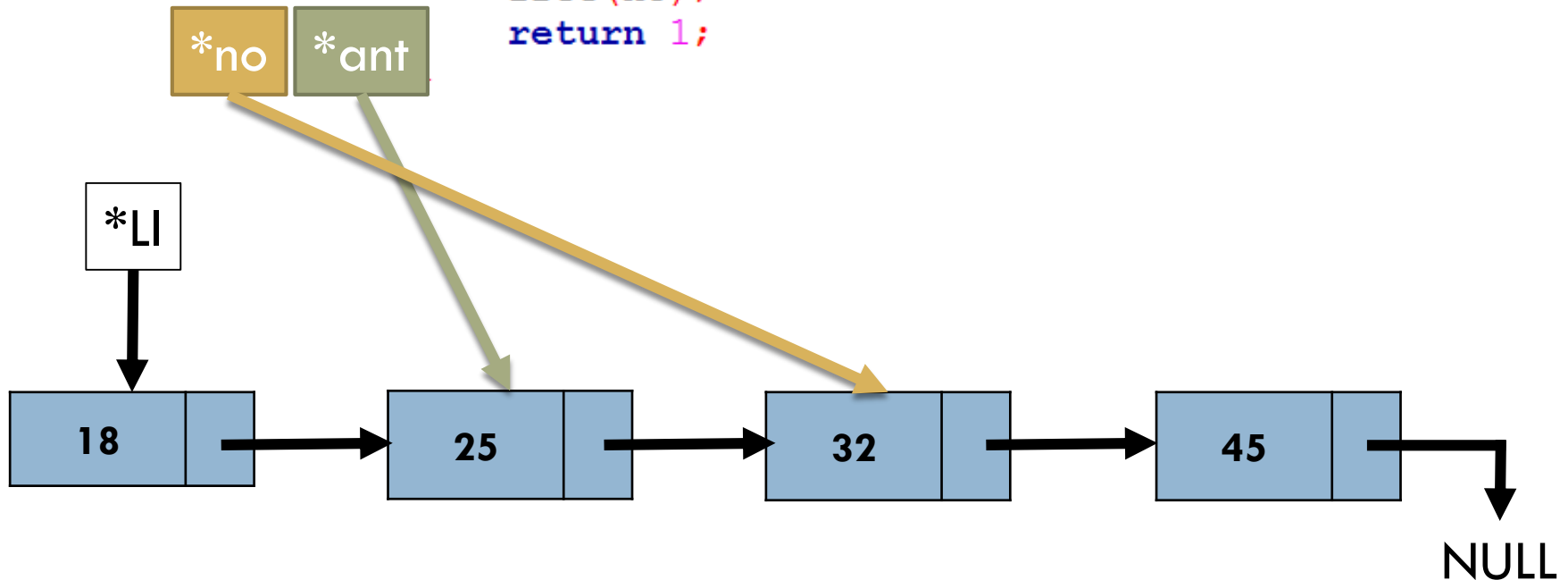
Lista Dinâmica: Remoção

144

32

```
→ if(no == NULL) //não encontrado  
    return 0;
```

```
if(no == *li) //remover o primeiro?  
    *li = no->prox;  
else  
    ant->prox = no->prox;  
free(no);  
return 1;
```



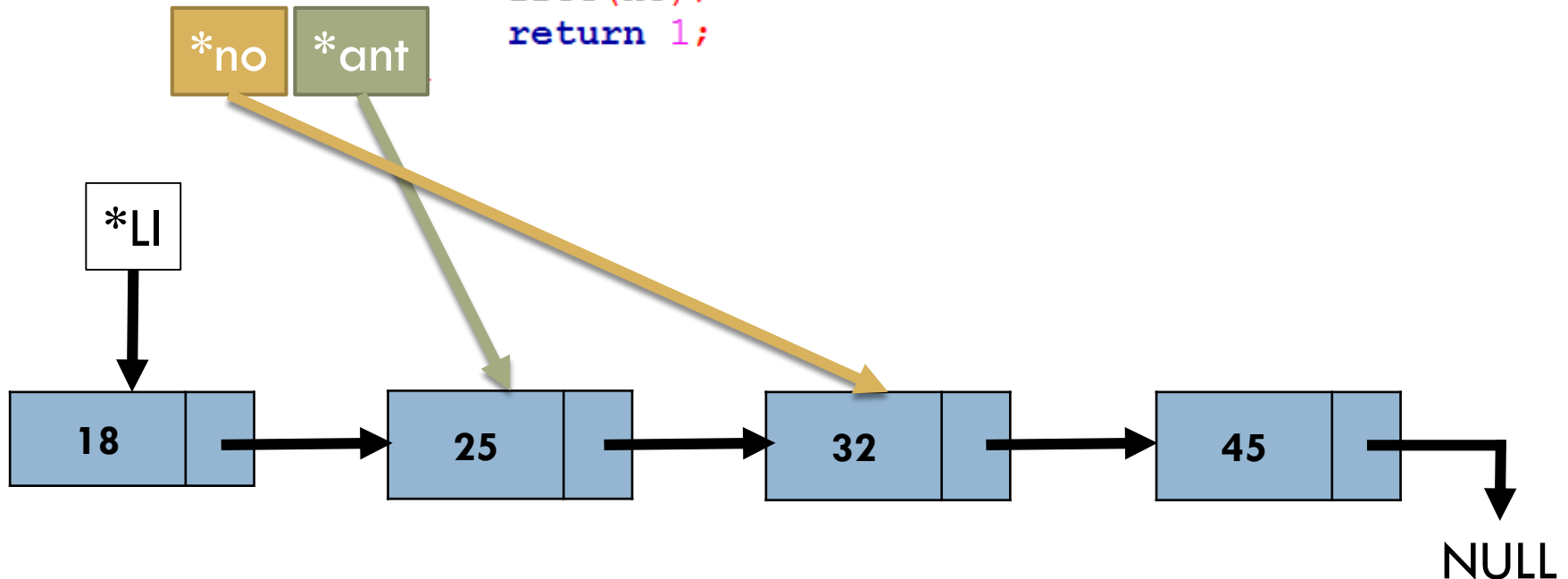
Lista Dinâmica: Remoção

145

32

```
if(no == NULL) //não encontrado  
    return 0;
```

```
→ if(no == *li) //remover o primeiro?  
    *li = no->prox;  
else  
    ant->prox = no->prox;  
    free(no);  
    return 1;
```



Lista Dinâmica: Remoção

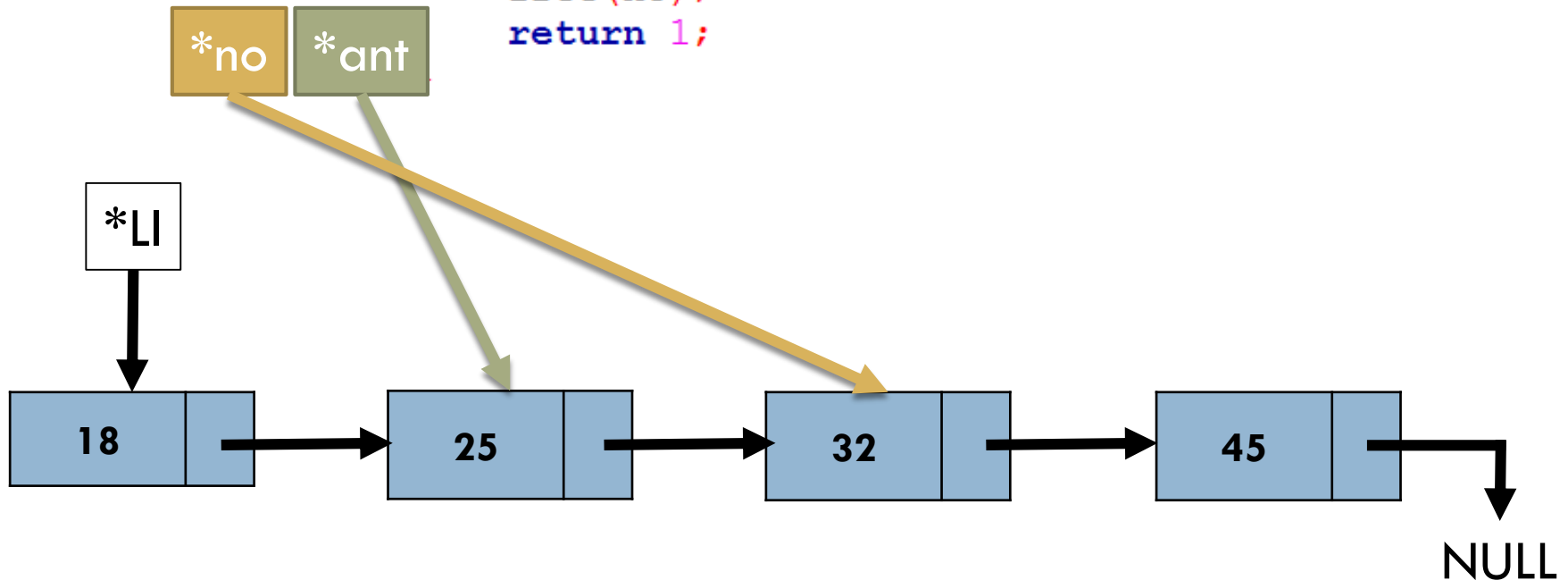
146

32

```
if(no == NULL) //não encontrado  
    return 0;
```

```
if(no == *li) //remover o primeiro?  
    *li = no->prox;
```

```
→ else  
    ant->prox = no->prox;  
    free(no);  
    return 1;
```



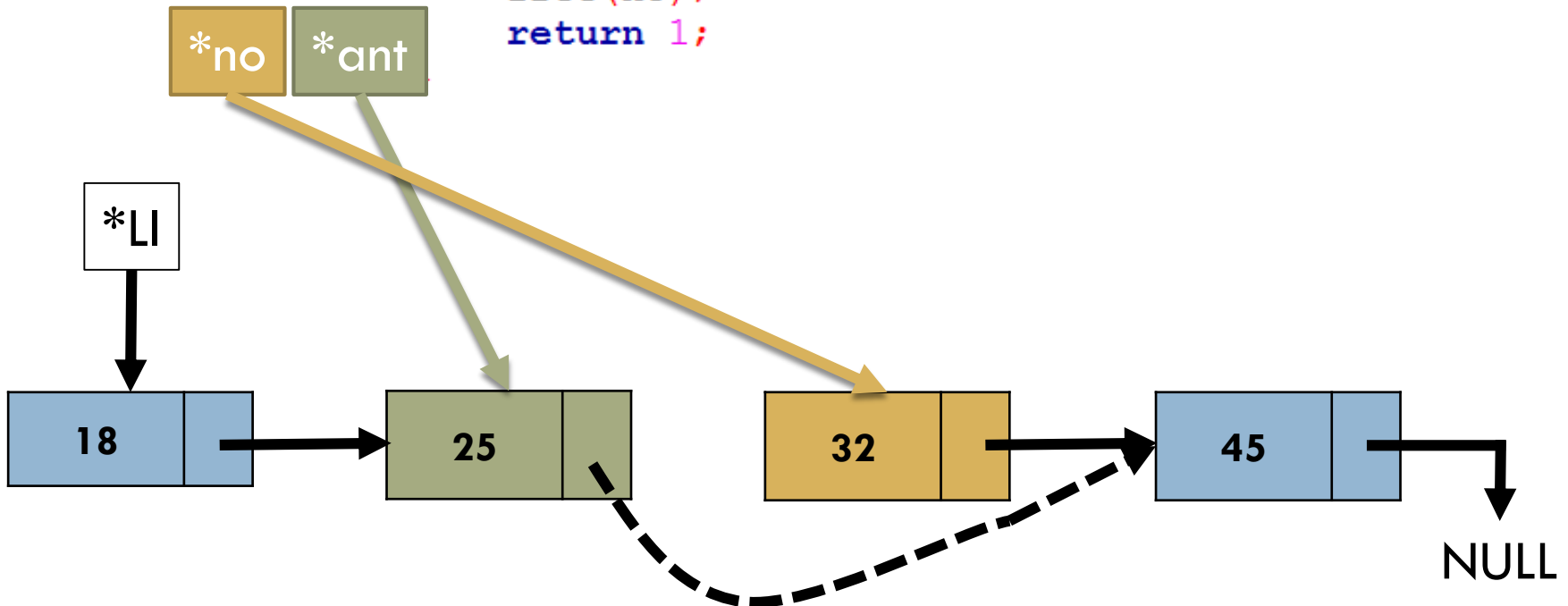
Lista Dinâmica: Remoção

147

32

```
if(no == NULL) //não encontrado
    return 0;

if(no == *li) //remover o primeiro?
    *li = no->prox;
else
    ant->prox = no->prox;
free(no);
return 1;
```



Lista Dinâmica: Remoção

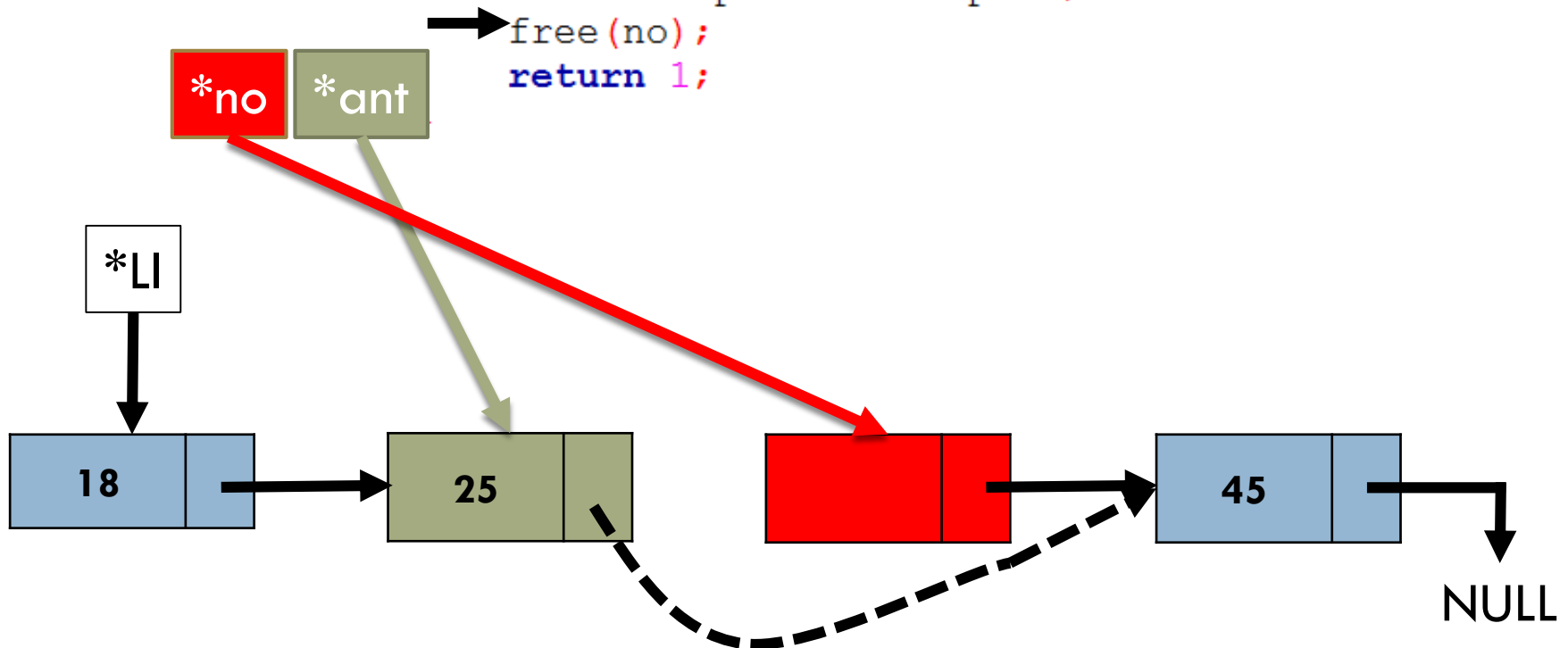
148

32

```
if(no == NULL) //não encontrado  
    return 0;
```

```
if(no == *li) //remover o primeiro?  
    *li = no->prox;
```

```
else  
    ant->prox = no->prox;  
free(no);  
return 1;
```



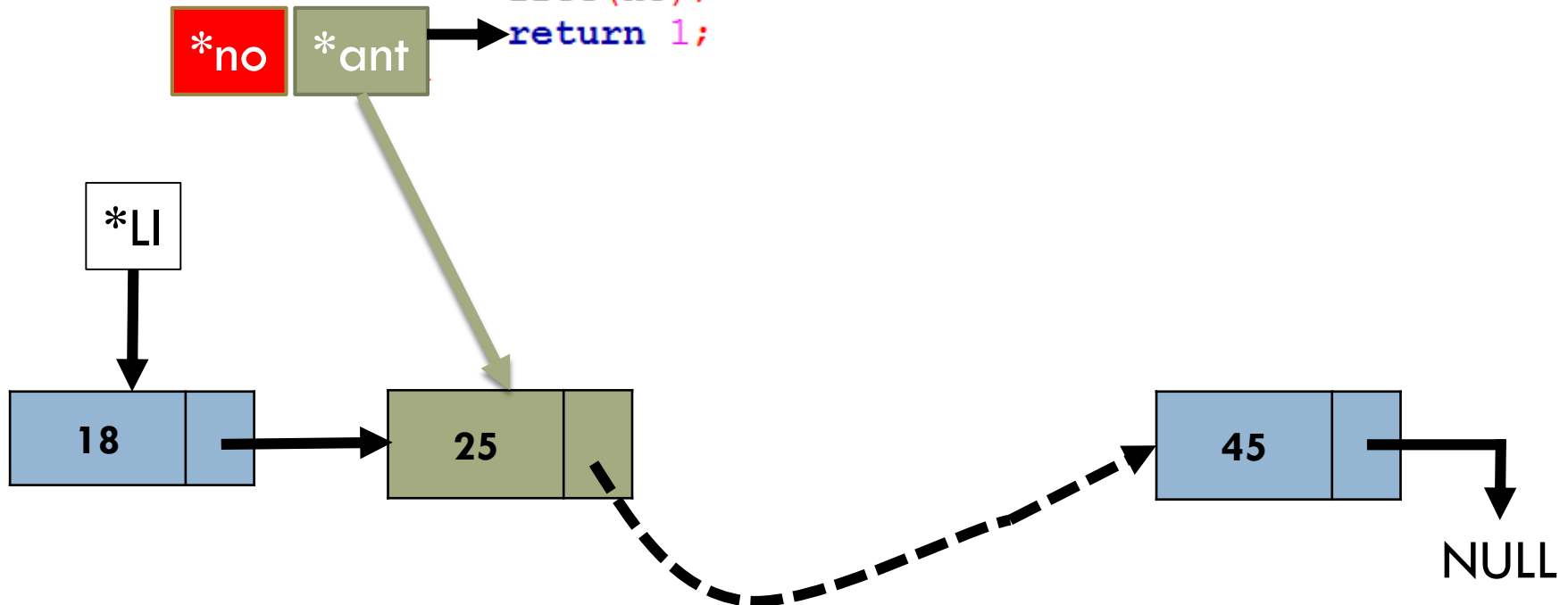
Lista Dinâmica: Remoção

149

32

```
if(no == NULL) //não encontrado  
    return  
  
if(no == *  
    *li =  
else  
    ant->prox = no->prox;  
    free(no);  
    return 1;
```

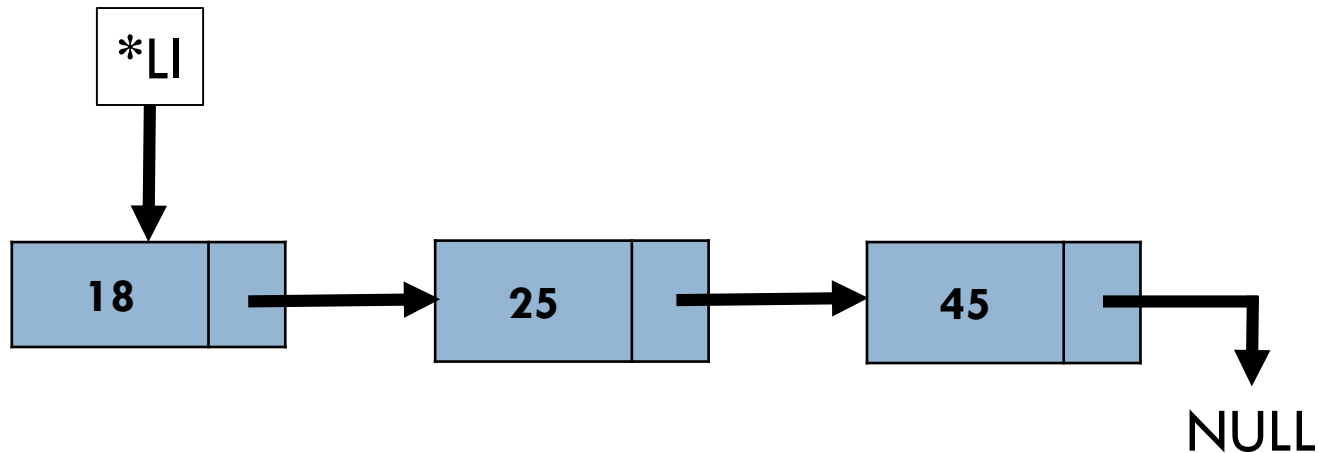
- RETORNA OK;
- FINAL DA REMOÇÃO DO ELEMENTO.



Lista Dinâmica: Remoção

150

REORGANIZANDO A IMAGEM

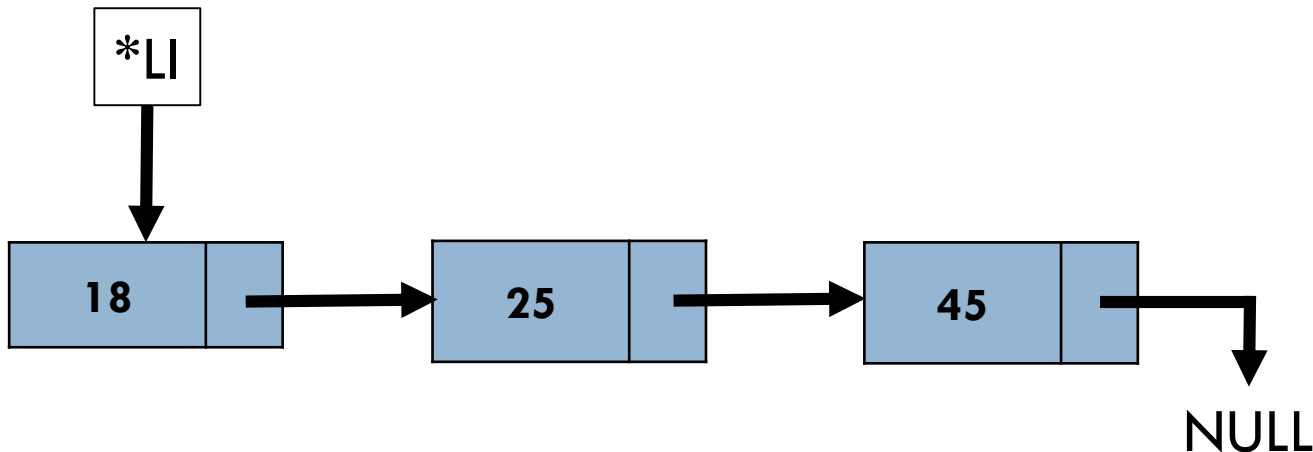


Lista Dinâmica: Remoção

151

18

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){
```

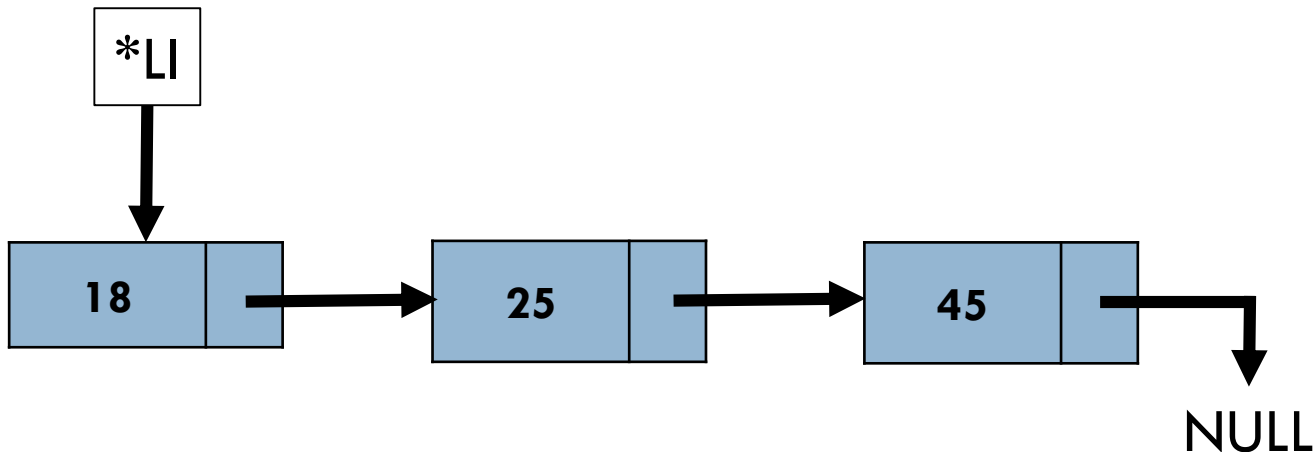


Lista Dinâmica: Remoção

152

18

```
int remove_lista(Lista* li, int mat){  
    → if(li == NULL)  
        return 0;  
    if((*li) == NULL) //lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){
```

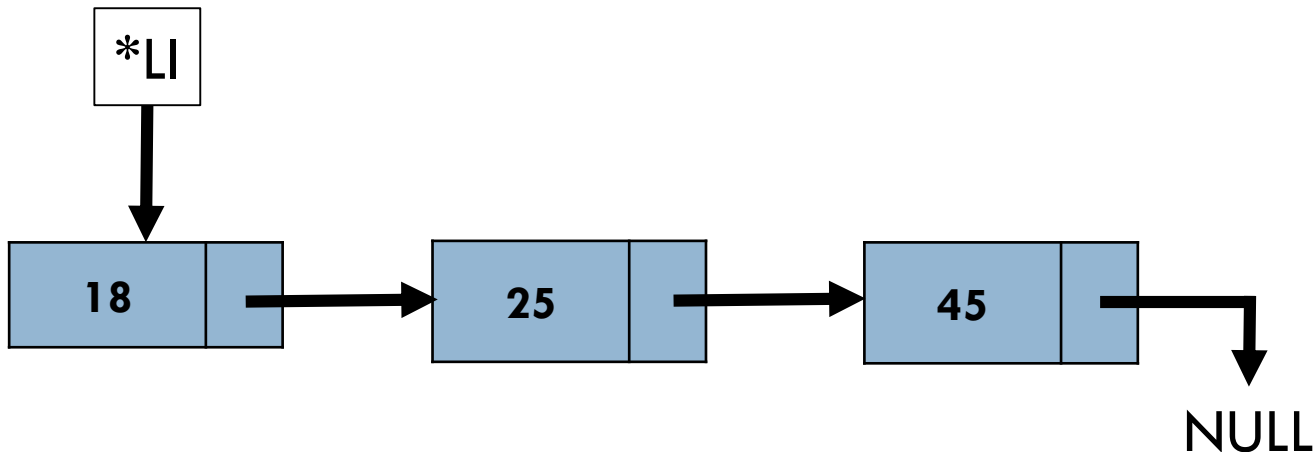


Lista Dinâmica: Remoção

153

18

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    → if((*li) == NULL) //lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){
```

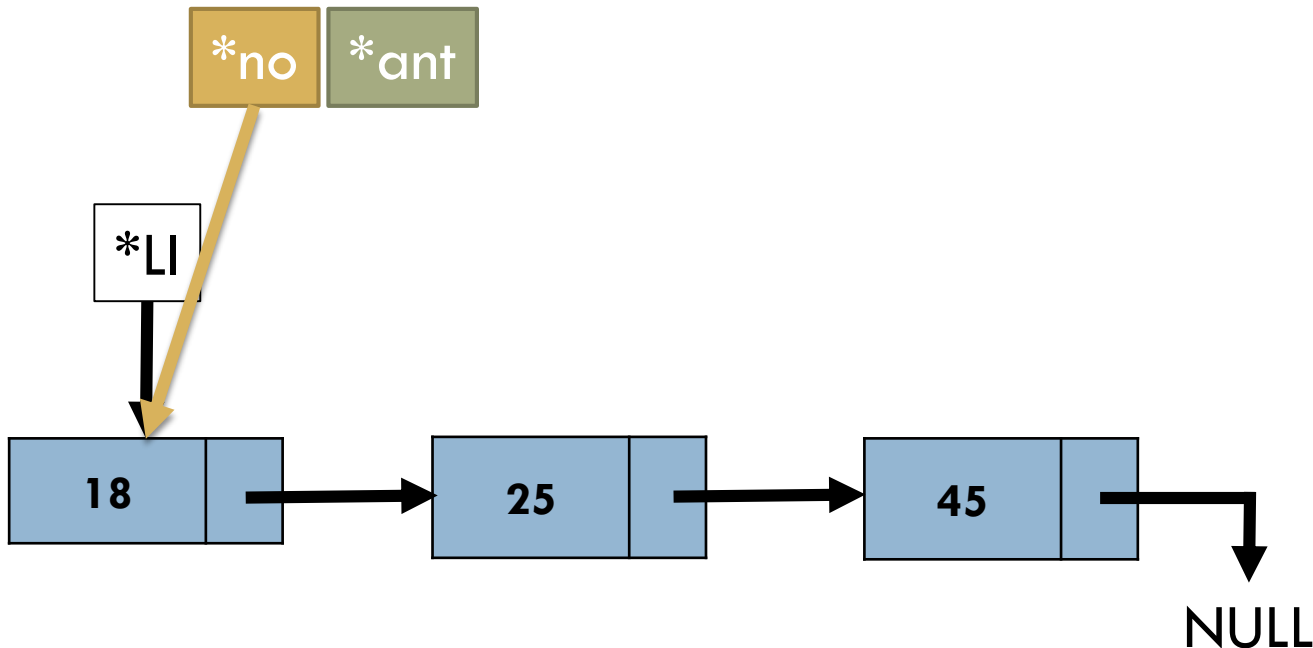


Lista Dinâmica: Remoção

154

18

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    → Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){
```

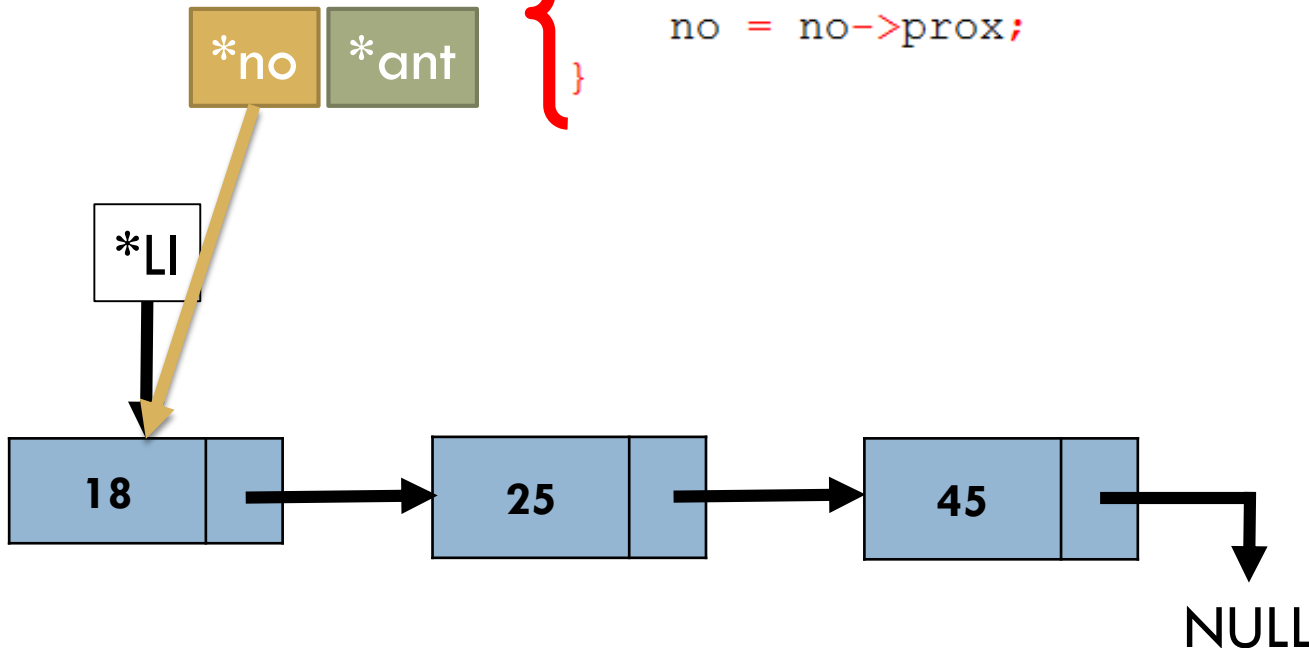


Lista Dinâmica: Remoção

155

18

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    → while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
}
```



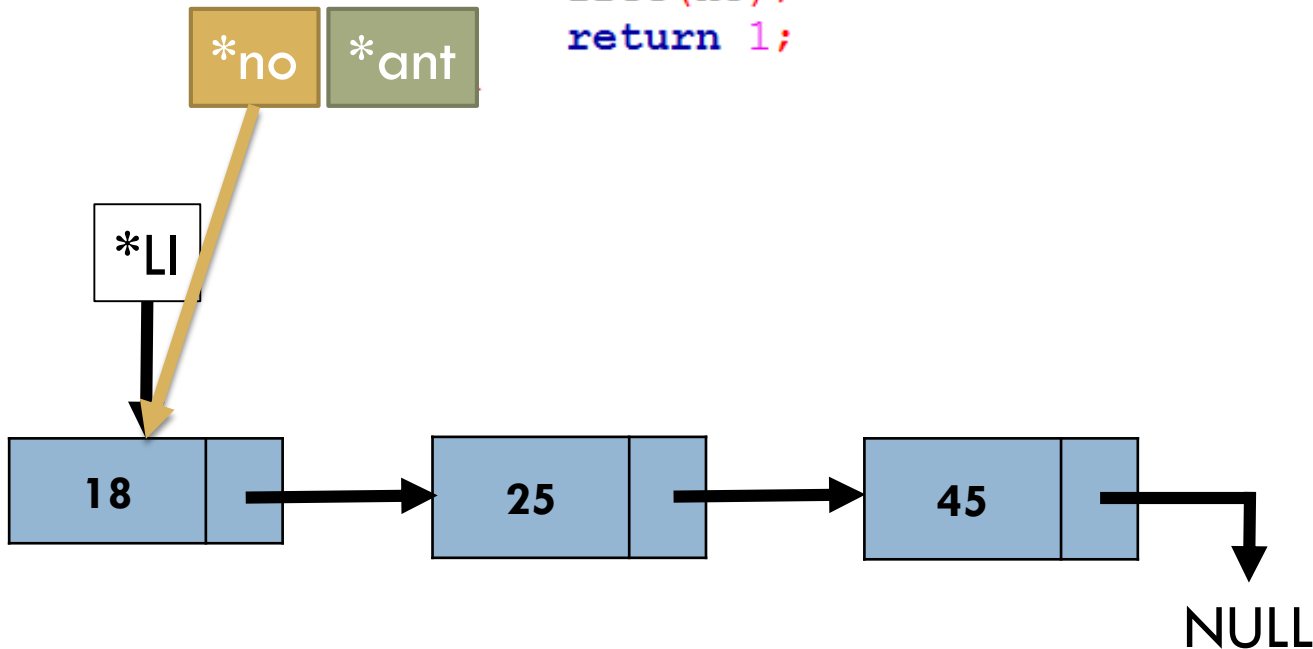
Lista Dinâmica: Remoção

156

18

```
→ if(no == NULL) // não encontrado  
    return 0;
```

```
if(no == *li) // remover o primeiro?  
    *li = no->prox;  
else  
    ant->prox = no->prox;  
free(no);  
return 1;
```



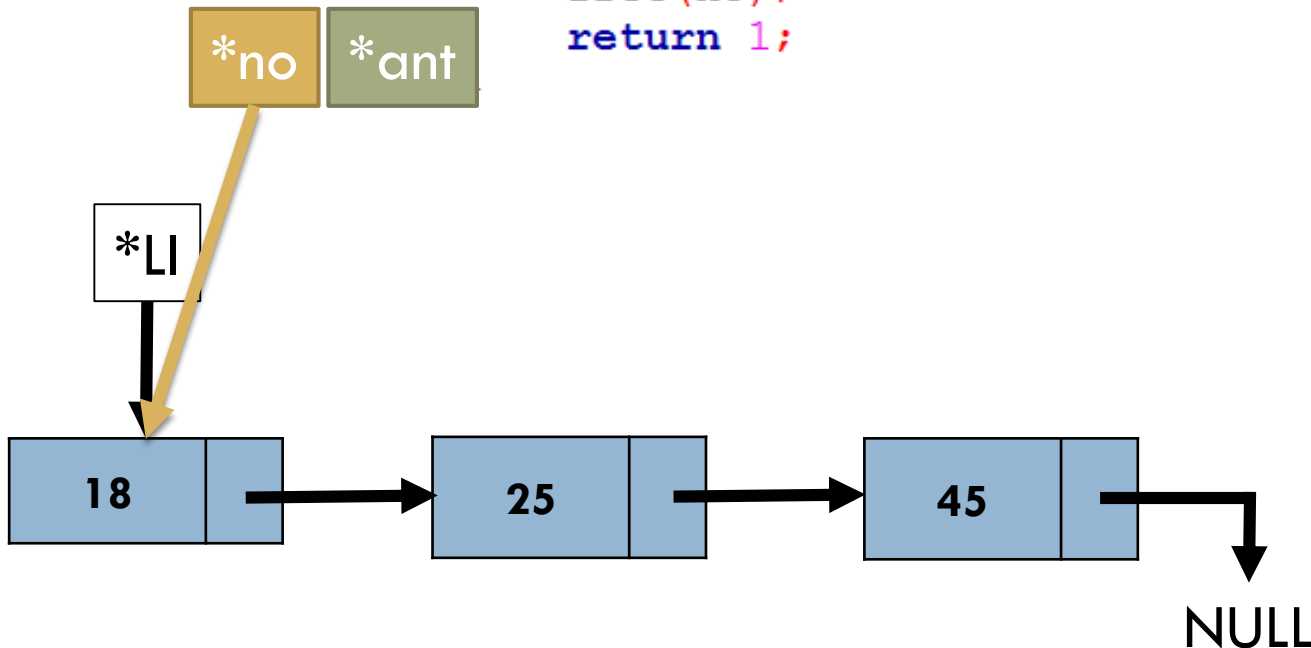
Lista Dinâmica: Remoção

157

18

```
if(no == NULL) //não encontrado  
    return 0;
```

```
→ if(no == *li) //remover o primeiro?  
    *li = no->prox;  
else  
    ant->prox = no->prox;  
free(no);  
return 1;
```



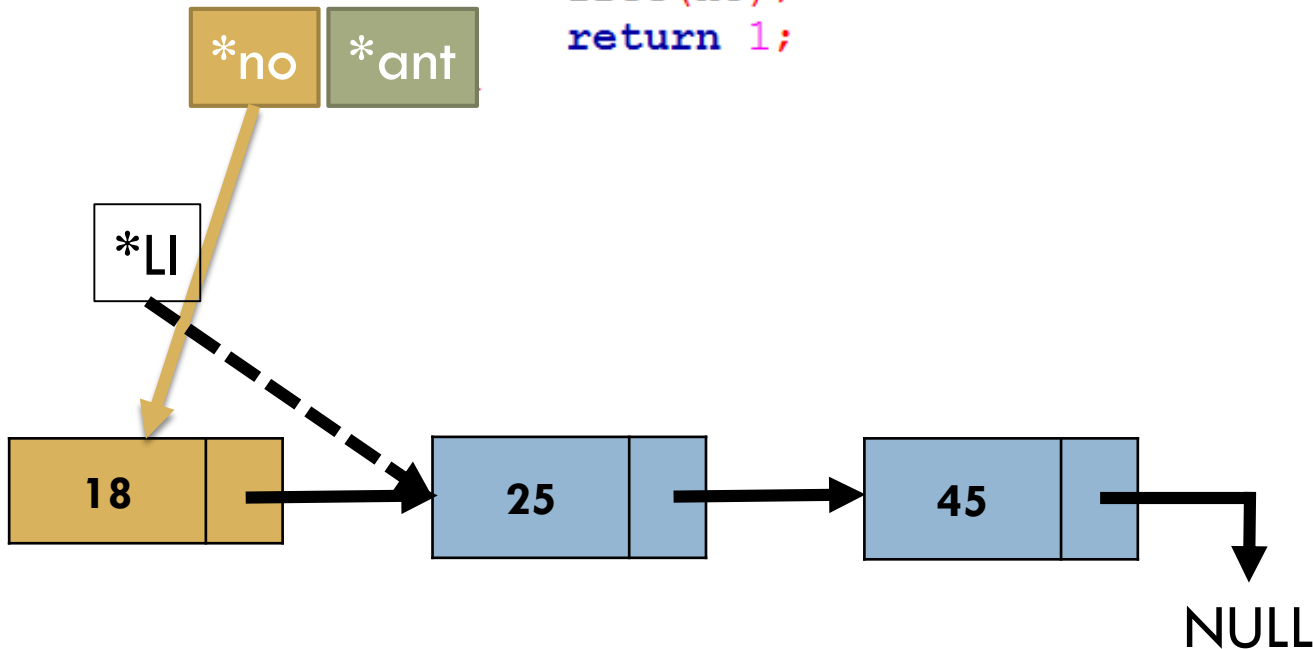
Lista Dinâmica: Remoção

158

18

```
if(no == NULL) //não encontrado  
    return 0;
```

```
if(no == *li) //remover o primeiro?  
    → *li = no->prox;  
else  
    ant->prox = no->prox;  
free(no);  
return 1;
```



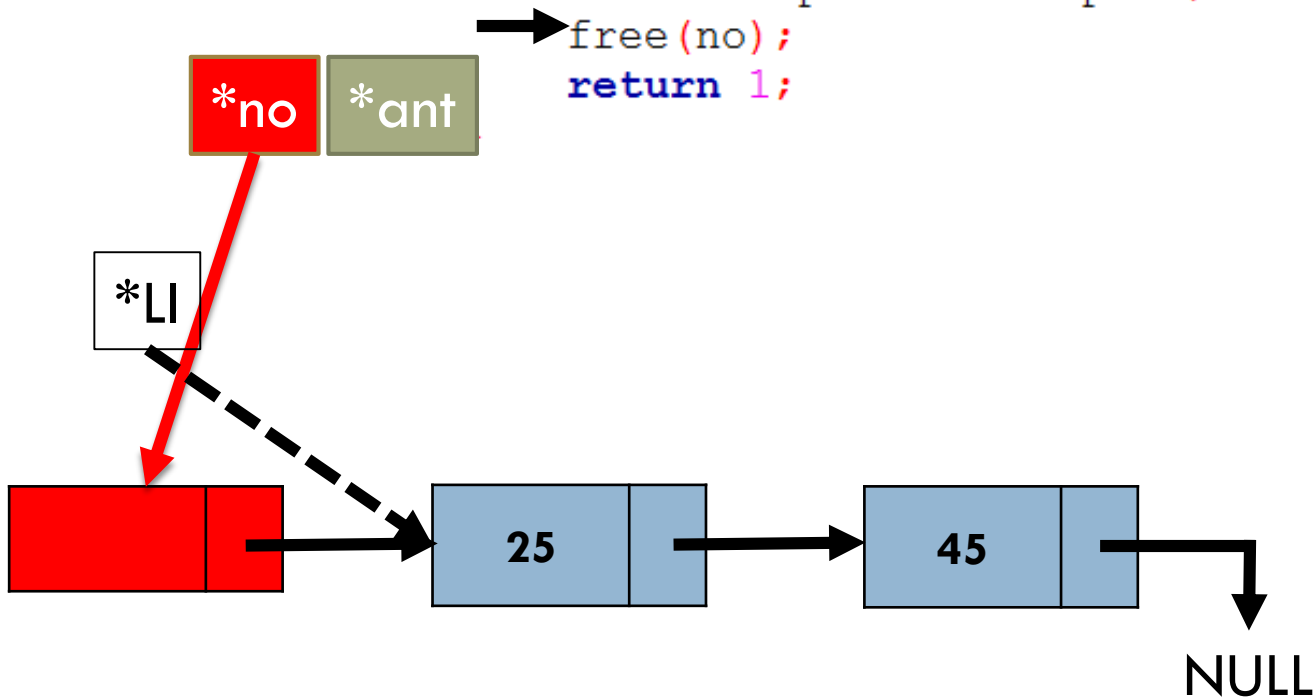
Lista Dinâmica: Remoção

159

18

```
if(no == NULL) //não encontrado
    return 0;

if(no == *li) //remover o primeiro?
    *li = no->prox;
else
    ant->prox = no->prox;
free(no);
return 1;
```



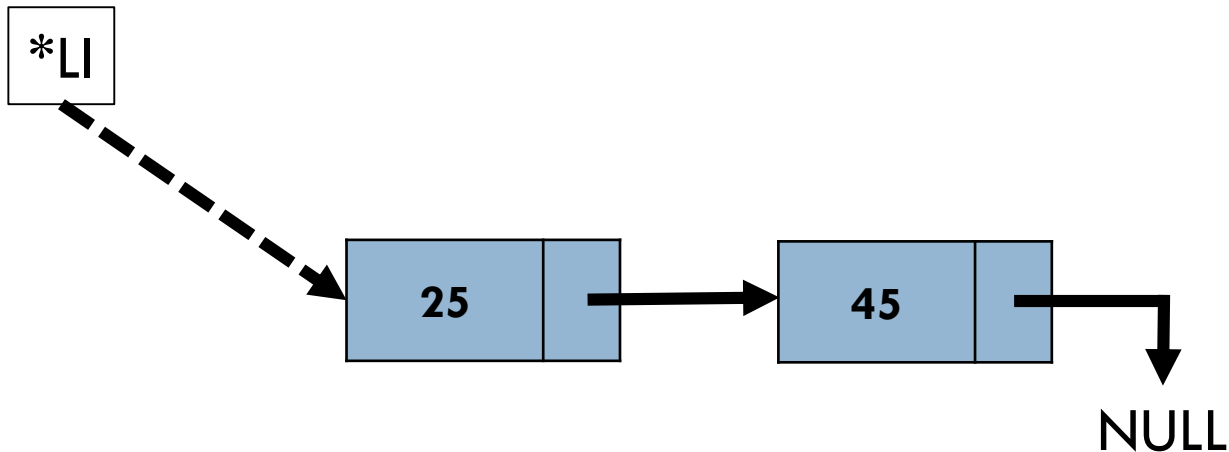
Lista Dinâmica: Remoção

160

18

```
if(no == NULL) //não encontrado  
    return  
  
if(no == *  
    *li =  
else  
    ant->prox = no->prox;  
    free(no);  
    return 1;
```

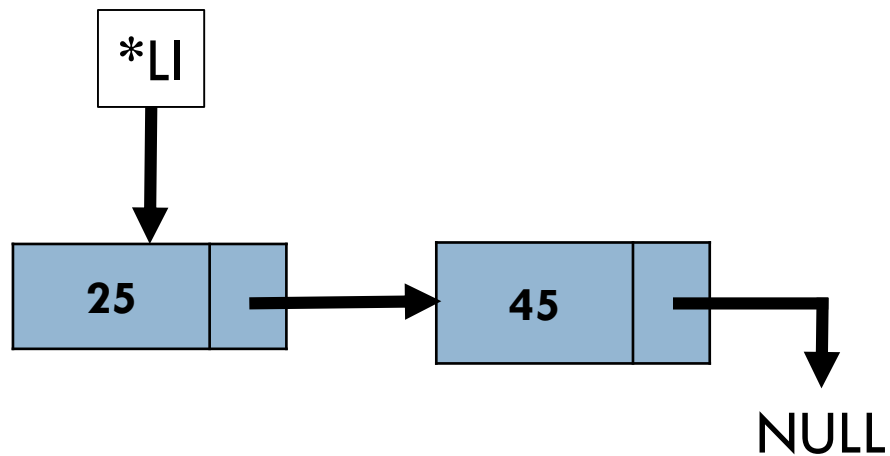
- RETORNA OK;
- FINAL DA REMOÇÃO DO ELEMENTO.



Lista Dinâmica: Remoção

161

REORGANIZANDO A IMAGEM

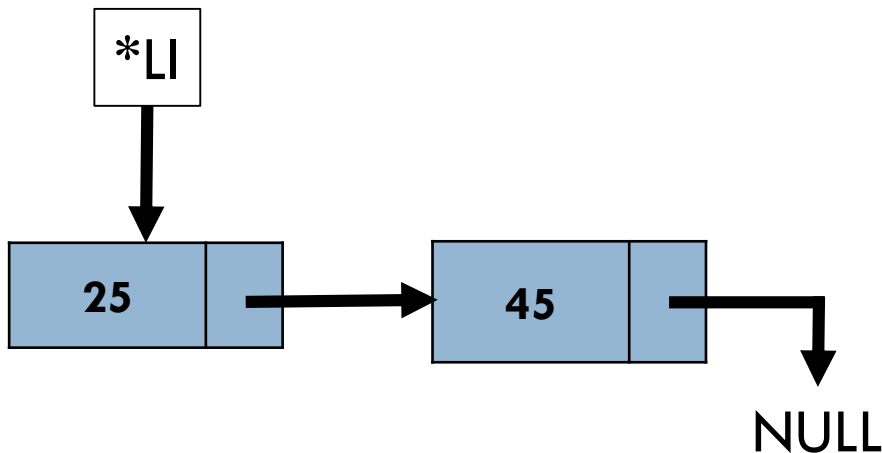


Lista Dinâmica: Remoção

162

3

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){
```

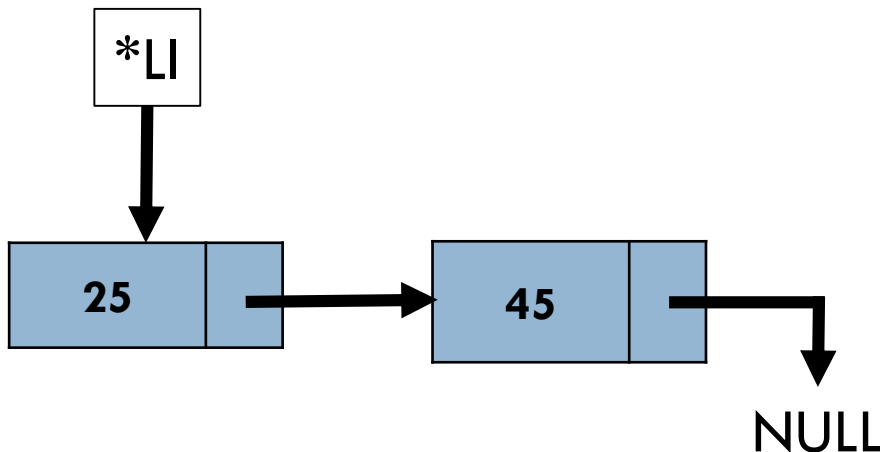


Lista Dinâmica: Remoção

163

3

```
int remove_lista(Lista* li, int mat){  
    → if(li == NULL)  
        return 0;  
    if((*li) == NULL) //lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){
```

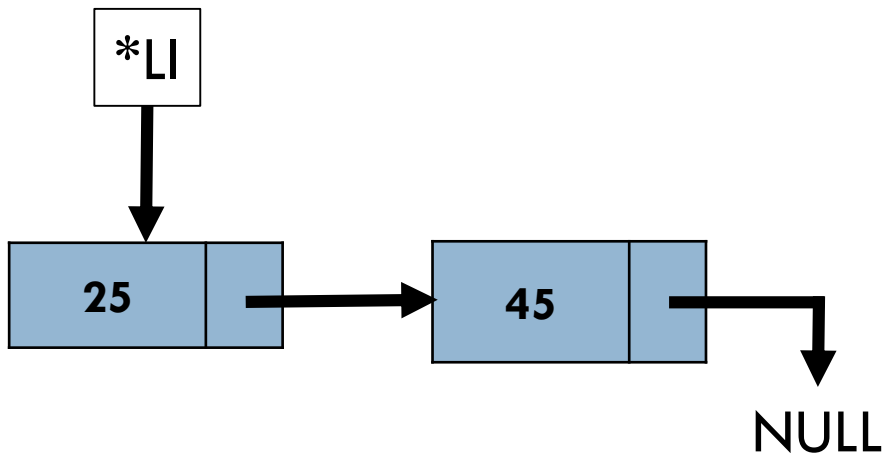


Lista Dinâmica: Remoção

164

3

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    → if((*li) == NULL) //lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){
```

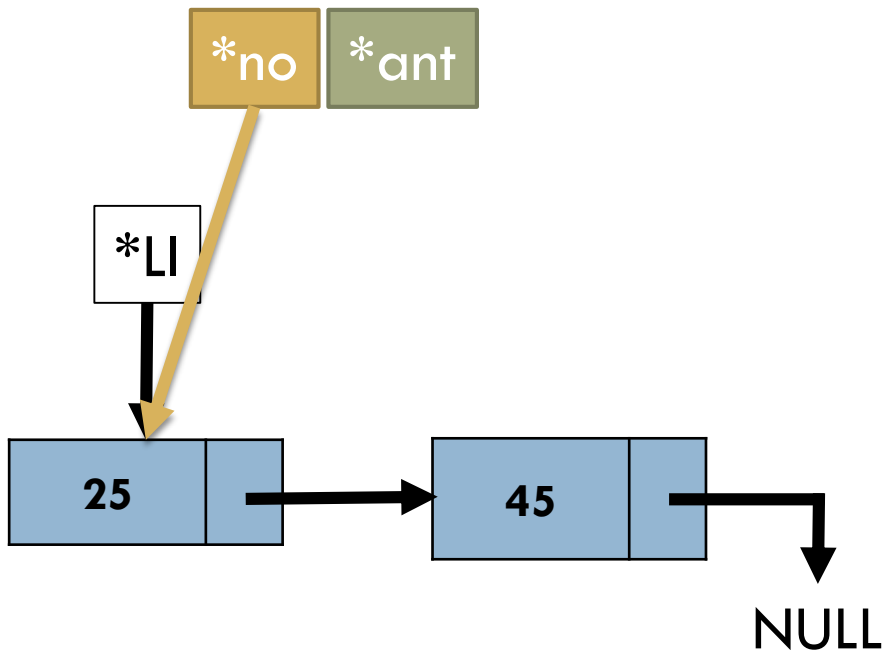


Lista Dinâmica: Remoção

165

3

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    → Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){
```

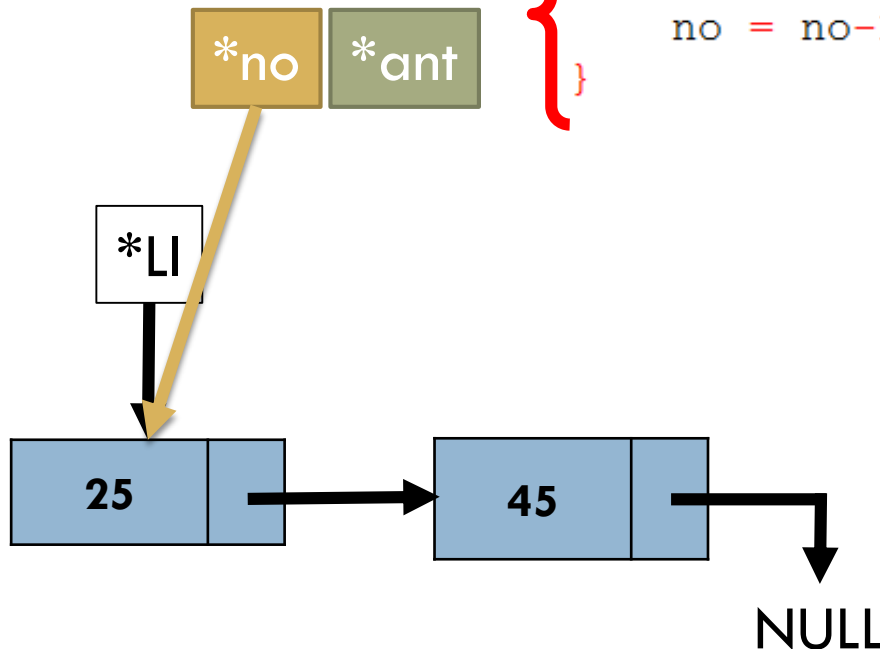


Lista Dinâmica: Remoção

166

3

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
}
```

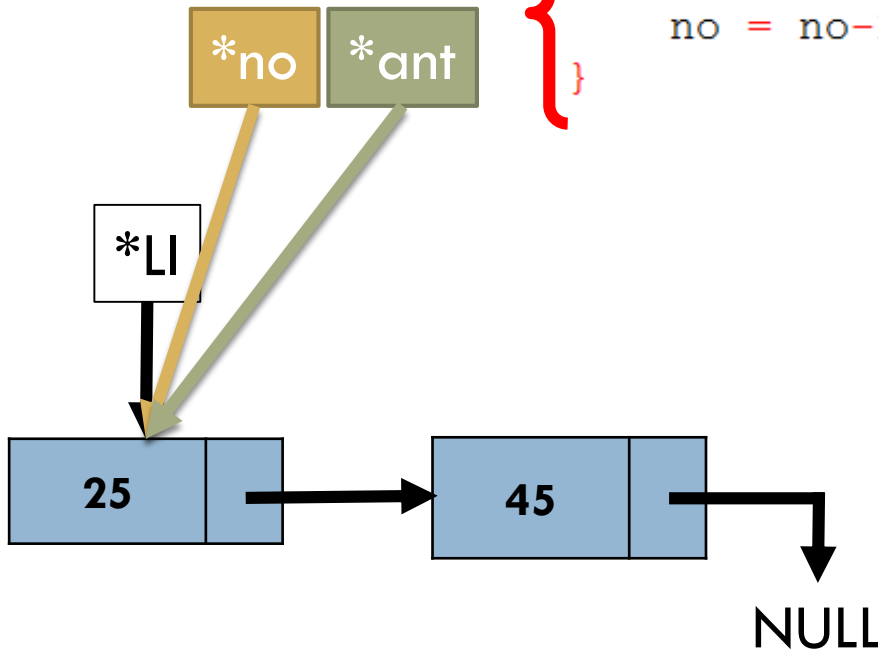


Lista Dinâmica: Remoção

167

3

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
}
```

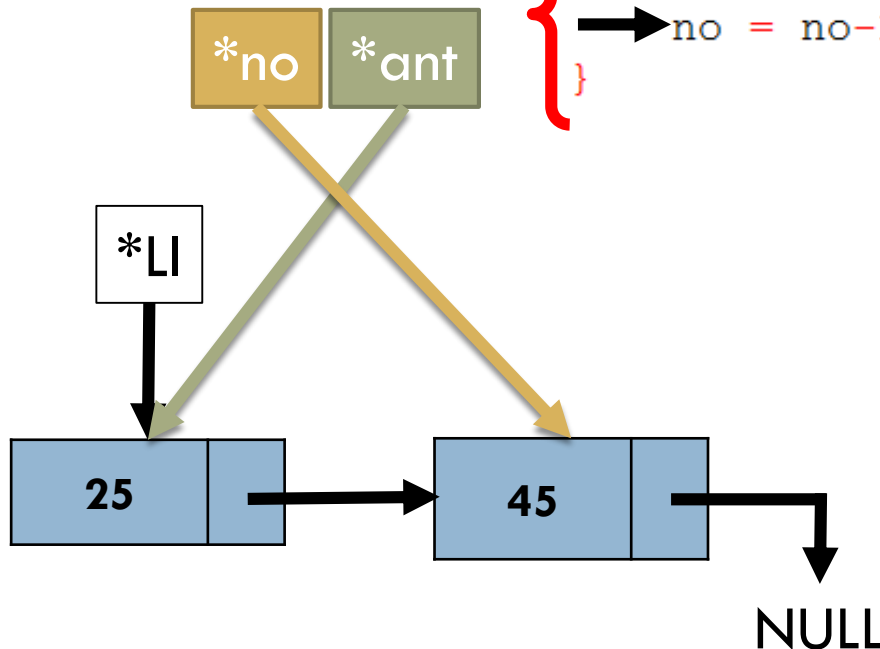


Lista Dinâmica: Remoção

168

3

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
}
```

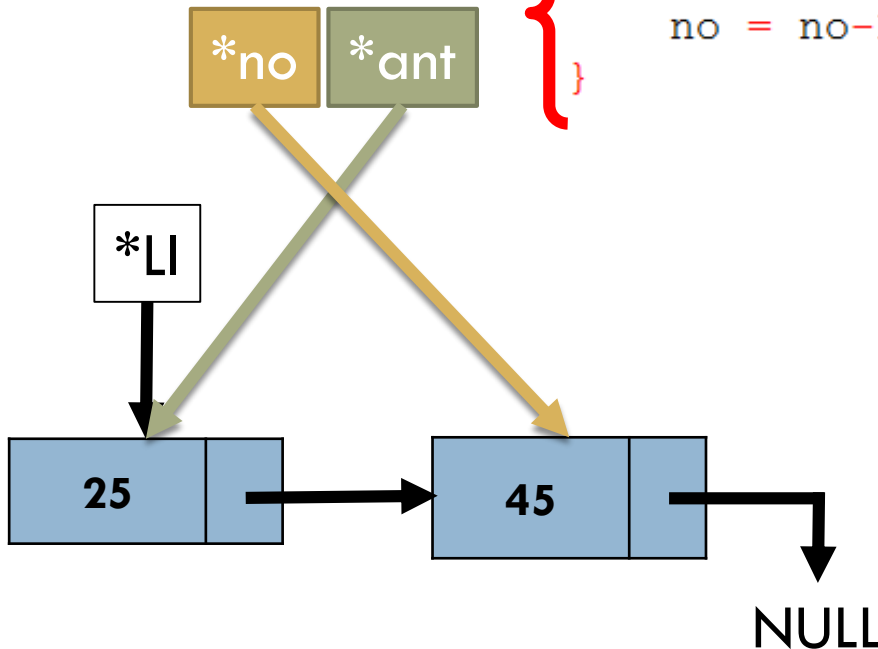


Lista Dinâmica: Remoção

169

3

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
}
```

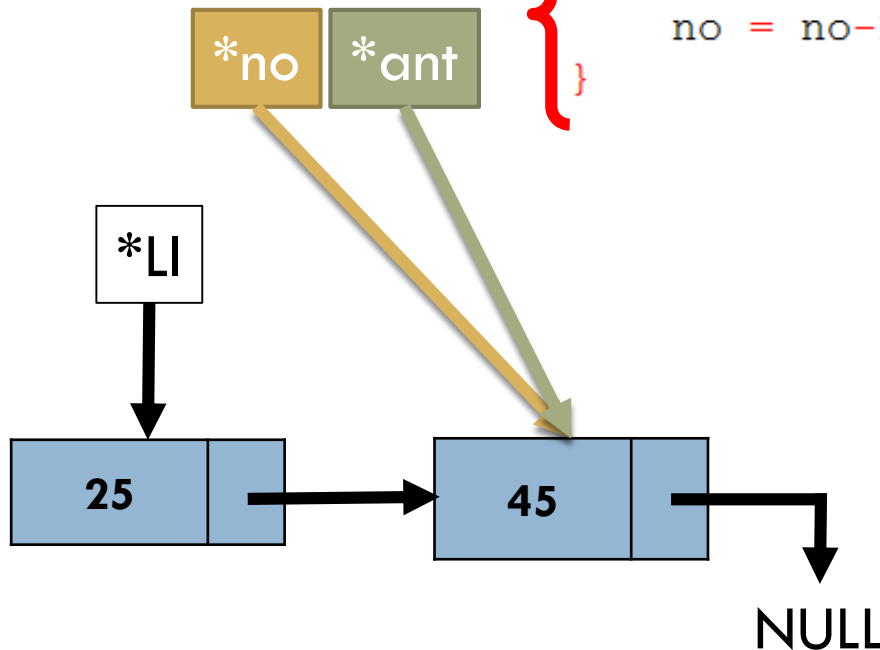


Lista Dinâmica: Remoção

170

3

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
}
```

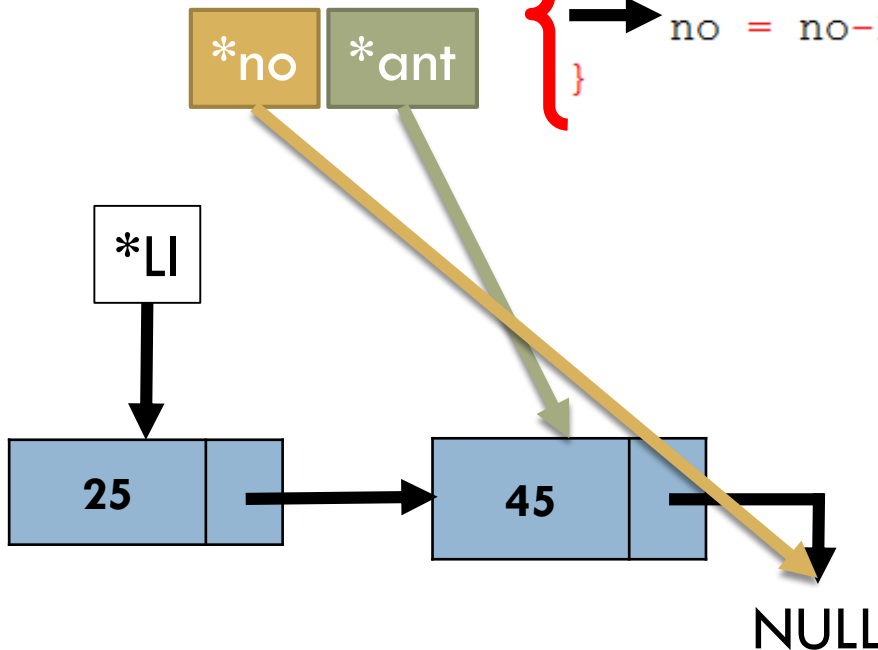


Lista Dinâmica: Remoção

171

3

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }
```

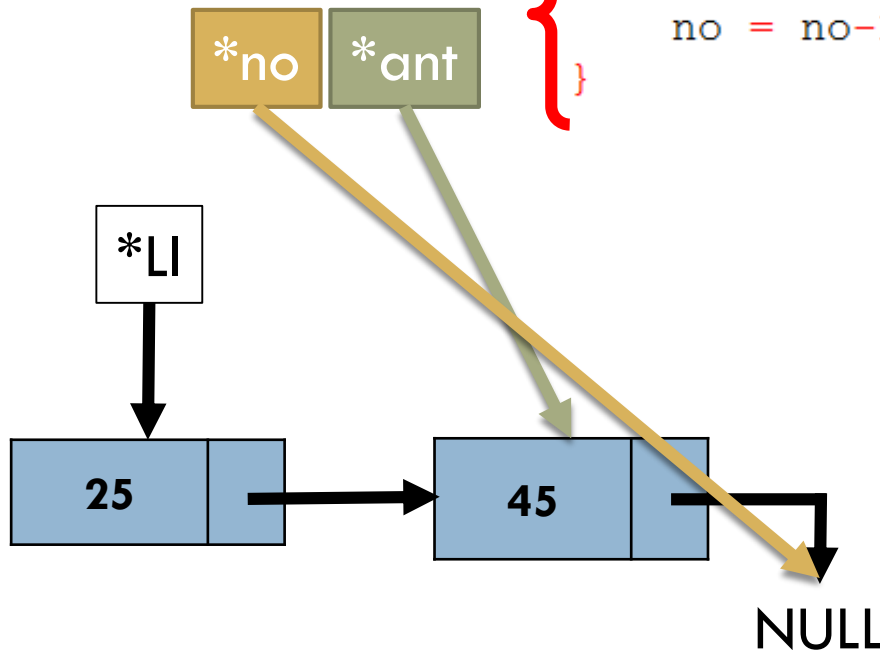


Lista Dinâmica: Remoção

172

3

```
int remove_lista(Lista* li, int mat){  
    if(li == NULL)  
        return 0;  
    if((*li) == NULL)//lista vazia  
        return 0;  
    Elem *ant, *no = *li;  
    while(no != NULL && no->dados.matricula != mat){  
        ant = no;  
        no = no->prox;  
    }  
}
```



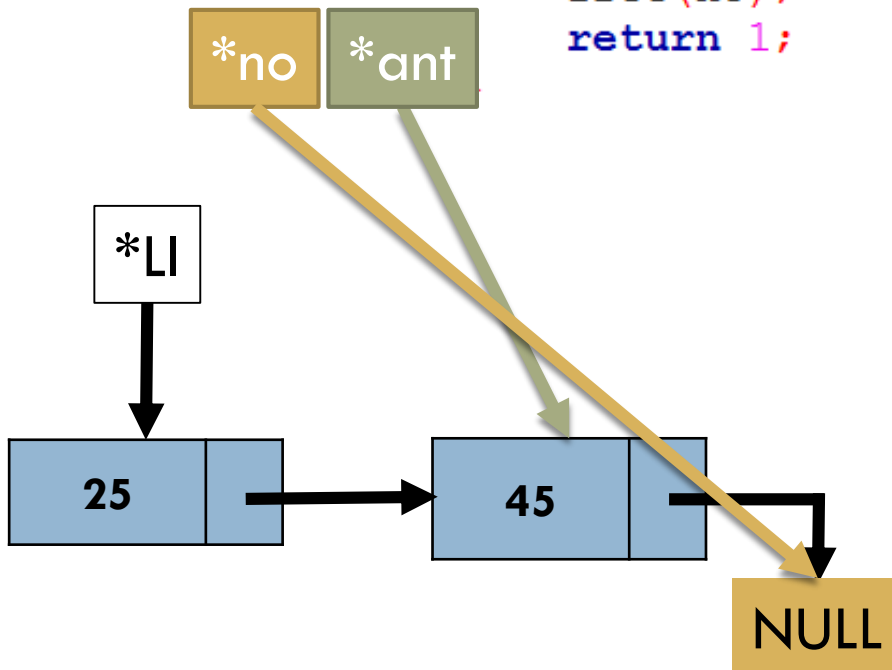
Lista Dinâmica: Remoção

173

3

```
→ if(no == NULL) // não encontrado  
    return 0;
```

```
if(no == *li) // remover o primeiro?  
    *li = no->prox;  
else  
    ant->prox = no->prox;  
free(no);  
return 1;
```



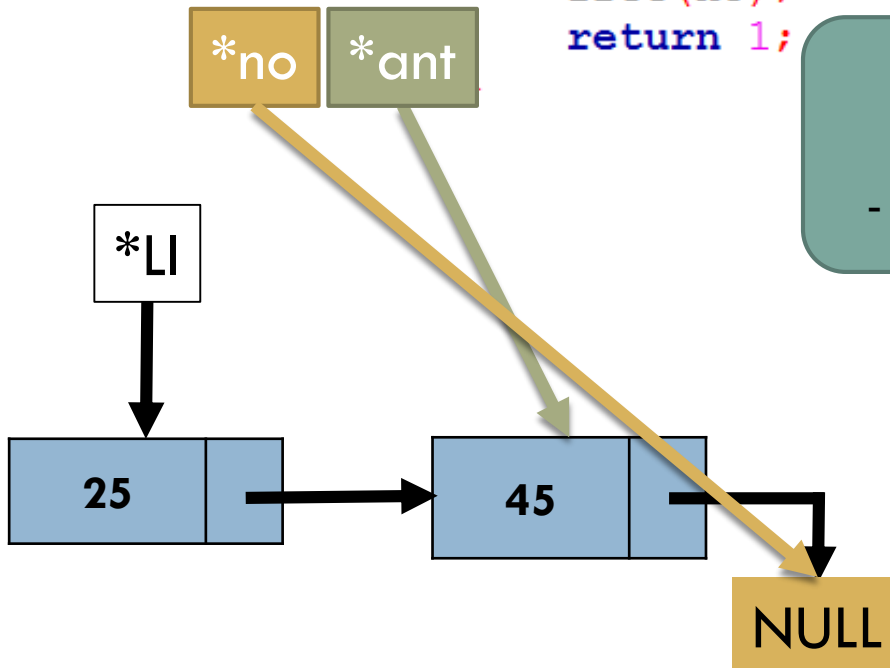
Lista Dinâmica: Remoção

174

3

```
if(no == NULL) //não encontrado  
    → return 0;  
  
if(no == *li) //remover o primeiro?  
    *li = no->prox;  
else  
    ant->prox = no->prox;  
free(no);  
return 1;
```

- ELEMENTO NÃO ENCONTRADO;
- RETORNA ERRO;
- FINAL DA OPERAÇÃO DE REMOÇÃO.



Lista Dinâmica: Impressão

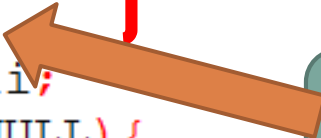
```
void imprime_lista(Lista* li){
    if(li == NULL)
        return;
    Elem* no = *li;
    while(no != NULL){
        printf("Matricula: %d\n", no->dados.matricula);
        printf("Nome: %s\n", no->dados.nome);
        printf("Notas: %f %f %f\n", no->dados.n1,
                                                    no->dados.n2,
                                                    no->dados.n3);

        printf("-----\n");

        no = no->prox;
    }
}
```

Lista Dinâmica: Impressão

```
void imprime_lista(Lista* li){  
    { if(li == NULL)  
        return;  
    Elem* no = *li;  
    while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
                                           no->dados.n2,  
                                           no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



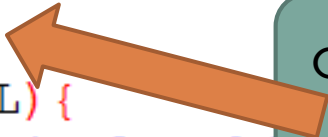
VERIFICAR SE A LISTA FOI ALOCADA

Lista Dinâmica: Impressão

```
void imprime_lista(Lista* li){
    if(li == NULL)
        return;
    Elem* no = *li;
    while(no != NULL){
        printf("Matricula: %d\n", no->dados.matricula);
        printf("Nome: %s\n", no->dados.nome);
        printf("Notas: %f %f %f\n", no->dados.n1,
                                                no->dados.n2,
                                                no->dados.n3);

        printf("-----\n");

        no = no->prox;
    }
}
```



CRIAR AUXILIAR PARA MANIPULAR PONTEIROS

Lista Dinâmica: Impressão

```
void imprime_lista(Lista* li)
{
    if(li == NULL)
        return;
    Elem* no = *li;
    while(no != NULL) {
        printf("Matricula: %d\n", no->dados.matricula);
        printf("Nome: %s\n", no->dados.nome);
        printf("Notas: %f %f %f\n", no->dados.n1,
                                                no->dados.n2,
                                                no->dados.n3);
        printf("-----\n");

        no = no->prox;
    }
}
```

VARRER TODOS OS ELEMENTOS
DA LISTA, IMPRIMINDO TODOS OS
CAMPOS DE CADA UM

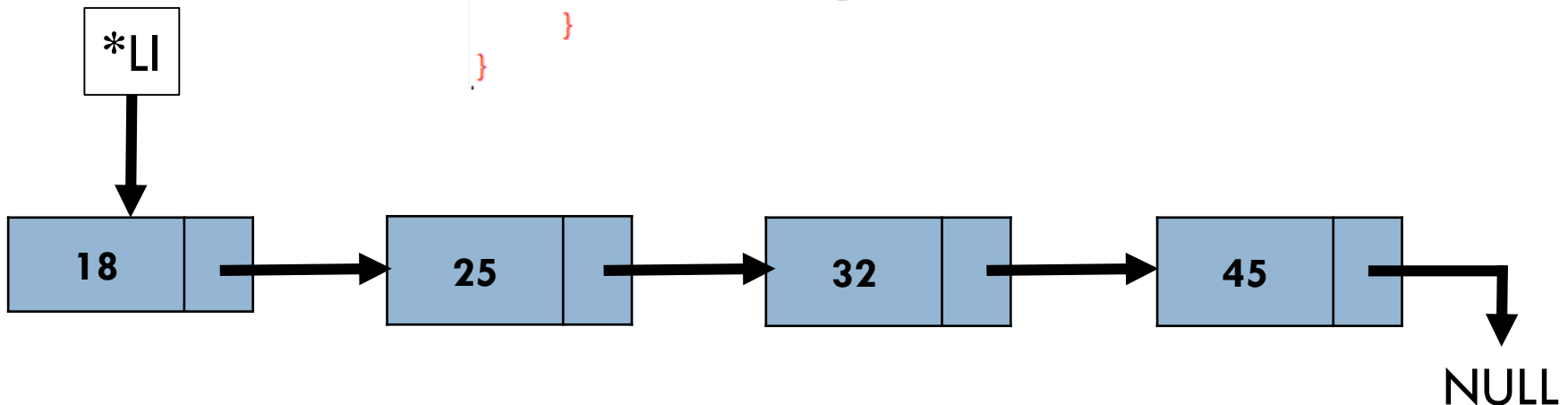
Lista Dinâmica: Impressão

179

```
void imprime_lista(Lista* li){
    if(li == NULL)
        return;
    Elem* no = *li;
    while(no != NULL){
        printf("Matricula: %d\n", no->dados.matricula);
        printf("Nome: %s\n", no->dados.nome);
        printf("Notas: %f %f %f\n", no->dados.n1,
                                                    no->dados.n2,
                                                    no->dados.n3);

        printf("-----\n");

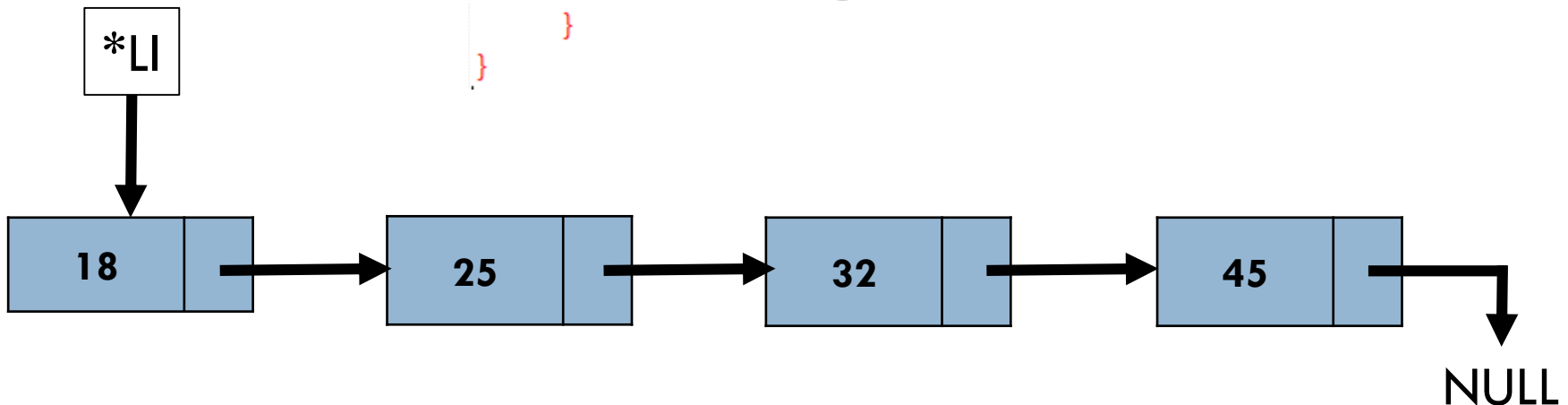
        no = no->prox;
    }
}
```



Lista Dinâmica: Impressão

180

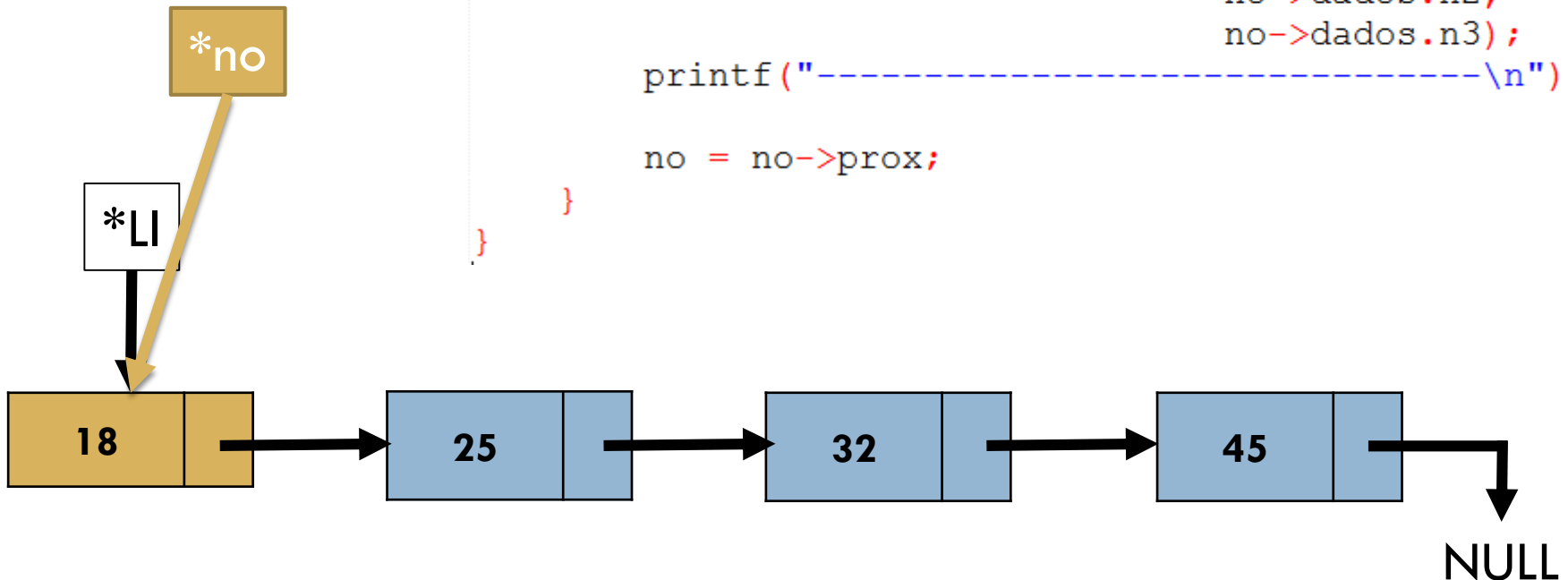
```
void imprime_lista(Lista* li){  
    → if(li == NULL)  
        return;  
    Elem* no = *li;  
    while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
                                                    no->dados.n2,  
                                                    no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



Lista Dinâmica: Impressão

181

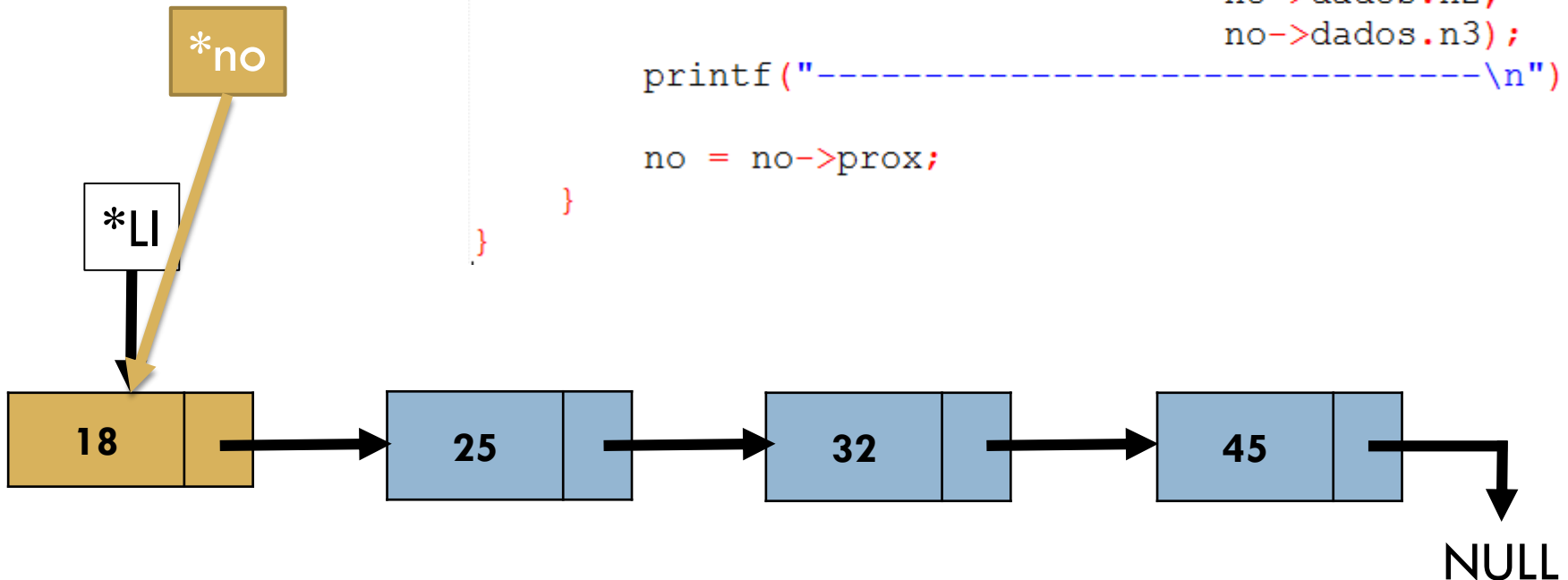
```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    → Elem* no = *li;  
    while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
                                           no->dados.n2,  
                                           no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



Lista Dinâmica: Impressão

182

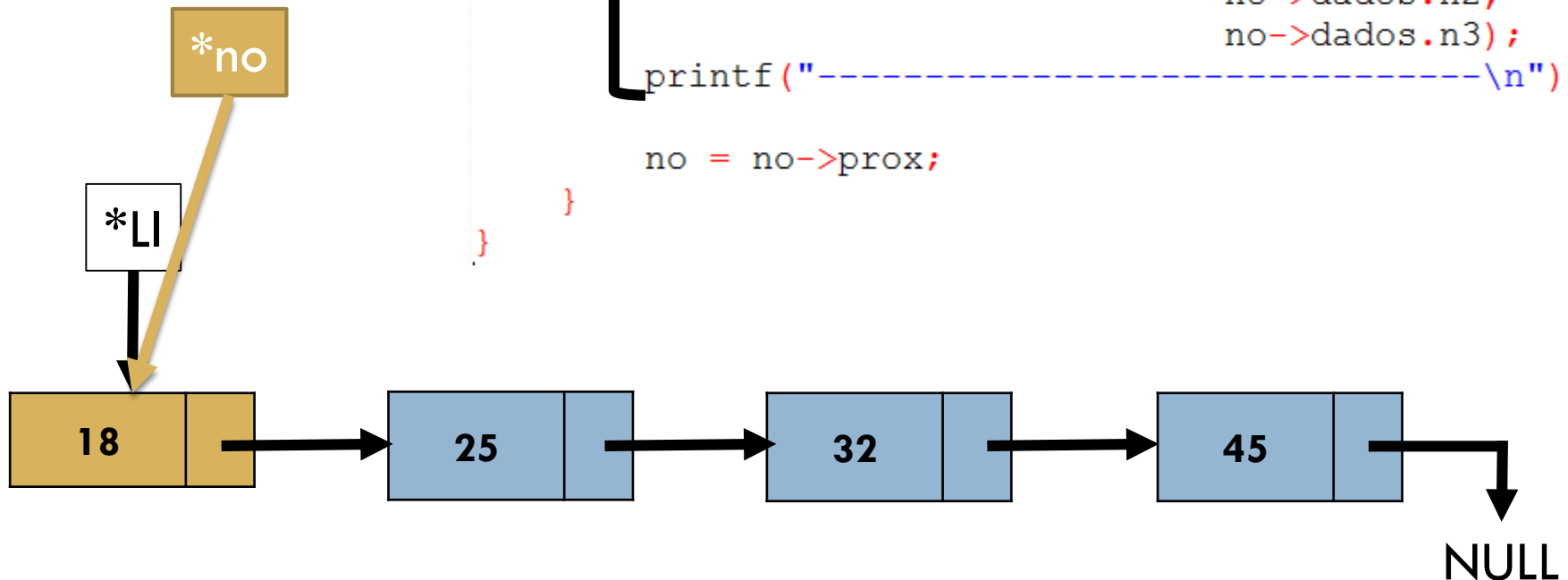
```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    Elem* no = *li;  
    → while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
                                                    no->dados.n2,  
                                                    no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



Lista Dinâmica: Impressão

183

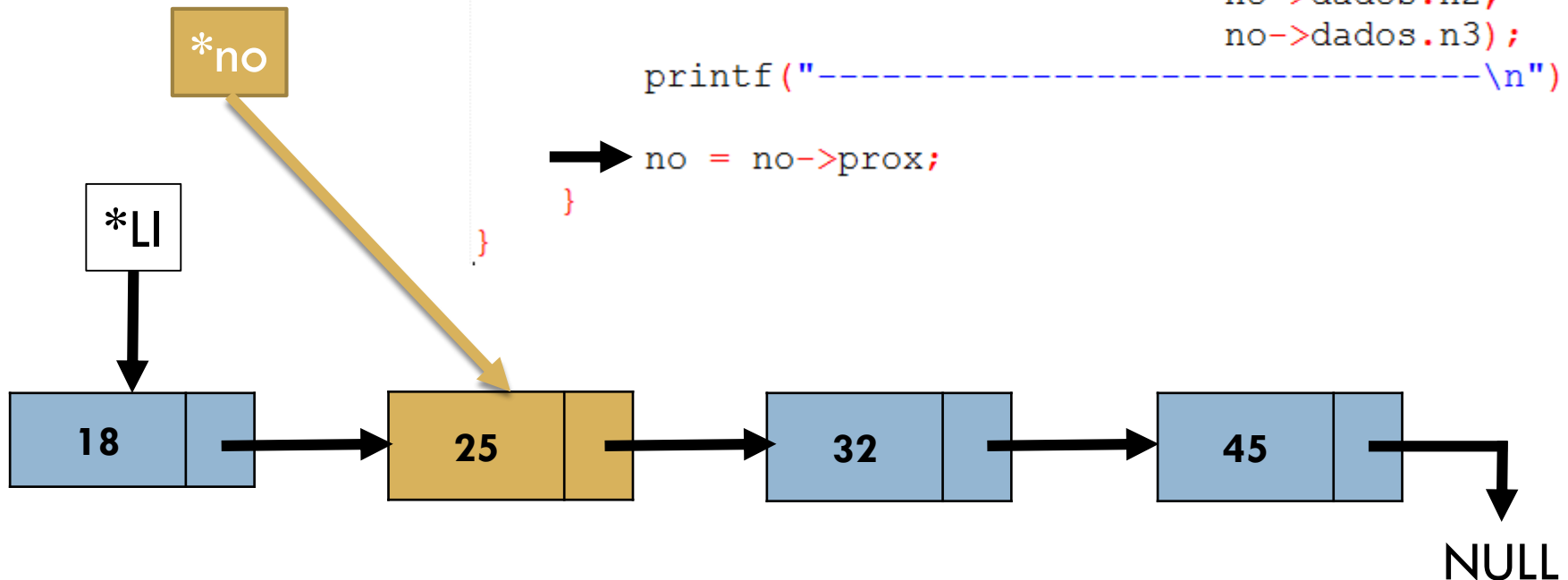
```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    Elem* no = *li;  
    while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
            no->dados.n2,  
            no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



Lista Dinâmica: Impressão

184

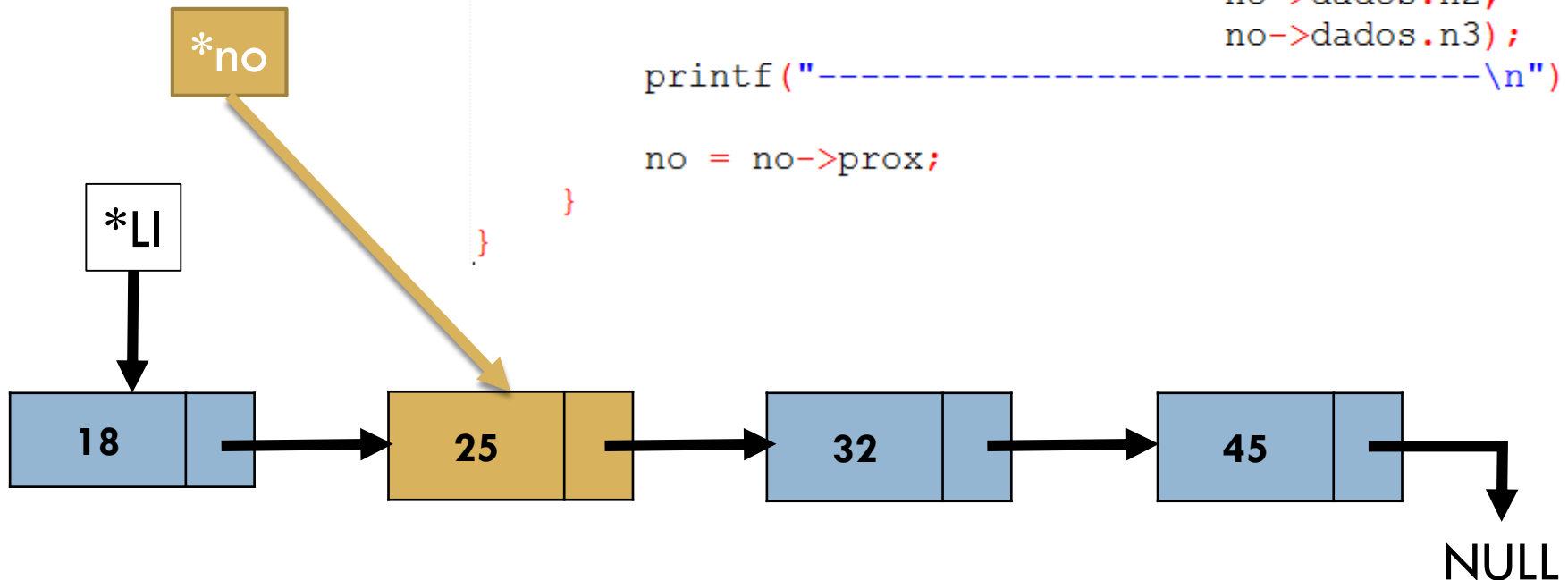
```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    Elem* no = *li;  
    while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
            no->dados.n2,  
            no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



Lista Dinâmica: Impressão

185

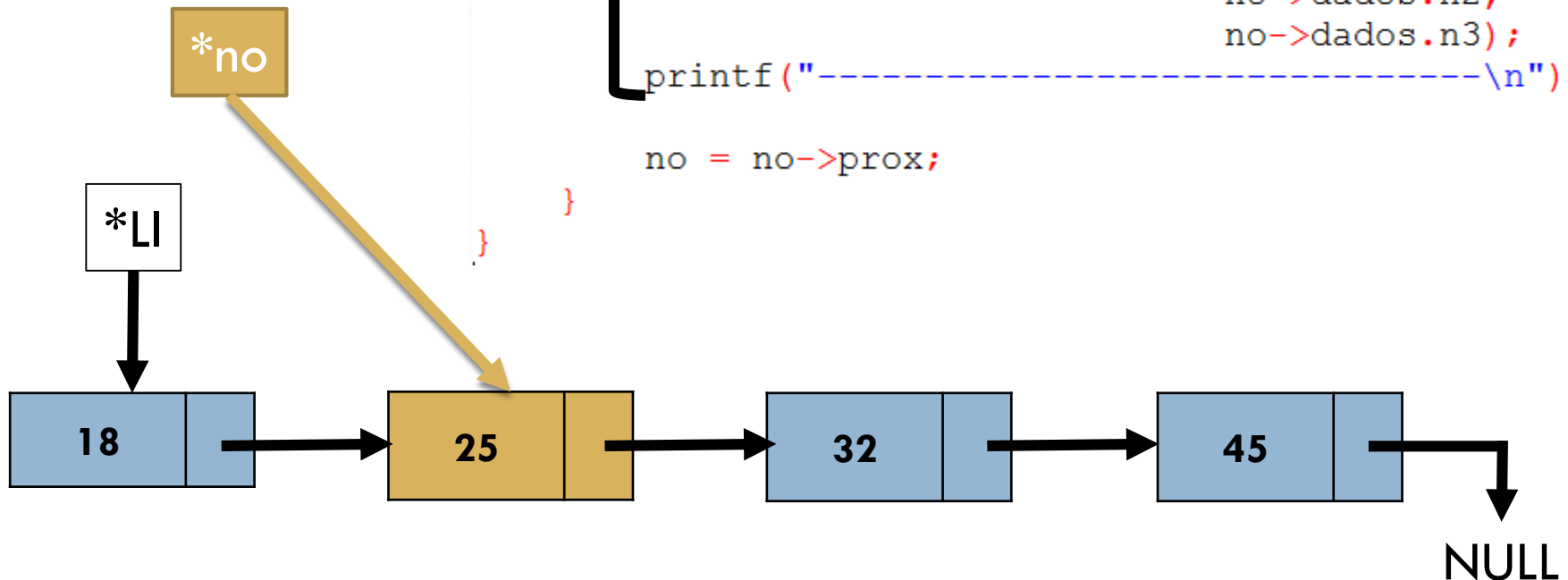
```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    Elem* no = *li;  
    → while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
                                                    no->dados.n2,  
                                                    no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



Lista Dinâmica: Impressão

186

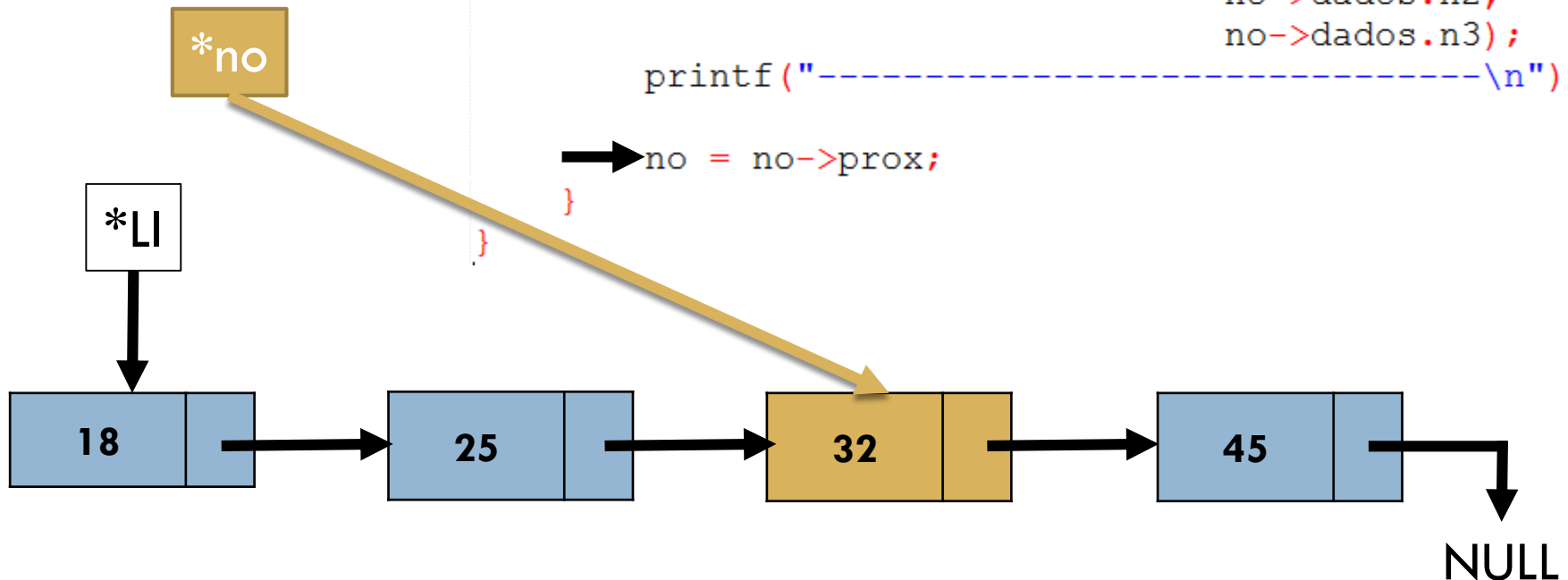
```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    Elem* no = *li;  
    while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
            no->dados.n2,  
            no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



Lista Dinâmica: Impressão

187

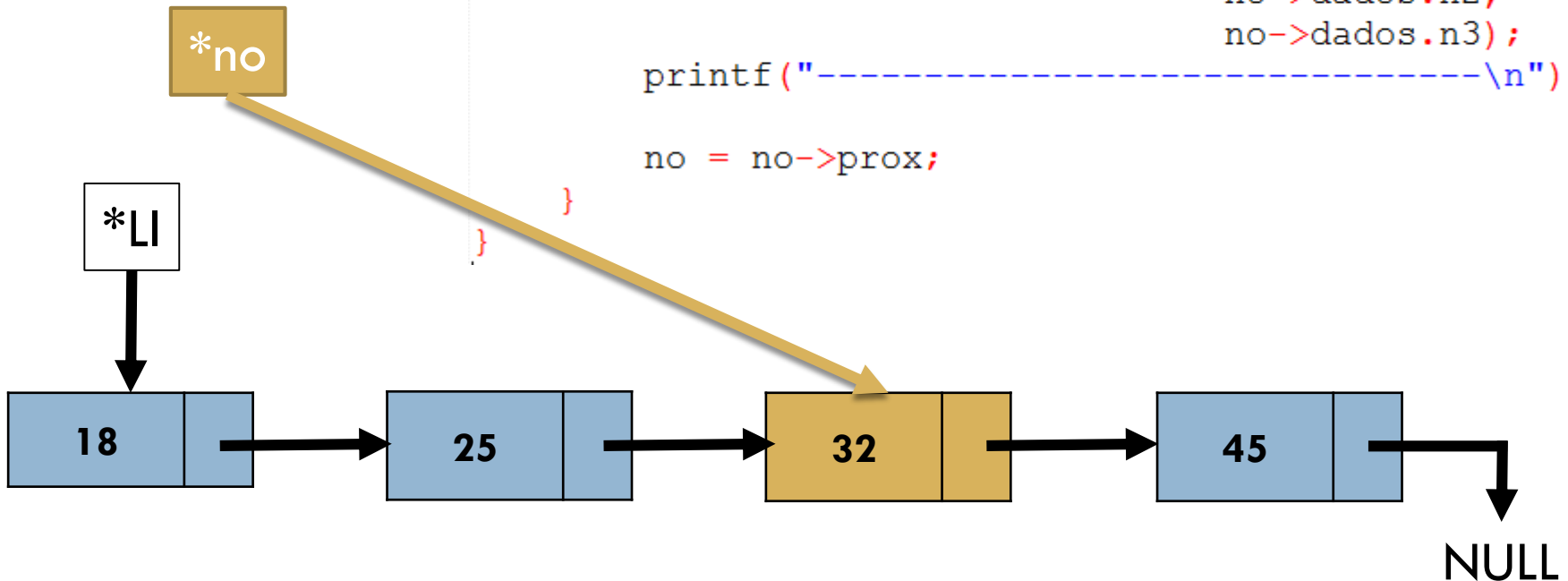
```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    Elem* no = *li;  
    while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
            no->dados.n2,  
            no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



Lista Dinâmica: Impressão

188

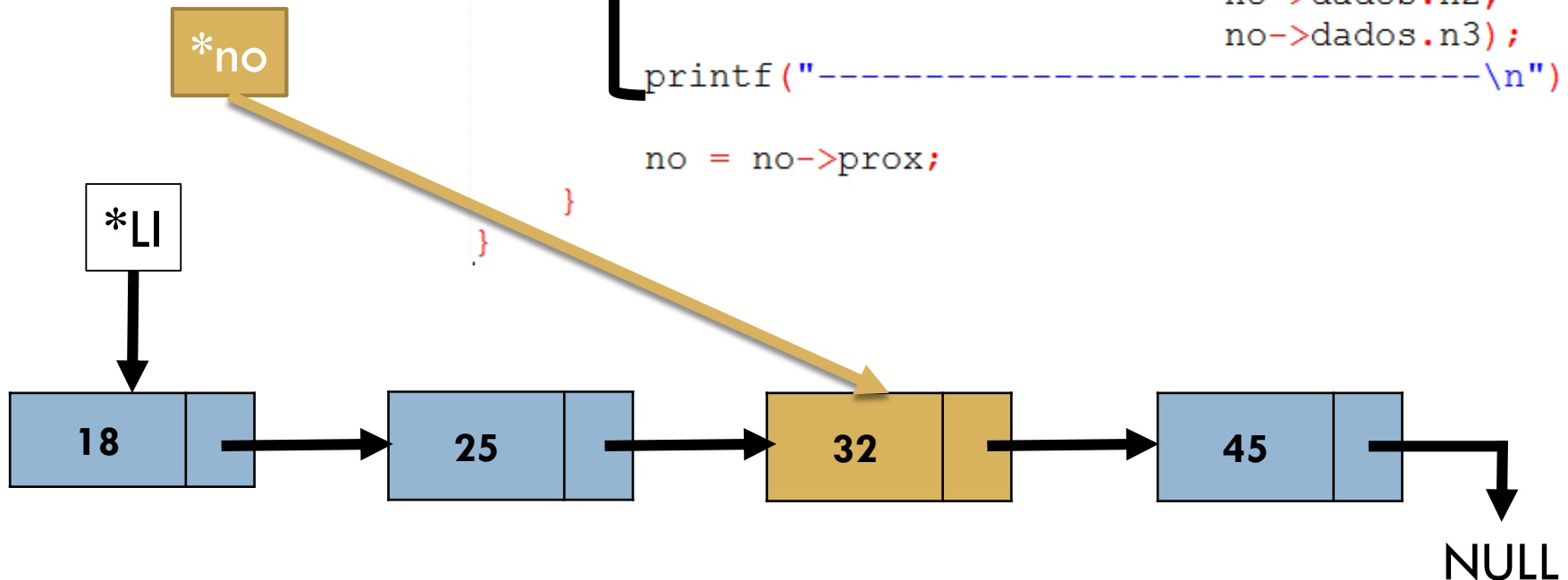
```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    Elem* no = *li;  
    → while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
                                                    no->dados.n2,  
                                                    no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



Lista Dinâmica: Impressão

189

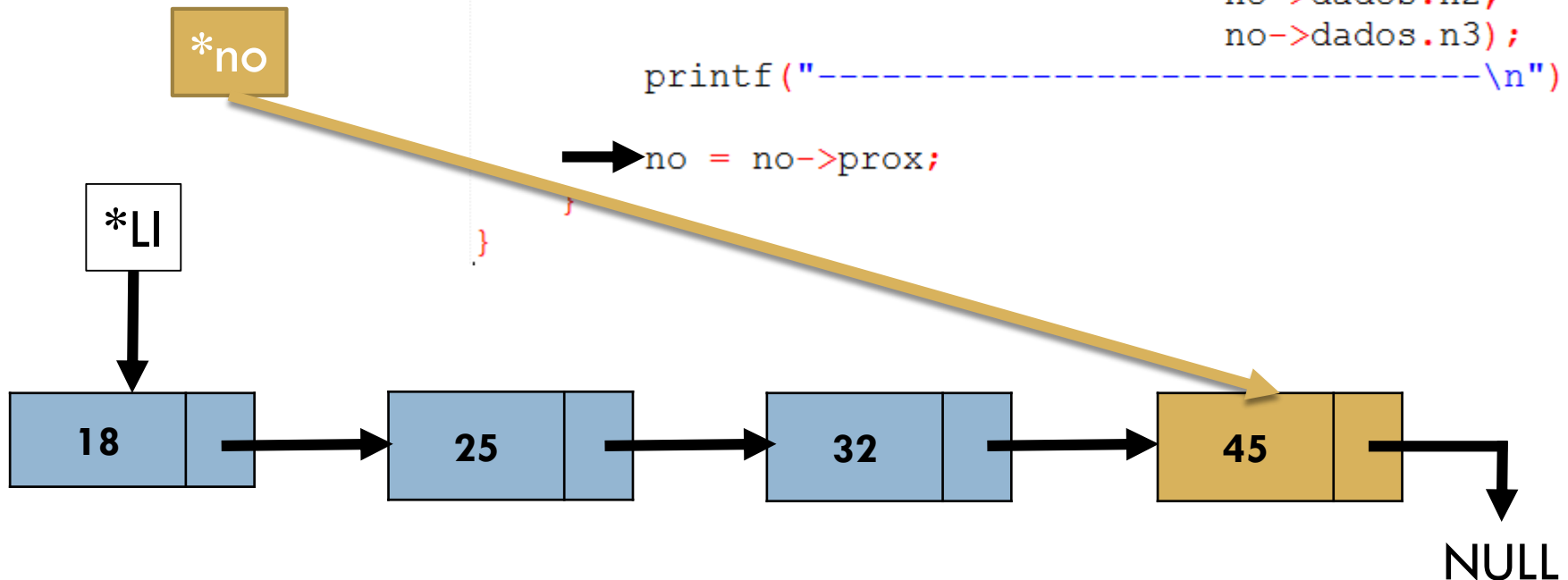
```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    Elem* no = *li;  
    while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
            no->dados.n2,  
            no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



Lista Dinâmica: Impressão

190

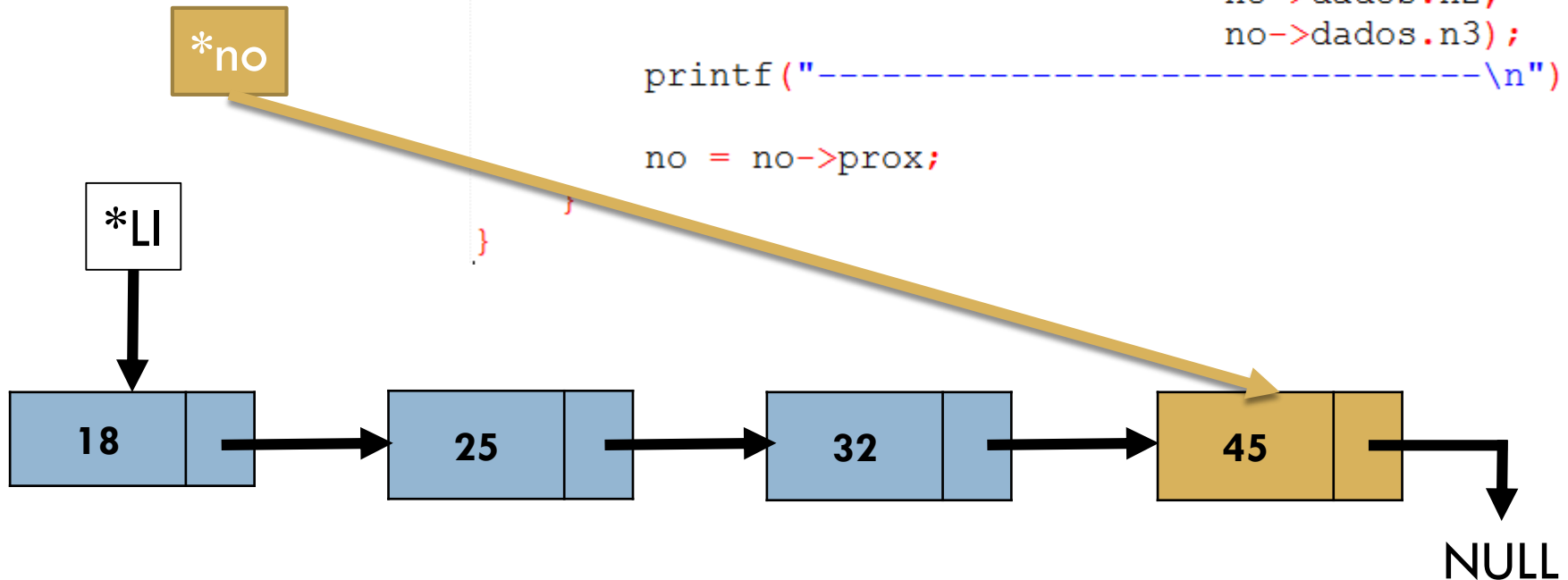
```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    Elem* no = *li;  
    while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
            no->dados.n2,  
            no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



Lista Dinâmica: Impressão

191

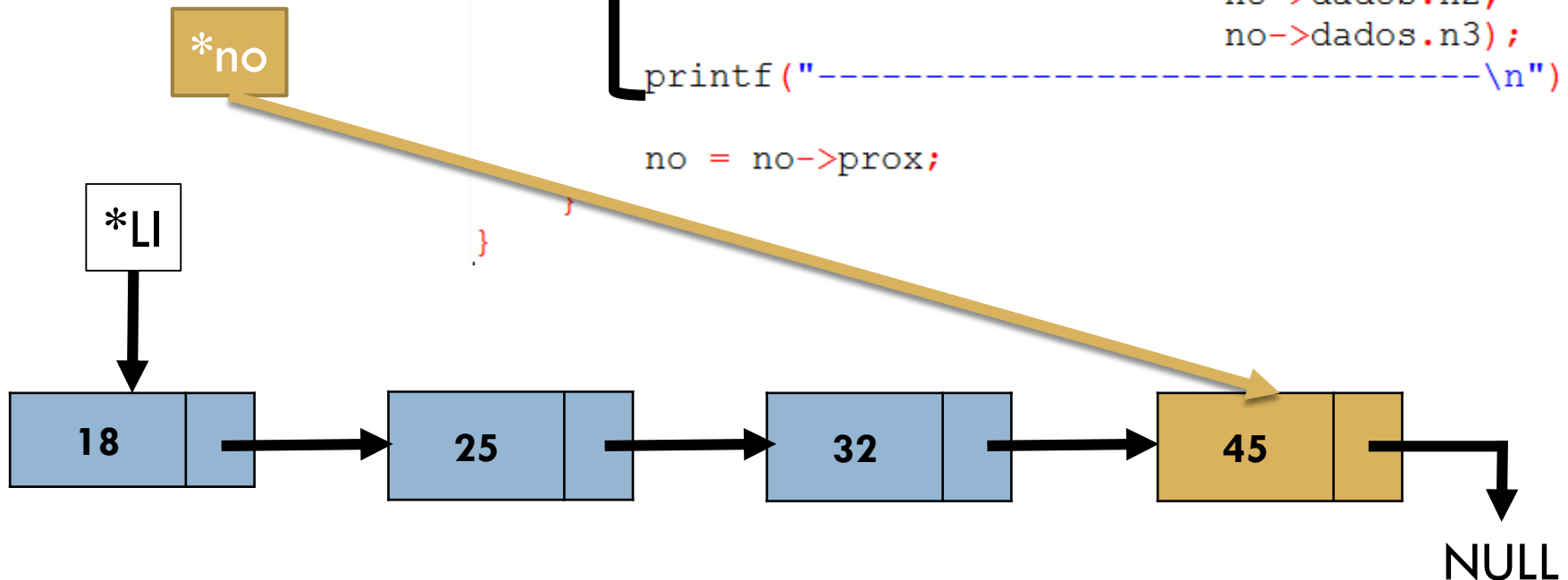
```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    Elem* no = *li;  
    → while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
            no->dados.n2,  
            no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



Lista Dinâmica: Impressão

192

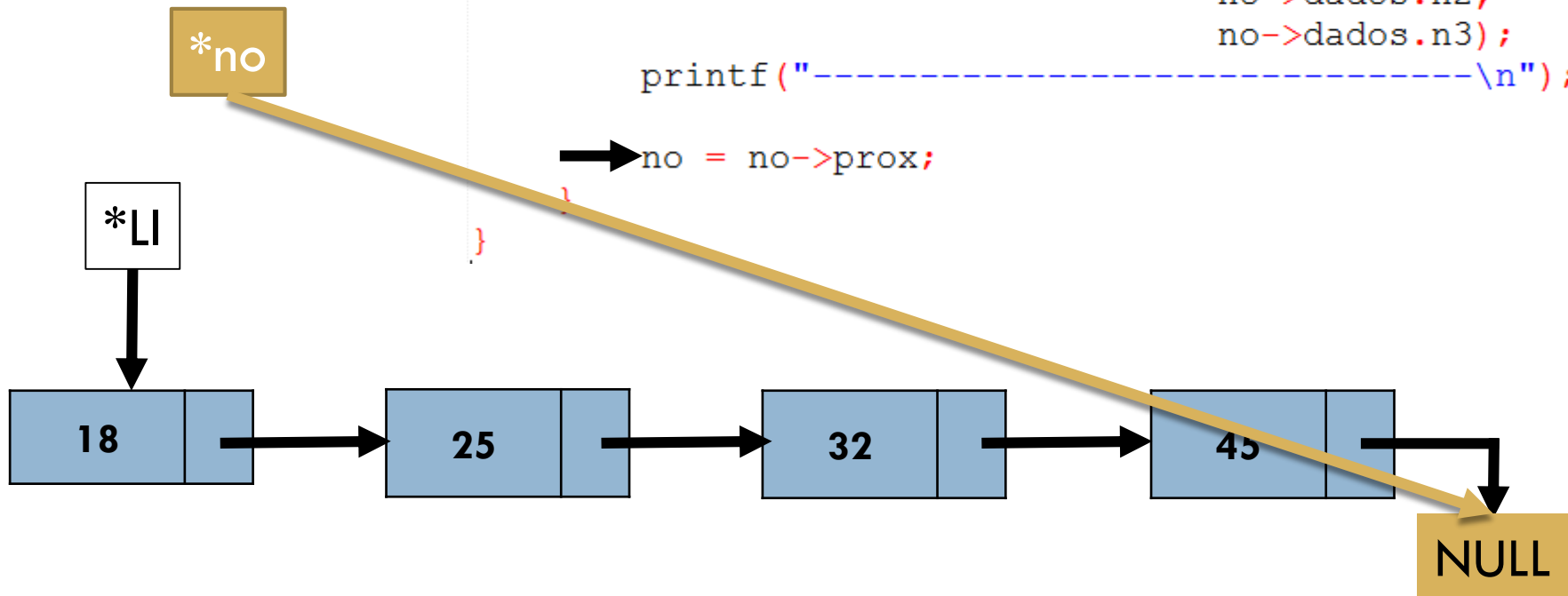
```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    Elem* no = *li;  
    while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
            no->dados.n2,  
            no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



Lista Dinâmica: Impressão

193

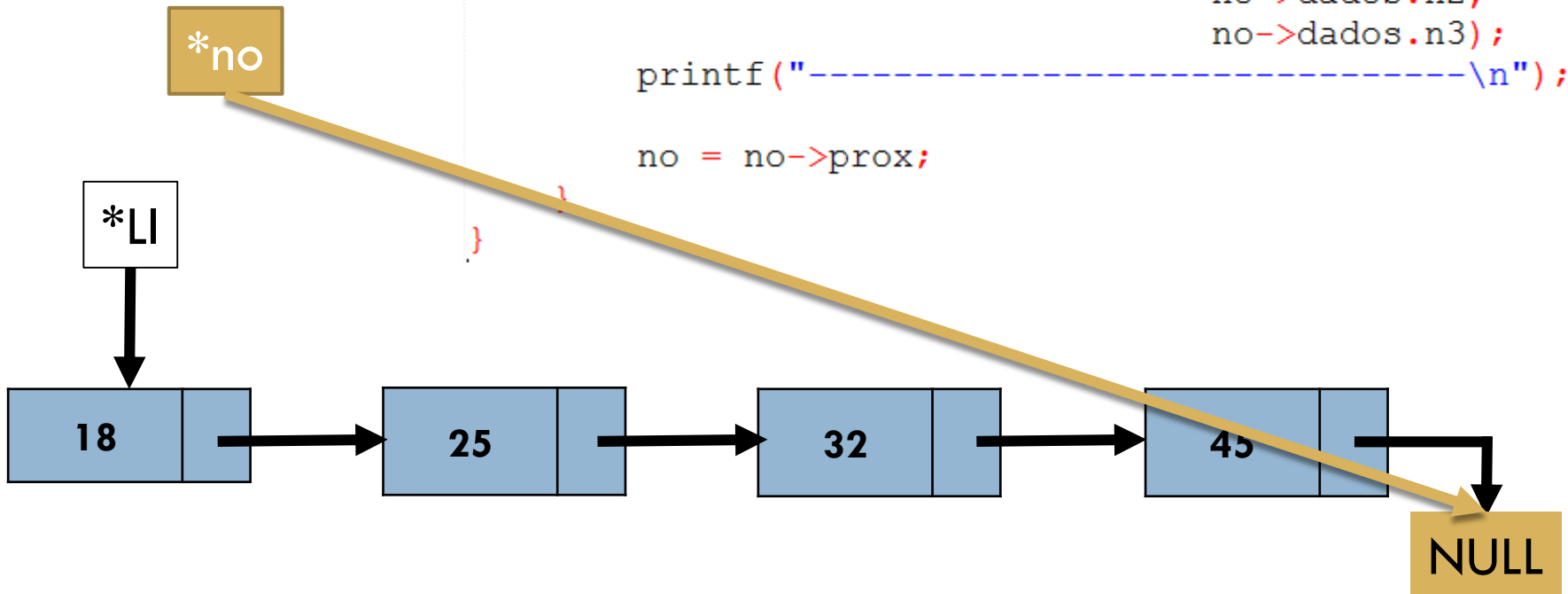
```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    Elem* no = *li;  
    while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
            no->dados.n2,  
            no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```



Lista Dinâmica: Impressão

194

```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    Elem* no = *li;  
    → while(no != NULL){  
        printf("Matricula: %d\n", no->dados.matricula);  
        printf("Nome: %s\n", no->dados.nome);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
            no->dados.n2,  
            no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```

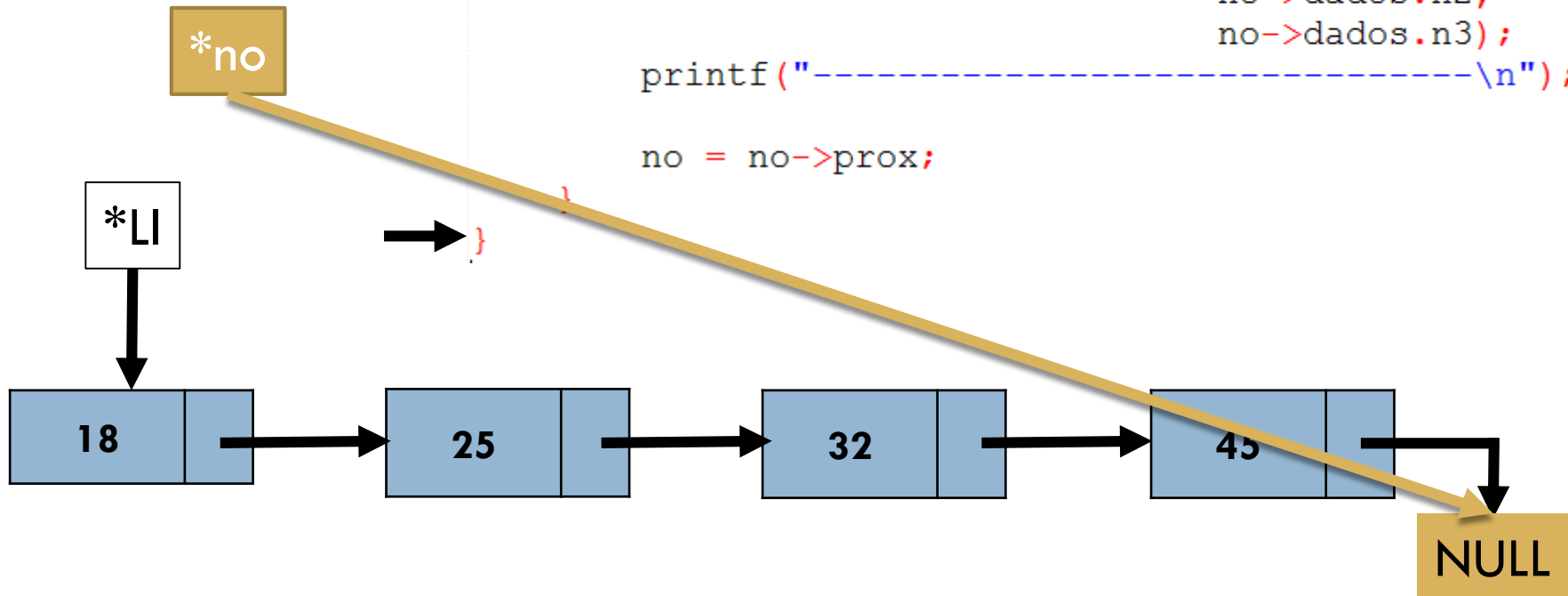


Lista Dinâmica: Impressão

195

```
void imprime_lista(Lista* li){  
    if(li == NULL)  
        return;  
    Elem* no = li->prox;  
    while(no != NULL){  
        printf("%d ", no->dados.n1);  
        printf("%d ", no->dados.n2);  
        printf("Notas: %f %f %f\n", no->dados.n1,  
                                     no->dados.n2,  
                                     no->dados.n3);  
        printf("-----\n");  
        no = no->prox;  
    }  
}
```

- FINAL DA IMPRESSÃO DA LISTA.



Lista Dinâmica

196

□ Vantagens:

- ▣ Melhor utilização dos recursos de memória.
- ▣ Sem movimentação dos elementos em operações de inserção e remoção.

□ Desvantagens:

- ▣ Acesso indireto aos elementos (trabalho com ponteiros).
- ▣ Necessidade de percorrer a lista para acessar um elemento.

Lista Dinâmica

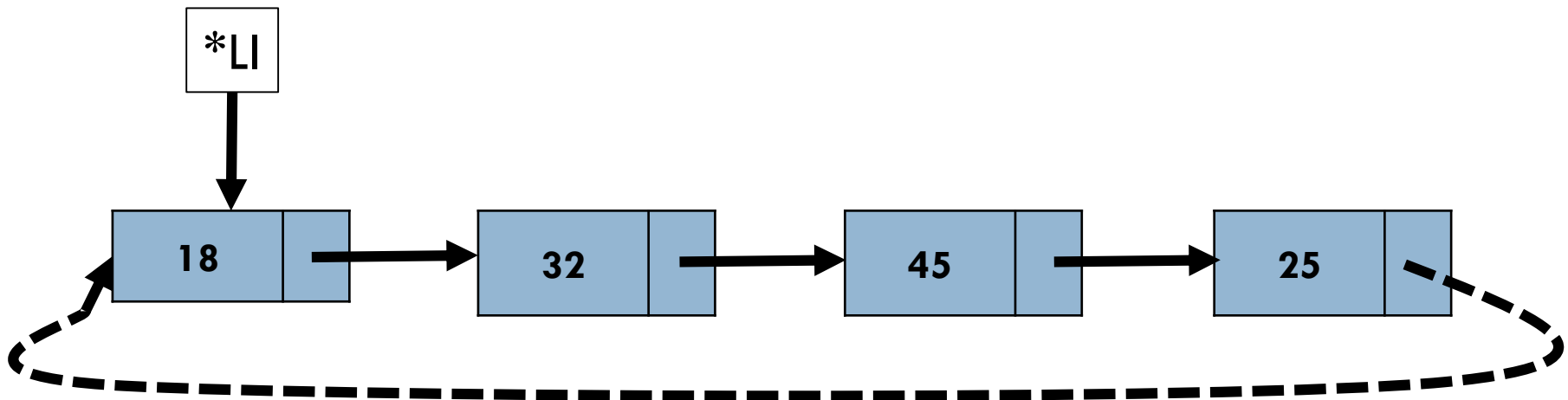
197

- Quando utilizar:
 - ▣ Problemas sem definição de espaço de memória;
 - ▣ Frequentes inserções e remoções em lista ordenada.

Lista Dinâmica: Lista Circular

198

- O **próximo do último** sempre será o **primeiro** elemento da lista.
 - ▣ Em que operações isso terá mais impacto?
 - ▣ Como saber quem é o último elemento?

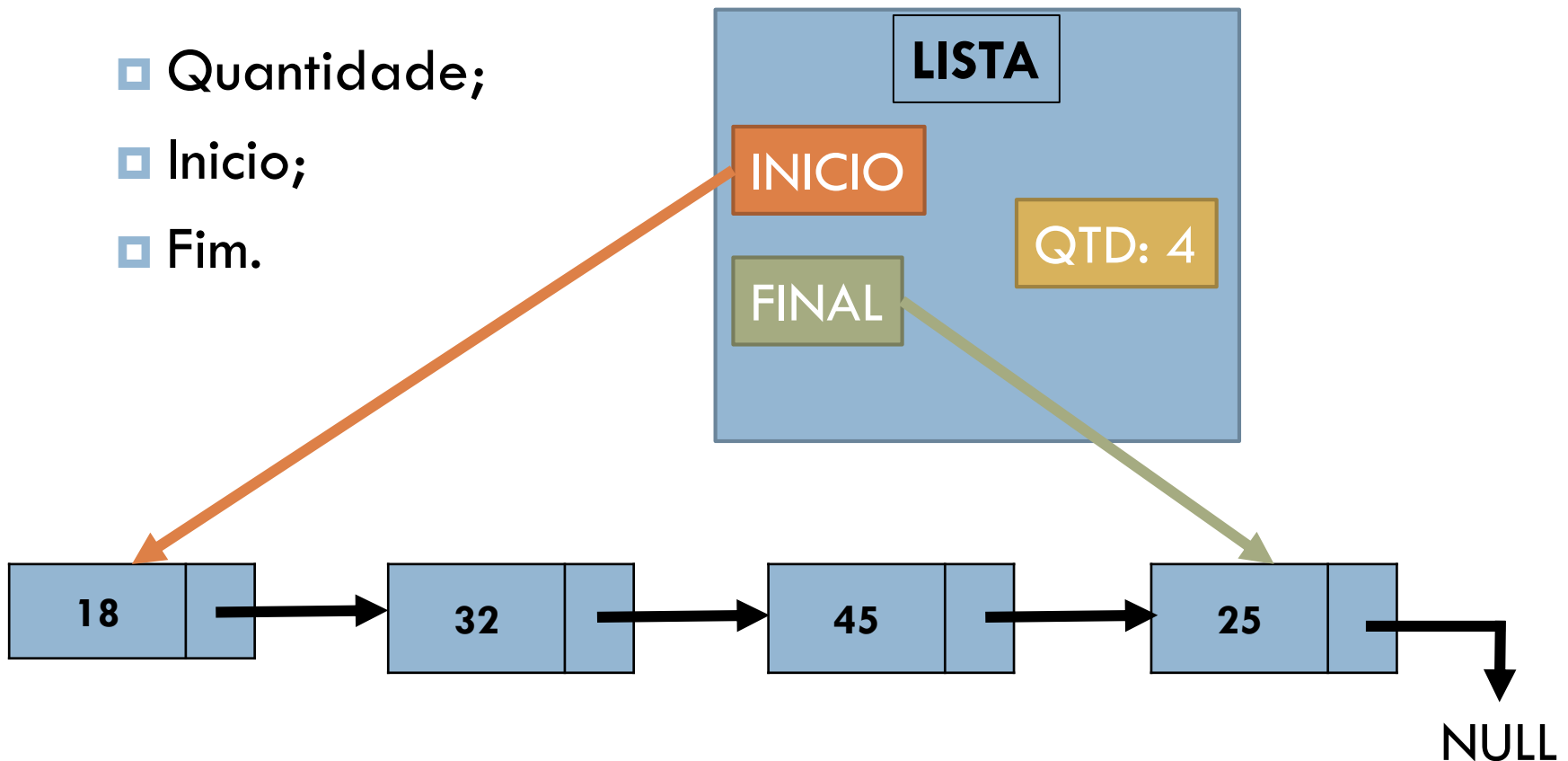


Lista Dinâmica: Nó descritor

199

- Consiste em criar uma **estrutura** que armazena mais dados sobre a lista:

- ▣ Quantidade;
- ▣ Inicio;
- ▣ Fim.



Lista Dinâmica: Nó descritor

200

- Consiste em criar uma **estrutura** que armazena mais dados sobre a lista:

- ▣ Quantidade;
- ▣ Inicio;
- ▣ Fim.

.c

```
//Definição do tipo lista
struct elemento{
    struct aluno dados;
    struct elemento *prox;
};
typedef struct elemento Elem;

//Definição do Nó Descritor
struct descritor{
    struct elemento *inicio;
    struct elemento *fim;
    int qtd;
};
```

.h

```
typedef struct descritor Lista;
```


Lista Dinâmica: Nó descritor

201

- Consiste em criar uma **estrutura** que armazena mais dados sobre a lista:
 - ▣ Quantidade;
 - ▣ Inicio;
 - ▣ Fim.

- Em que operações isso terá mais impacto?

Lista Dinâmica: Exercícios

202

- Baseado na lista dinâmica do exemplo, faça as implementações necessárias para adicionar as seguintes operações:
 - ▣ Descobrir se a lista está cheia (retornar);
 - ▣ Descobrir se a lista está vazia (retornar);
 - ▣ Descobrir o tamanho da lista (retornar);
 - ▣ Inserir elemento no final da lista;
 - ▣ Inserir elemento no início da lista;
 - ▣ Remover elemento do final da lista;
 - ▣ Remover elemento do início da lista;

Lista Dinâmica: Exercícios (cont.)

203

- Baseado na lista dinâmica do exemplo, faça as implementações necessárias para adicionar as seguintes operações:
 - ▣ Consultar um elemento:
 - Passe como parâmetros **a lista, a matrícula que será buscada e um ponteiro do elemento.**
 - Não retorne o elemento encontrado pelo retorno da função, armazene o endereço no ponteiro do parâmetro.

```
int consulta(Lista* li, int mat, struct aluno *al)
```