



UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS - CSHNB
CURSO DE SISTEMAS DE INFORMAÇÃO
PICOS - PI

LISTAS ENCADEADAS

Prof. Ma. Luana Batista da Cruz
luana.b.cruz@nca.ufma.br

Roteiro

2

- Introdução
- Lista encadeada simples
 - Estudo de caso
- Lista encadeada circular

Introdução

3

❏ Vetor

- Ocupa um espaço contíguo de memória
- Permite acesso randômico aos elementos
- Deve ser dimensionado com um número máximo de elementos



Introdução

4

❑ Vetor

- Ocupa um espaço contíguo de memória
- Permite acesso randômico aos elementos
- Deve ser dimensionado com um número máximo de elementos

❑ Limitações?

❑ Tamanho fixo

- Quantidade de elementos não pode ser maior que a quantidade declarada

❑ Desperdício de memória

- Caso todas as posições não sejam utilizadas

Introdução

5

❑ Estruturas de dados dinâmicas

- Crescem (ou decrescem) à medida que elementos são inseridos (ou removidos)

❑ Exemplo

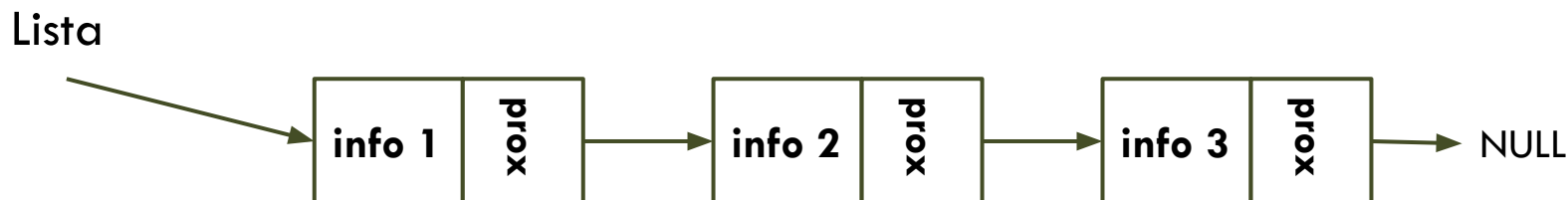
❑ Listas encadeadas

- Maior flexibilidade que vetores
- Conjunto de dados pode crescer ou diminuir
- Evita o desperdício de memória
- Elementos podem ser inseridos ou removidos em posições específicas (sem necessidade de reordenação do restante dos elementos)

Lista encadeada simples

6

- ❑ Sequência encadeada de elementos, chamados de nós da lista
- ❑ **Nó da lista** é representado por **dois campos**:
 - A **informação** armazenada
 - O **ponteiro** para o próximo elemento da lista
- ❑ O ponteiro do último elemento é NULL (indica o fim da lista)
- ❑ A **lista** é representada por um ponteiro para o primeiro nó



Lista encadeada simples

7

❏ Exemplo

- Lista encadeada armazenando valores inteiros

```
typedef struct lista Lista;  
  
struct lista{  
    int info;  
    Lista *prox;  
};
```



Lista encadeada simples

8

❏ **Função inicializa**

- Cria uma lista vazia, representada pelo ponteiro NULL

```
Lista* inicializa (void){  
    return NULL;  
}
```

Lista → NULL

Lista encadeada simples

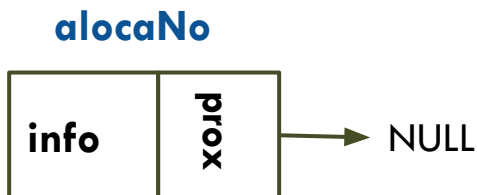
9

❏ Função insere

- Aloca memória para armazenar o elemento (nó)
- Encadeia o elemento na lista existente

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```



Lista encadeada simples

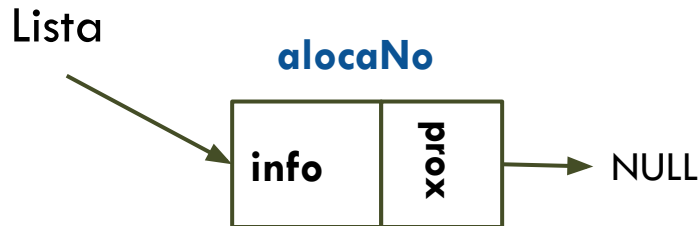
10

❏ Função insere

- Aloca memória para armazenar o elemento (nó)
- Encadeia o elemento na lista existente

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```



Lista encadeada simples

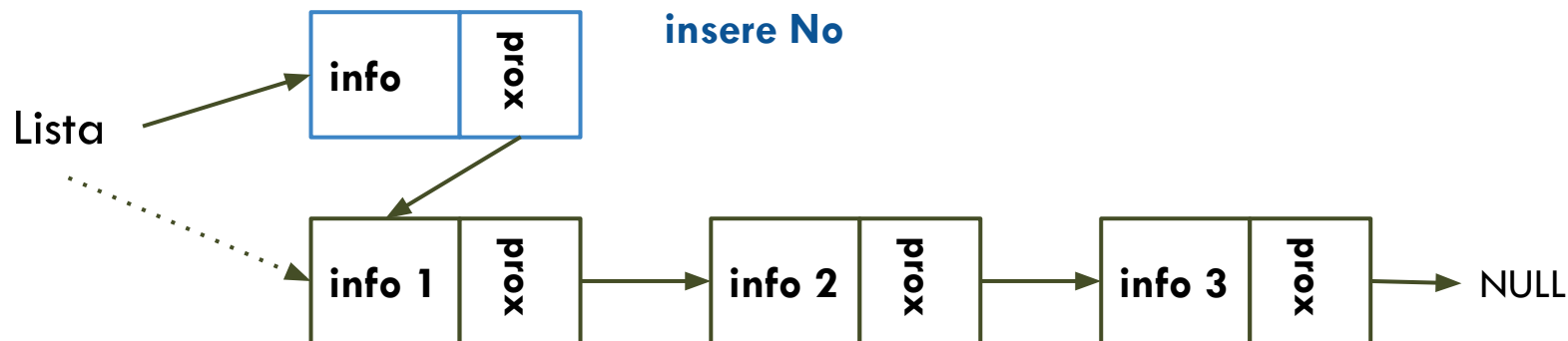
11

❏ Função insere

- Aloca memória para armazenar o elemento (nó)
- Encadeia o elemento na lista existente

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```



Lista encadeada simples

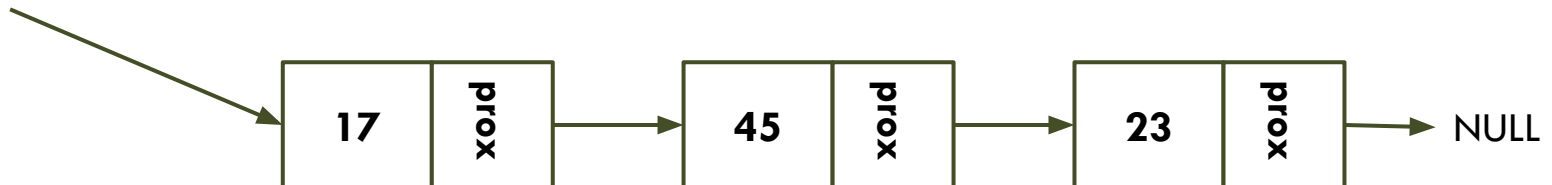
12

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p != NULL; p = p->prox){  
        printf(" %d ", p->info);  
    }  
}
```

Lista



Lista encadeada simples

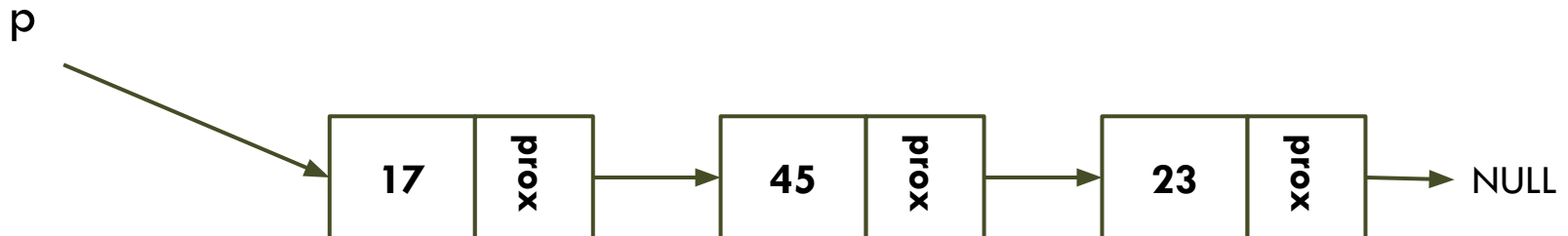
13

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p != NULL; p = p->prox){  
        printf(" %d ", p->info);  
    }  
}
```

Saída:



Lista encadeada simples

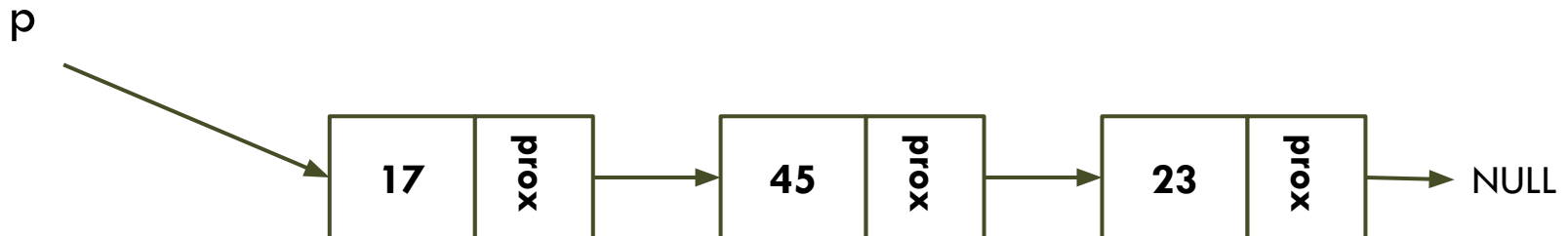
14

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p != NULL; p = p->prox){  
        printf(" %d ", p->info);  
    }  
}
```

Saída: 17



Lista encadeada simples

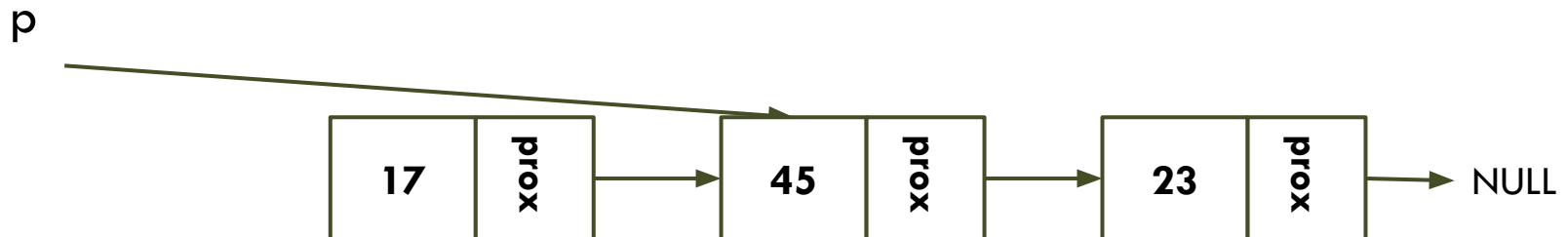
15

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p!= NULL; p = p->prox){  
        printf(" %d ", p->info);  
    }  
}
```

Saída: 17



Lista encadeada simples

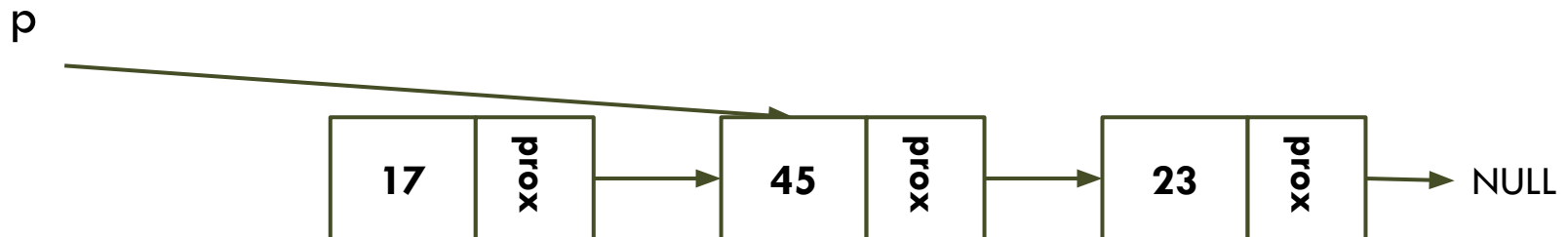
16

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p != NULL; p = p->prox){  
        printf(" %d ", p->info);  
    }  
}
```

Saída: 17 45



Lista encadeada simples

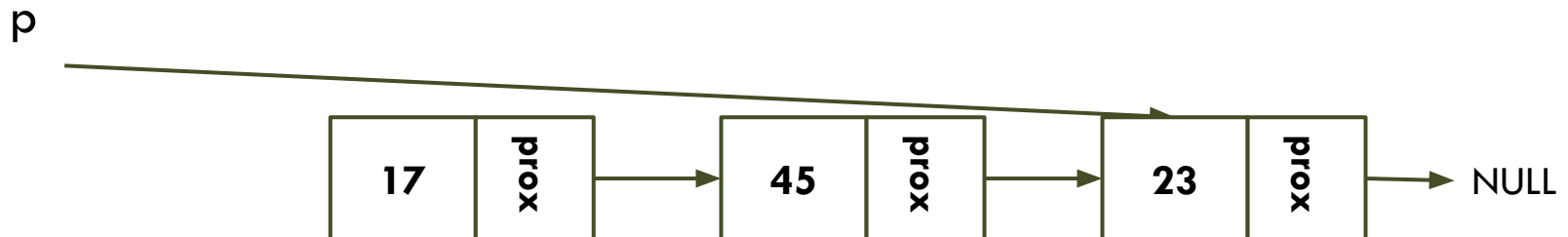
17

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p!= NULL; p = p->prox){  
        printf(" %d ", p->info);  
    }  
}
```

Saída: 17 45



Lista encadeada simples

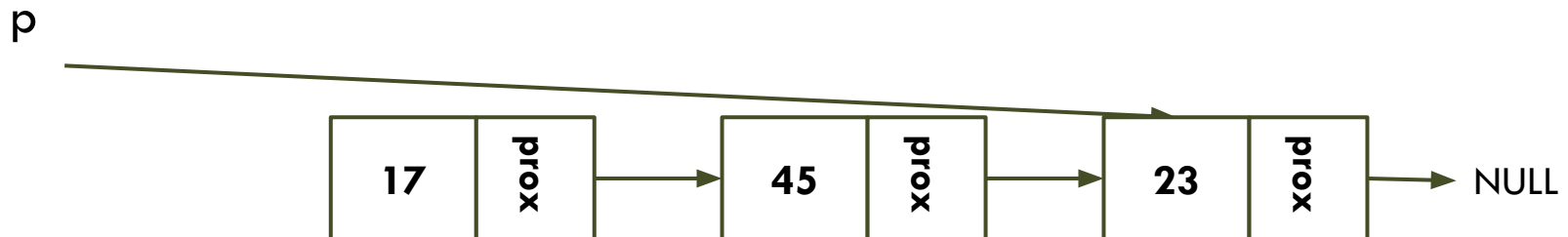
18

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p != NULL; p = p->prox){  
        printf(" %d ", p->info);  
    }  
}
```

Saída: 17 45 23



Lista encadeada simples

19

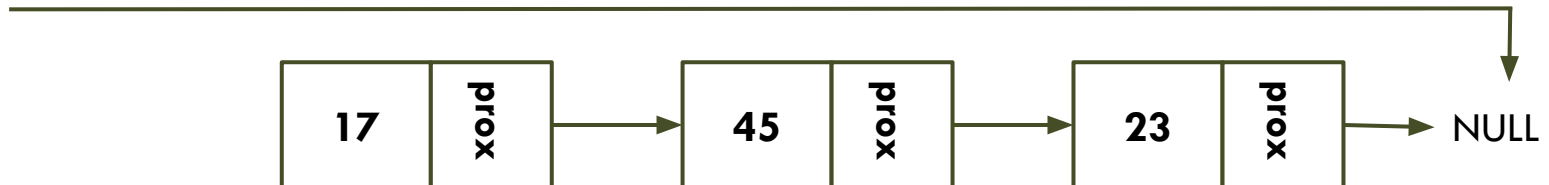
❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p != NULL; p = p->prox){  
        printf(" %d ", p->info);  
    }  
}
```

Saída: 17 45 23

p



Lista encadeada simples

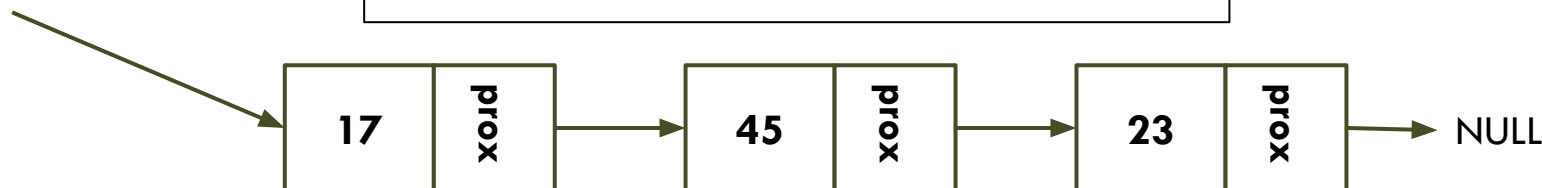
20

❏ Função busca

- Recebe a informação referente ao elemento a pesquisar
- Retorna o ponteiro do nó da lista que representa o elemento, ou NULL, caso o elemento não seja encontrado na lista

```
Lista* busca (Lista* l, int v){  
    Lista* p;  
    for (p=l; p!=NULL; p = p->prox) {  
        if (p->info == v)  
            return p;  
    }  
    return NULL;  
}
```

Lista



Lista encadeada simples

21

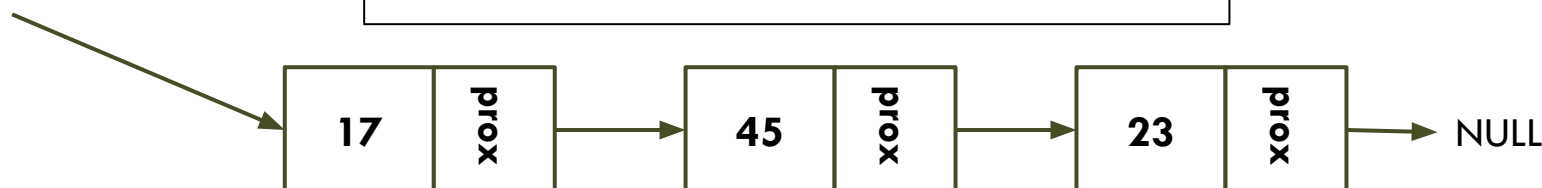
❏ Função busca

- Recebe a informação referente ao elemento a pesquisar
- Retorna o ponteiro do nó da lista que representa o elemento, ou NULL, caso o elemento não seja encontrado na lista

```
Lista* busca (Lista* l, int v){  
    Lista* p;  
    for (p=l; p!=NULL; p = p->prox) {  
        if (p->info == v)  
            return p;  
    }  
    return NULL;  
}
```

Buscando: 45

Lista



Lista encadeada simples

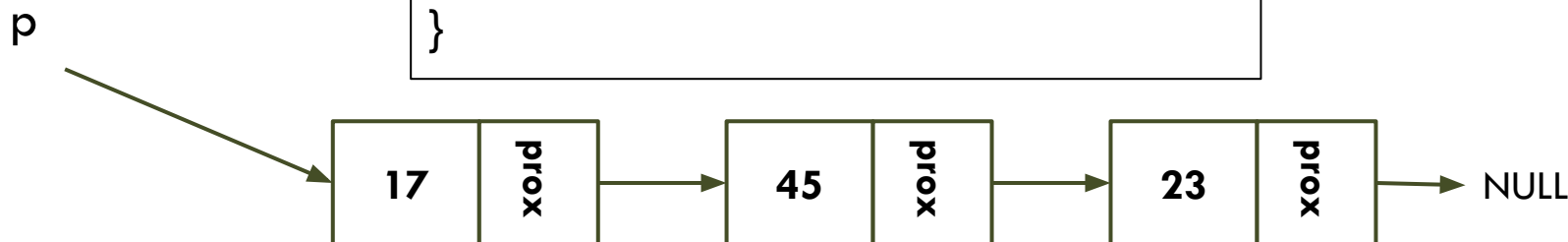
22

❏ Função busca

- Recebe a informação referente ao elemento a pesquisar
- Retorna o ponteiro do nó da lista que representa o elemento, ou NULL, caso o elemento não seja encontrado na lista

```
Lista* busca (Lista* l, int v){  
    Lista* p;  
    for (p=l; p!=NULL; p = p->prox) {  
        if (p->info == v)  
            return p;  
    }  
    return NULL;  
}
```

Buscando: 45



Lista encadeada simples

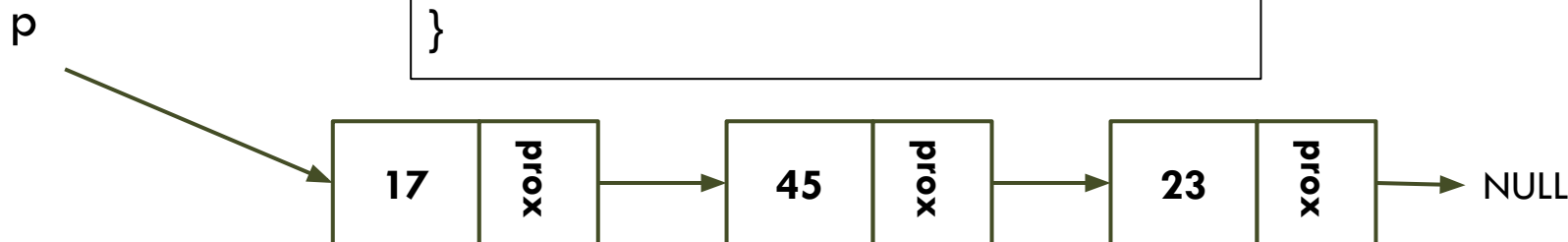
23

❏ Função busca

- Recebe a informação referente ao elemento a pesquisar
- Retorna o ponteiro do nó da lista que representa o elemento, ou NULL, caso o elemento não seja encontrado na lista

```
Lista* busca (Lista* l, int v){  
    Lista* p;  
    for (p=l; p!=NULL; p = p->prox) {  
        if (p->info == v)  
            return p;  
    }  
    return NULL;  
}
```

Buscando: 45



Lista encadeada simples

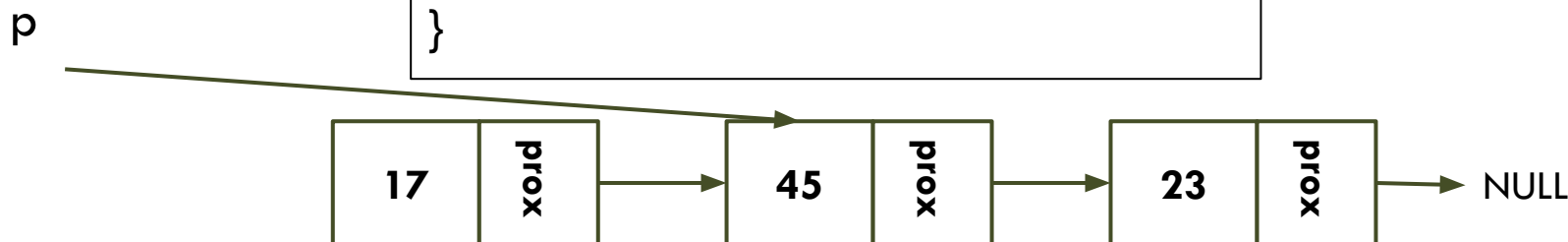
24

❏ Função busca

- Recebe a informação referente ao elemento a pesquisar
- Retorna o ponteiro do nó da lista que representa o elemento, ou NULL, caso o elemento não seja encontrado na lista

```
Lista* busca (Lista* l, int v){  
    Lista* p;  
    for (p=l; p!=NULL; p = p->prox) {  
        if (p->info == v)  
            return p;  
    }  
    return NULL;  
}
```

Buscando: 45



Lista encadeada simples

25

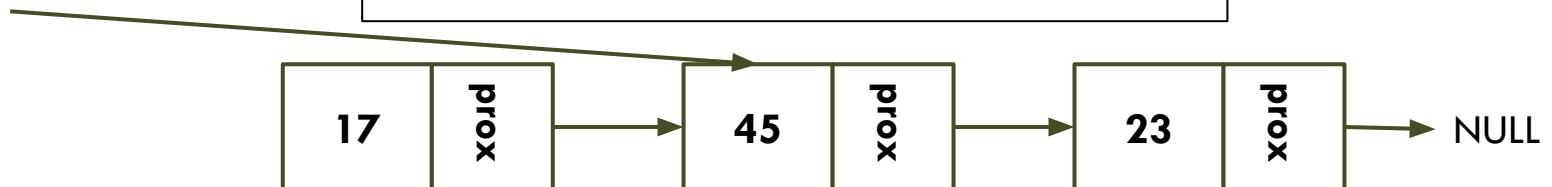
❑ Função busca

- Recebe a informação referente ao elemento a pesquisar
- Retorna o ponteiro do nó da lista que representa o elemento, ou NULL, caso o elemento não seja encontrado na lista

```
Lista* busca (Lista* l, int v){  
    Lista* p;  
    for (p=l; p!=NULL; p = p->prox) {  
        if (p->info == v)  
            return p;  
    }  
    return NULL;  
}
```

Buscando: 45

p



Lista encadeada simples

26

❏ Função busca

- Recebe a informação referente ao elemento a pesquisar
- Retorna o ponteiro do nó da lista que representa o elemento, ou NULL, caso o elemento não seja encontrado na lista

```
Lista* busca (Lista* l, int v){  
    Lista* p;  
    for (p=l; p!=NULL; p = p->prox) {  
        if (p->info == v)  
            return p;  
    }  
    return NULL;  
}
```

Buscando: 45

Retorna p

p

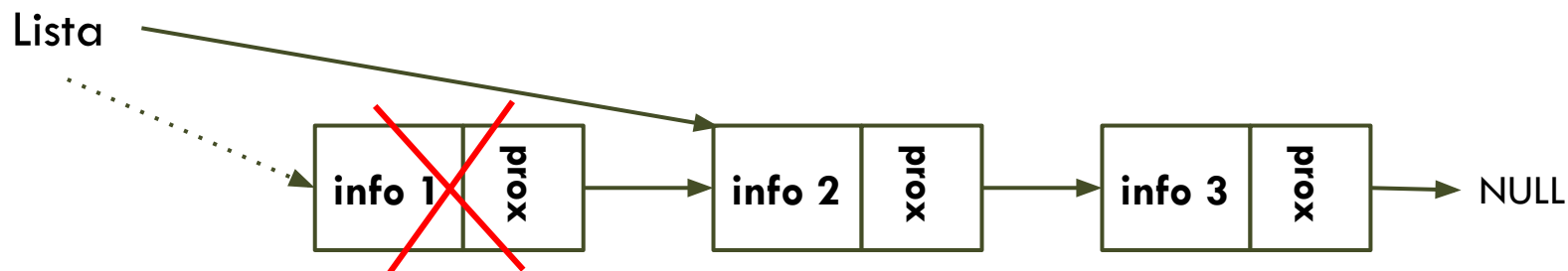


Lista encadeada simples

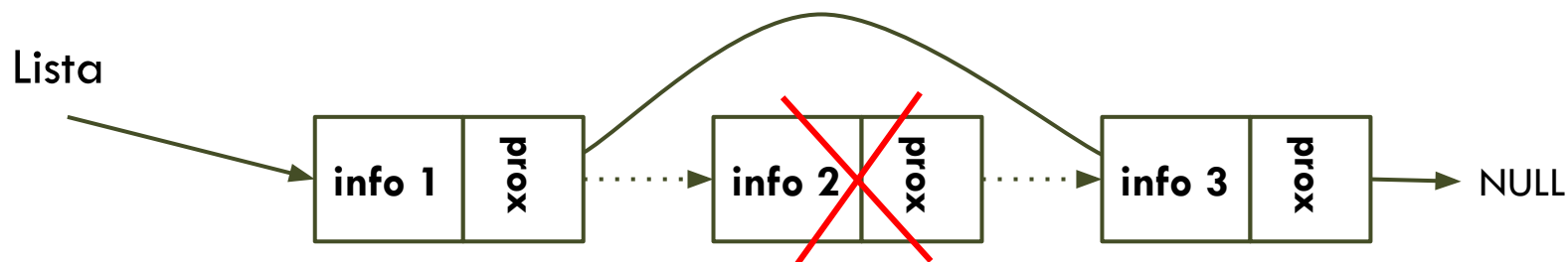
27

❏ Função retira

- Recebe como entrada a lista e o valor do elemento a retirar
- Atualiza o valor da lista, se o elemento removido for o primeiro



- Caso seja o elemento do meio, remove o elemento da lista e o ponteiro do elemento anterior aponta para o próximo

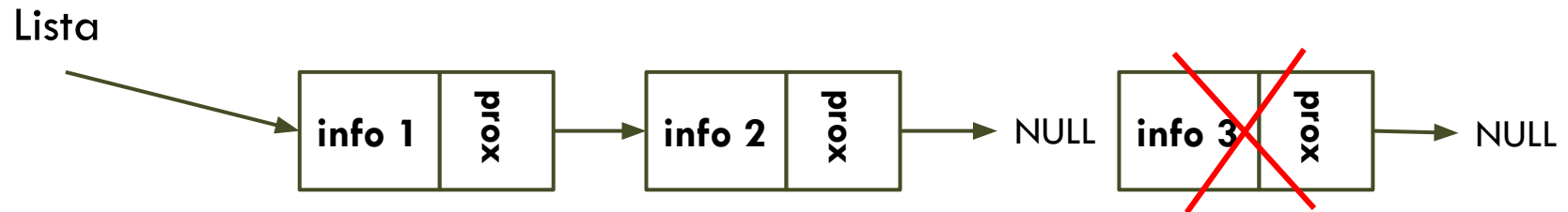


Lista encadeada simples

28

❏ Função retira

- Caso seja o último elemento, remove da lista e o ponteiro do elemento anterior aponta para NULL



Lista encadeada simples

29

❏ Função retira

```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;

    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

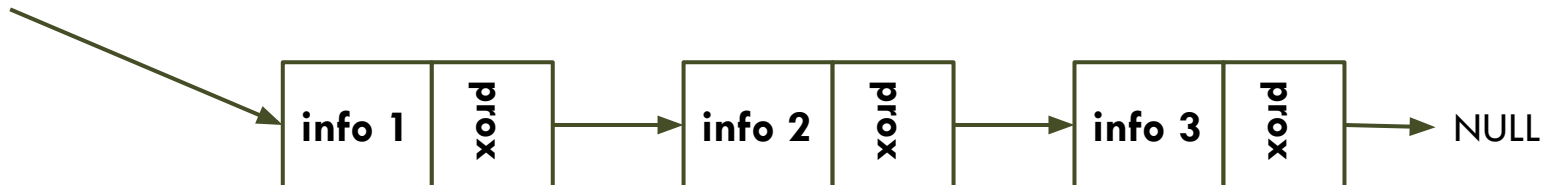
30

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```

Lista



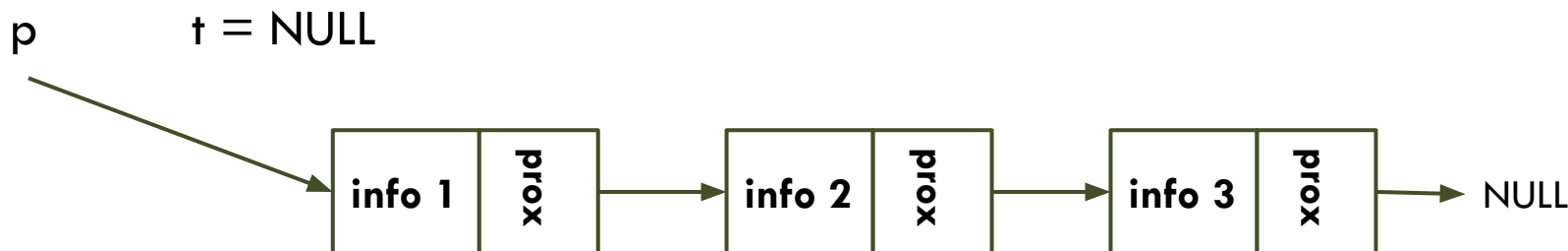
Lista encadeada simples

31

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```



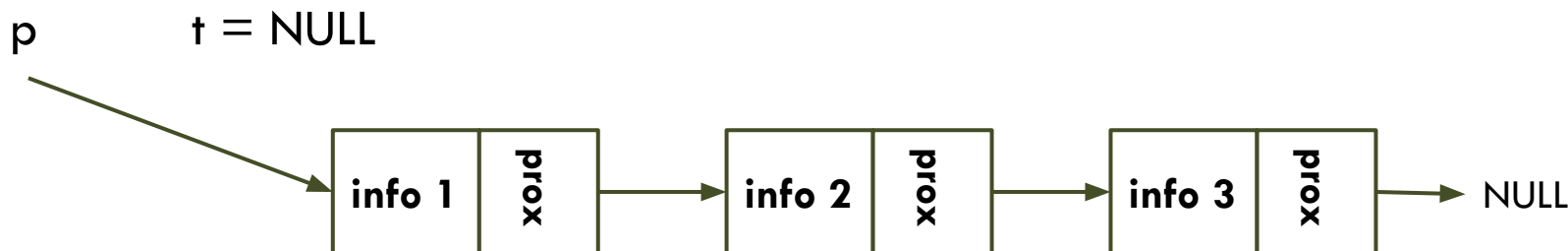
Lista encadeada simples

32

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```



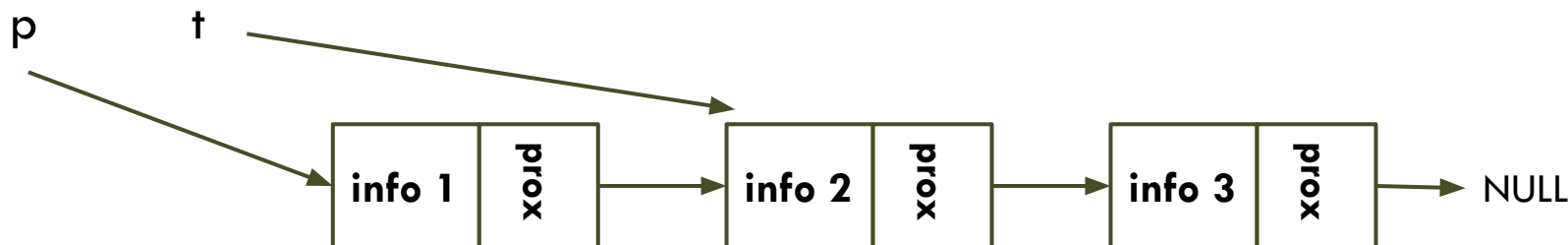
Lista encadeada simples

33

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```



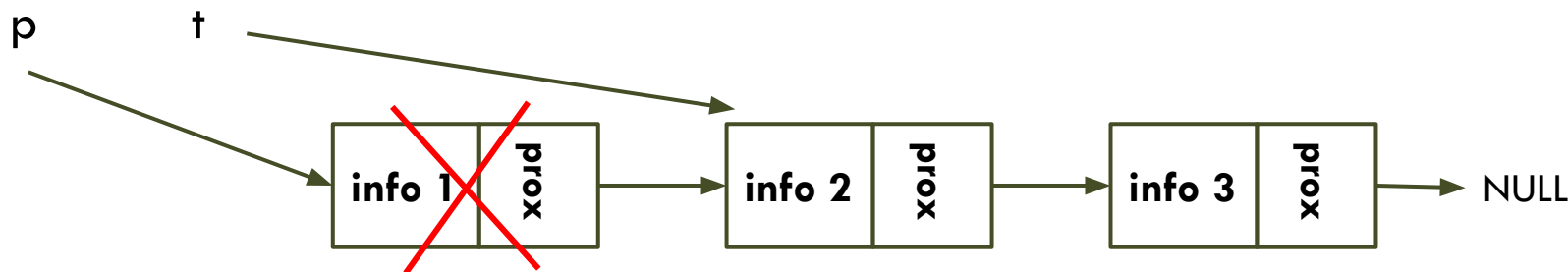
Lista encadeada simples

34

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```



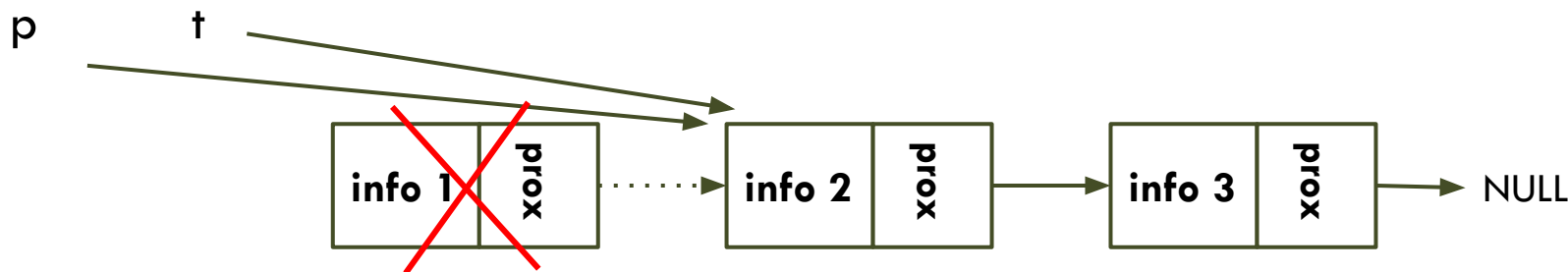
Lista encadeada simples

35

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```



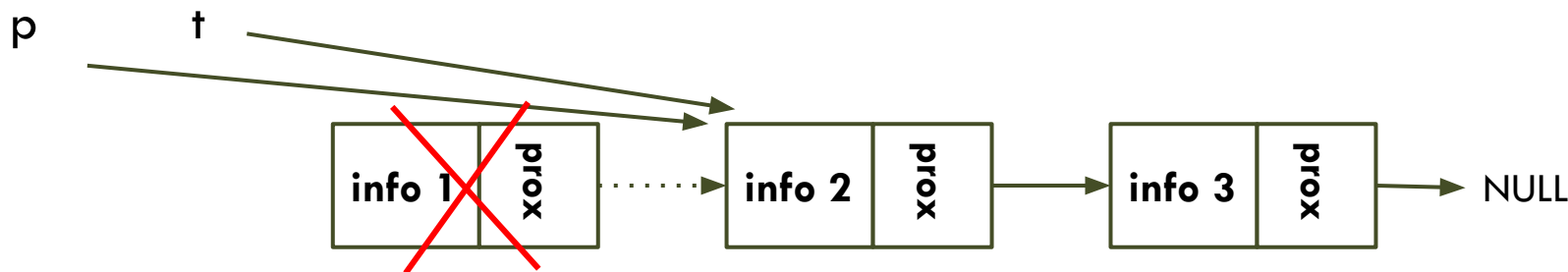
Lista encadeada simples

36

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```



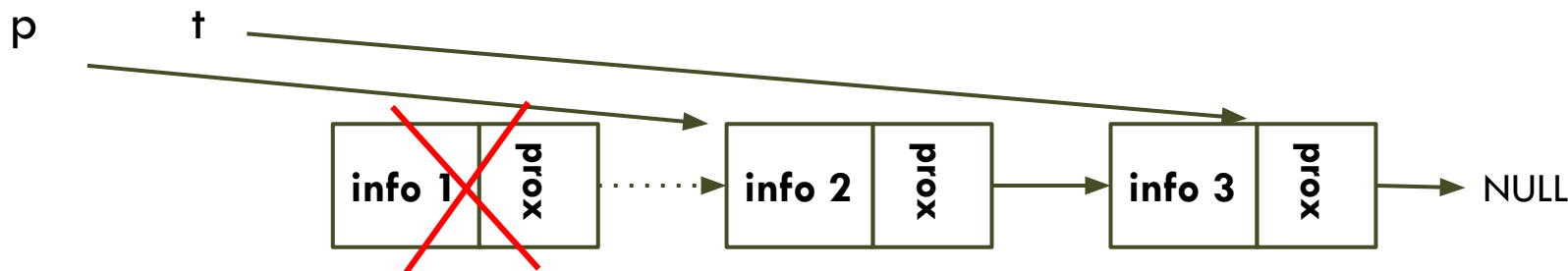
Lista encadeada simples

37

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```



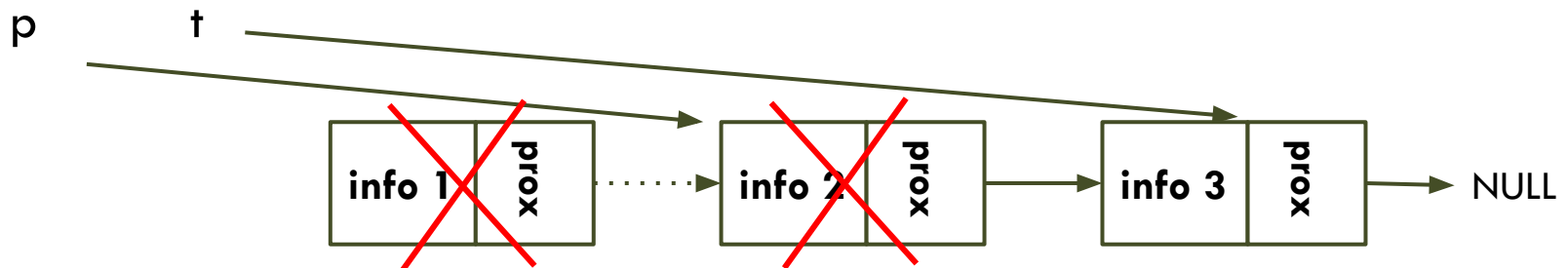
Lista encadeada simples

38

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```



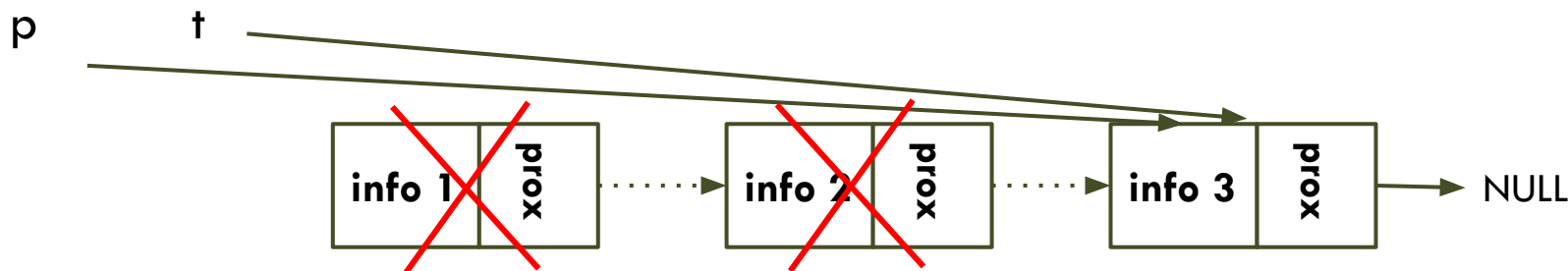
Lista encadeada simples

39

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```



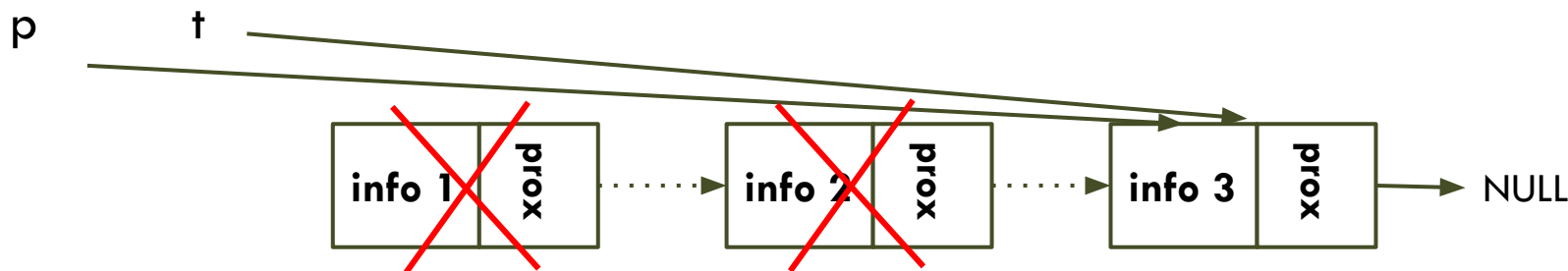
Lista encadeada simples

40

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```



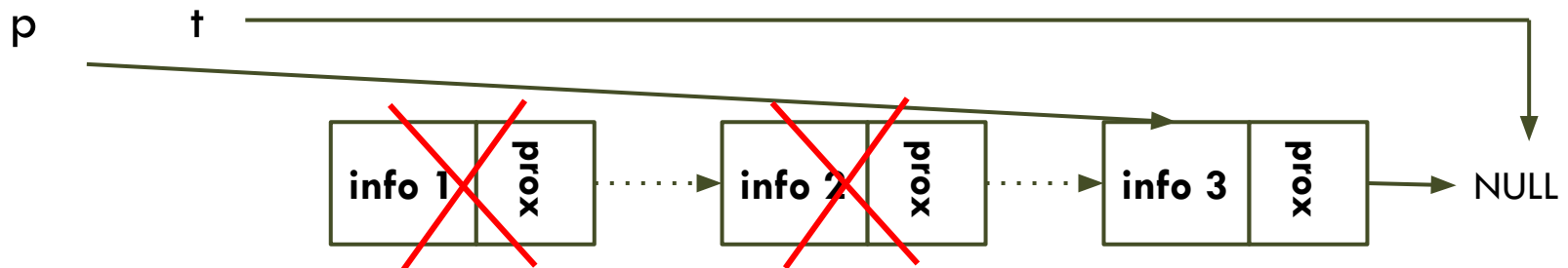
Lista encadeada simples

41

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```



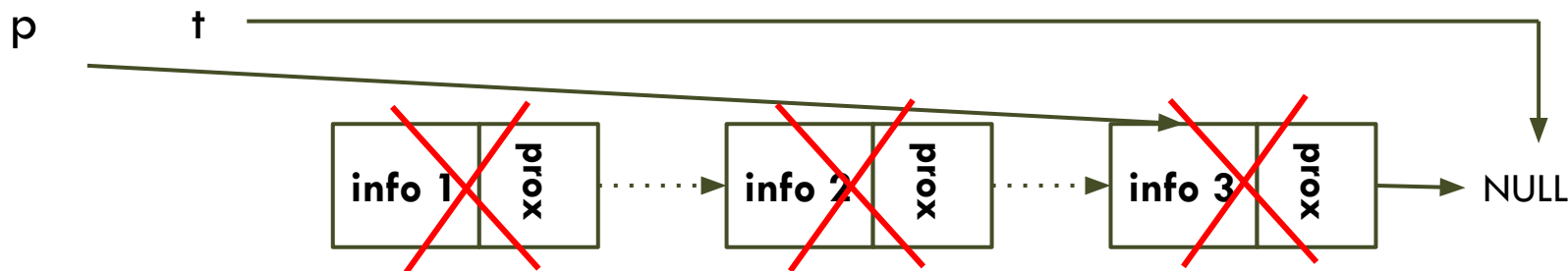
Lista encadeada simples

42

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```



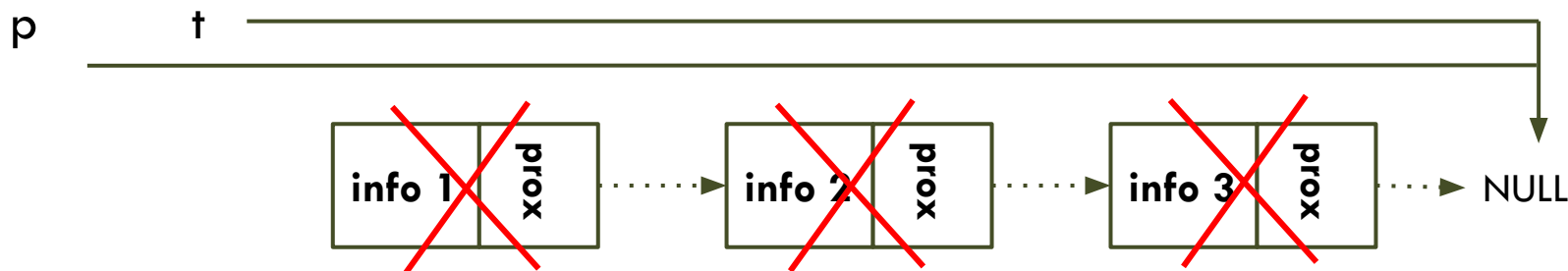
Lista encadeada simples

43

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```



Lista encadeada simples

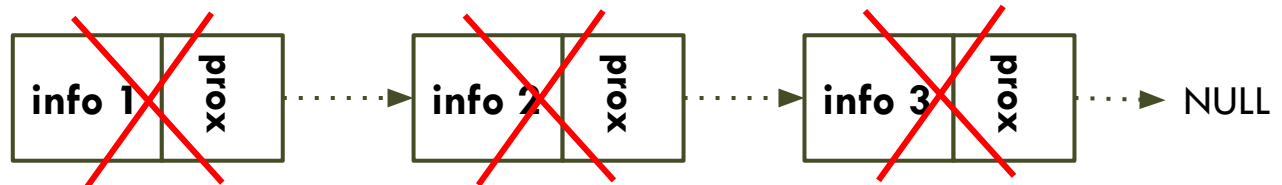
44

❏ Função libera

- Liberado todos os elementos (nós) alocados

```
Lista* libera (Lista* l){  
    Lista* p = l;  
    Lista* t = NULL;  
    while (p != NULL) {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
    return NULL;  
}
```

Lista → NULL



Lista encadeada simples

45

❏ Estudo de caso

```
int main (){
    Lista* lista;
    lista = inicializa();
    lista = insere(lista, 23);
    lista = insere(lista, 45);
    lista = insere(lista, 17);
    imprime(lista);

    //remove início
    remove(lista, 17);
    // remove do meio
    // remove(lista, 45);
    // remove fim
    // remove(lista, 23);
    return 0;
}
```

Lista encadeada simples

Estudo de caso - inicializa

46

Passo 1

lista = lixo

main()

Lista* lista;
lista = inicializa();

Passo 2

lista = NULL

```
Lista* inicializa (void){  
    return NULL;  
}
```

Lista encadeada simples

Estudo de caso - insere

47

lista = NULL

main()

lista = insere(lista, 23);



```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

Lista encadeada simples

Estudo de caso - insere

48

lista = NULL

main()

lista = insere(lista, 23);



```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```



Lista encadeada simples

Estudo de caso - insere

49

lista = NULL

main()

lista = insere(lista, 23);



```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```



Lista encadeada simples

Estudo de caso - insere

50

lista = NULL



main()

lista = insere(lista, 23);



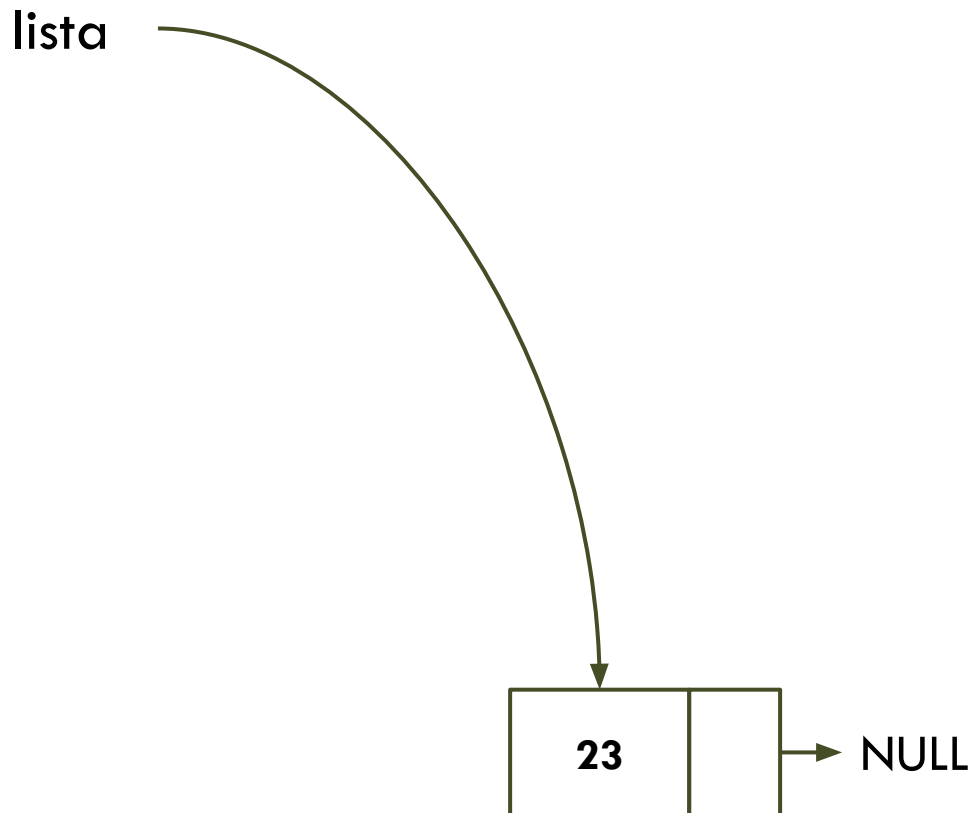
```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

Lista encadeada simples

Estudo de caso - insere

51



main()

lista = insere(lista, 23);

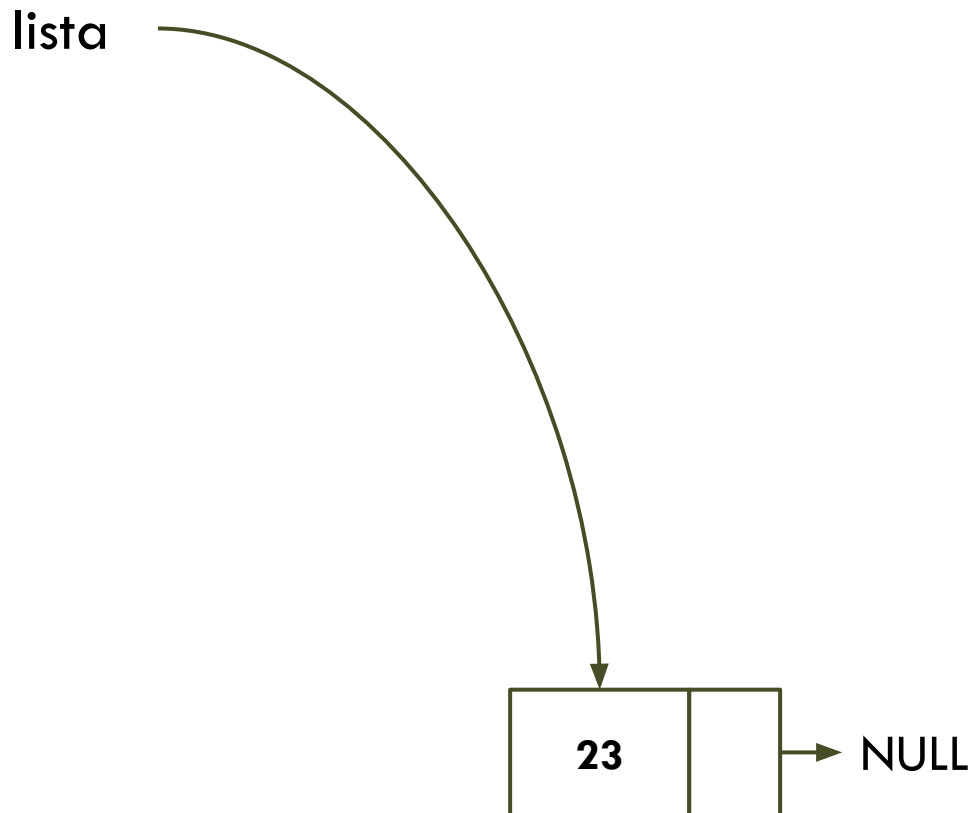
```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

Lista encadeada simples

Estudo de caso - insere

52



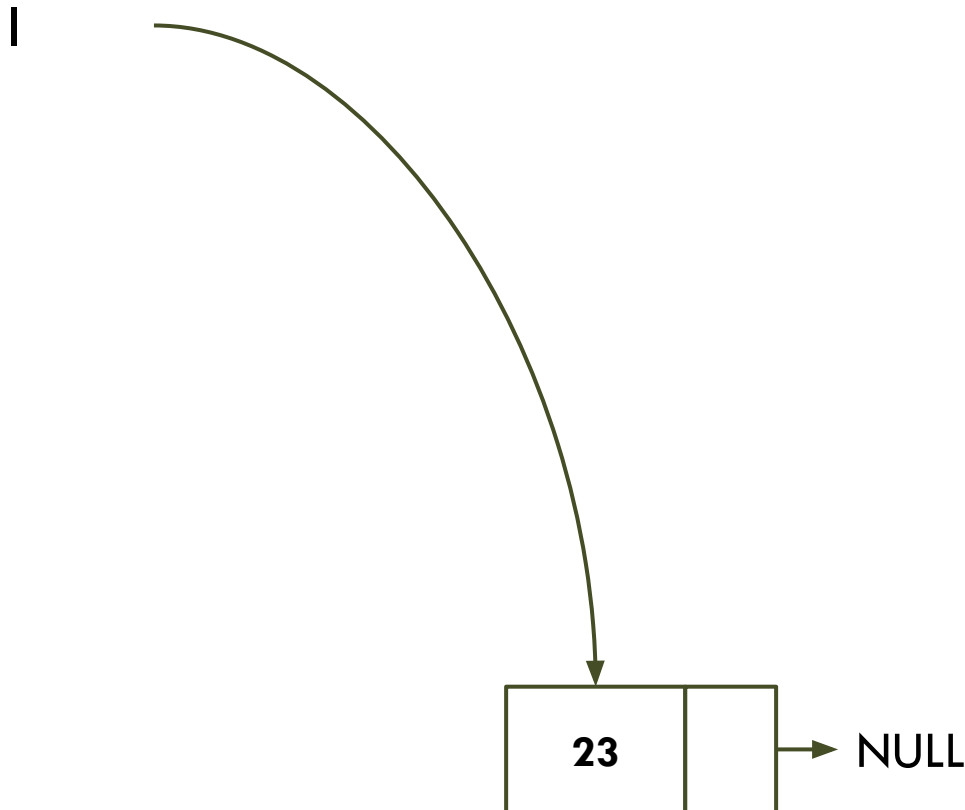
main()

```
lista = insere(lista, 45);
```

Lista encadeada simples

Estudo de caso - insere

53



main()

lista = insere(lista, 45);



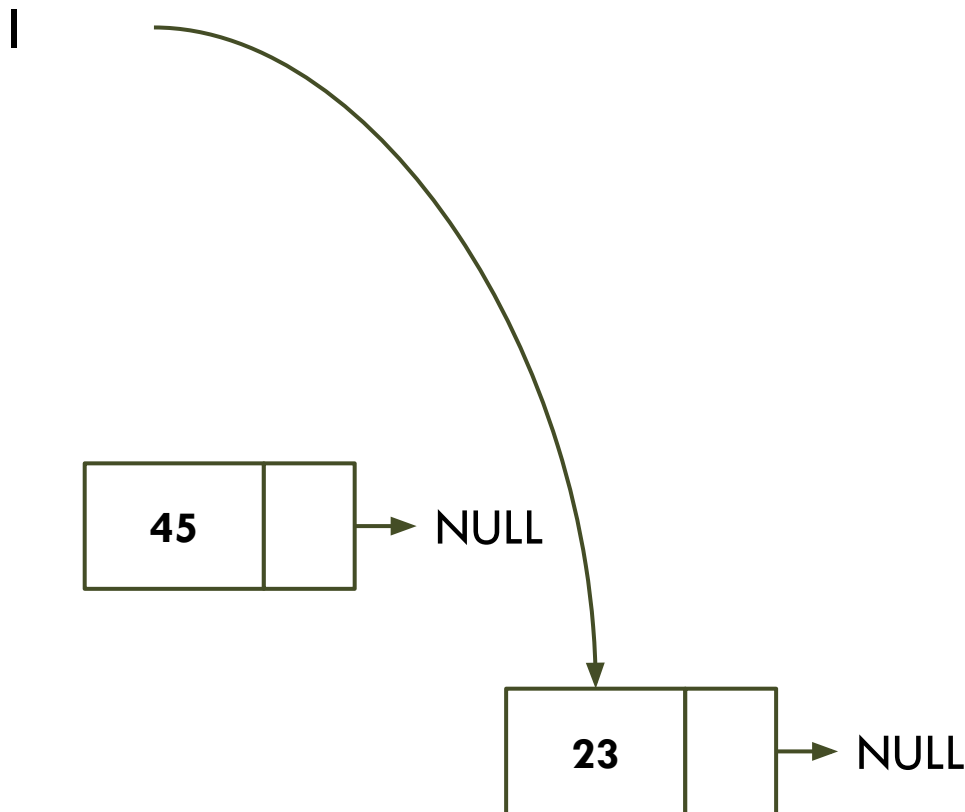
```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

Lista encadeada simples

Estudo de caso - insere

54



main()

```
lista = insere(lista, 45);
```

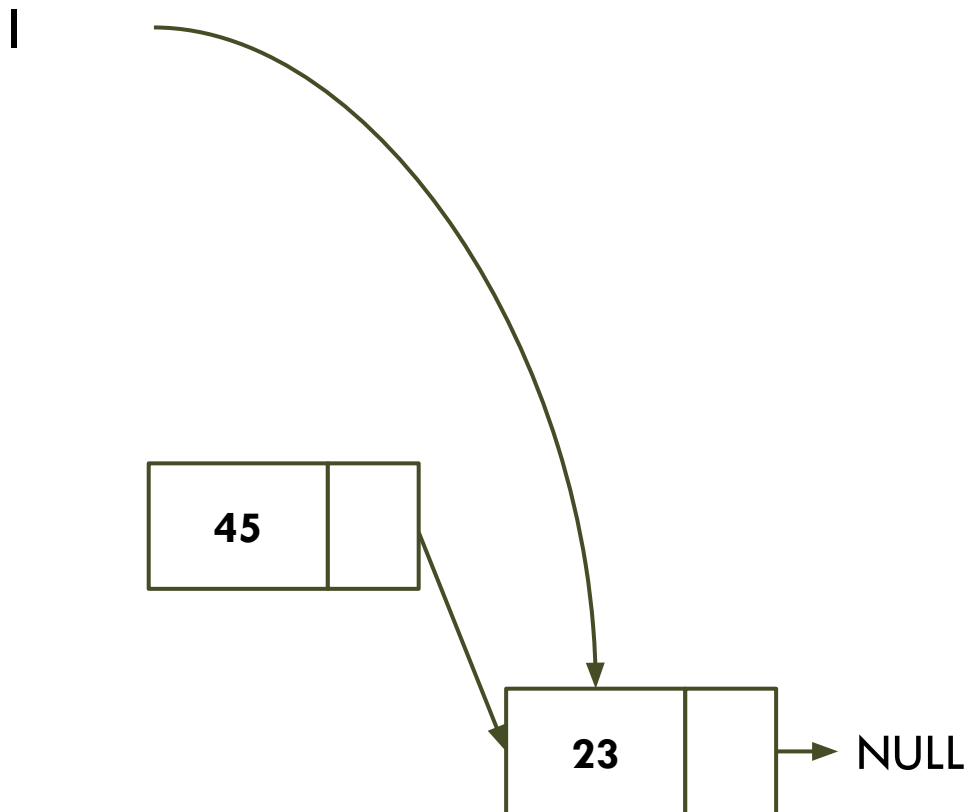
```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

Lista encadeada simples

Estudo de caso - insere

55



main()

```
lista = insere(lista, 45);
```

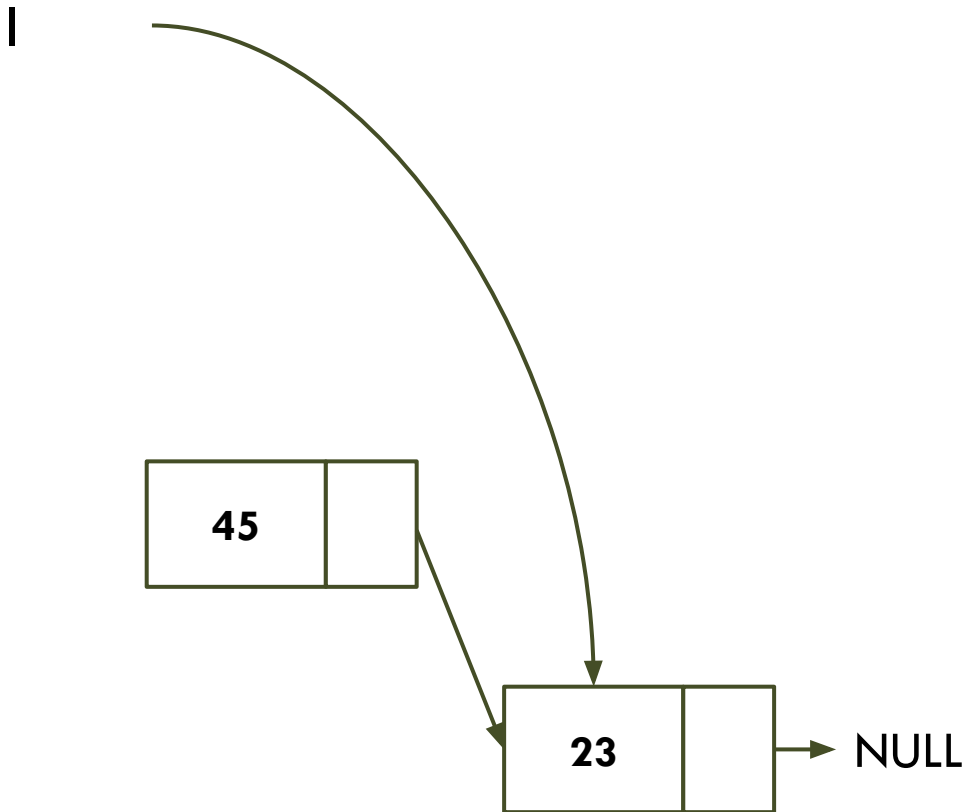
```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

Lista encadeada simples

Estudo de caso - insere

56



main()

```
lista = insere(lista, 45);
```

```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```

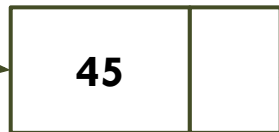
```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```


Lista encadeada simples

Estudo de caso - insere

57

lista



NULL

main()

`lista = insere(lista, 45);`

```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```

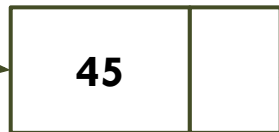
```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

Lista encadeada simples

Estudo de caso - insere

58

lista



NULL

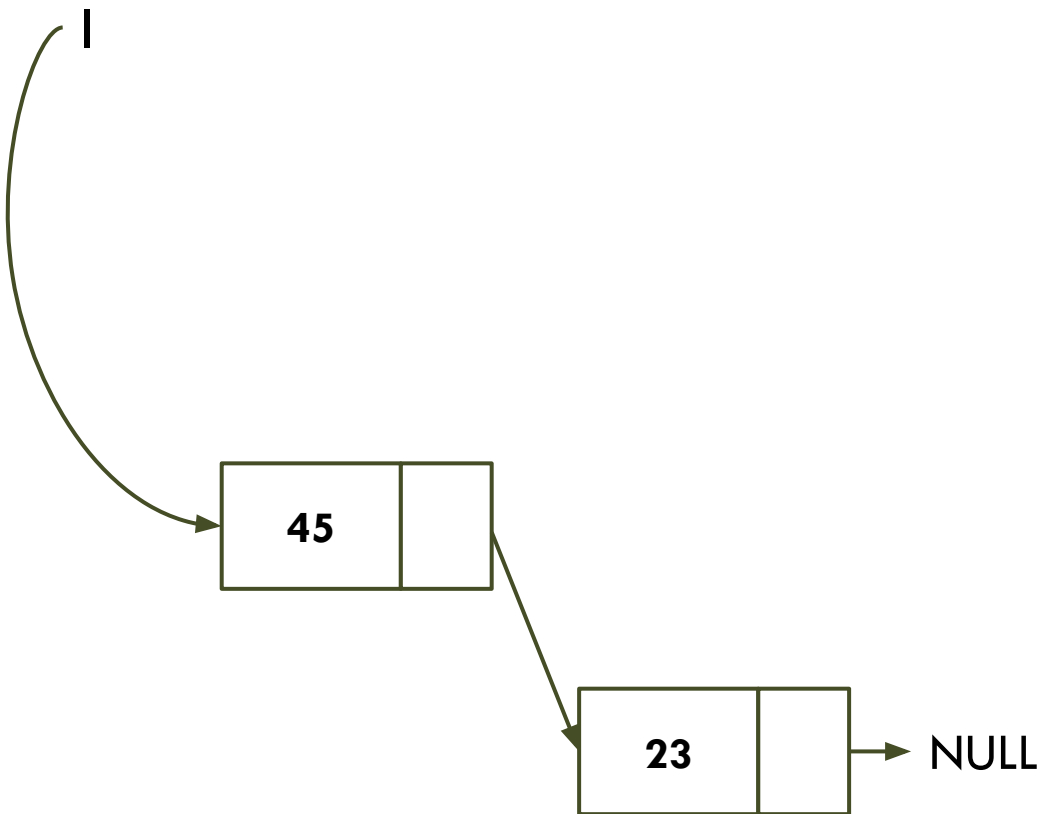
main()

```
lista = insere(lista, 17);
```

Lista encadeada simples

Estudo de caso - insere

59



main()

lista = insere(lista, 17);

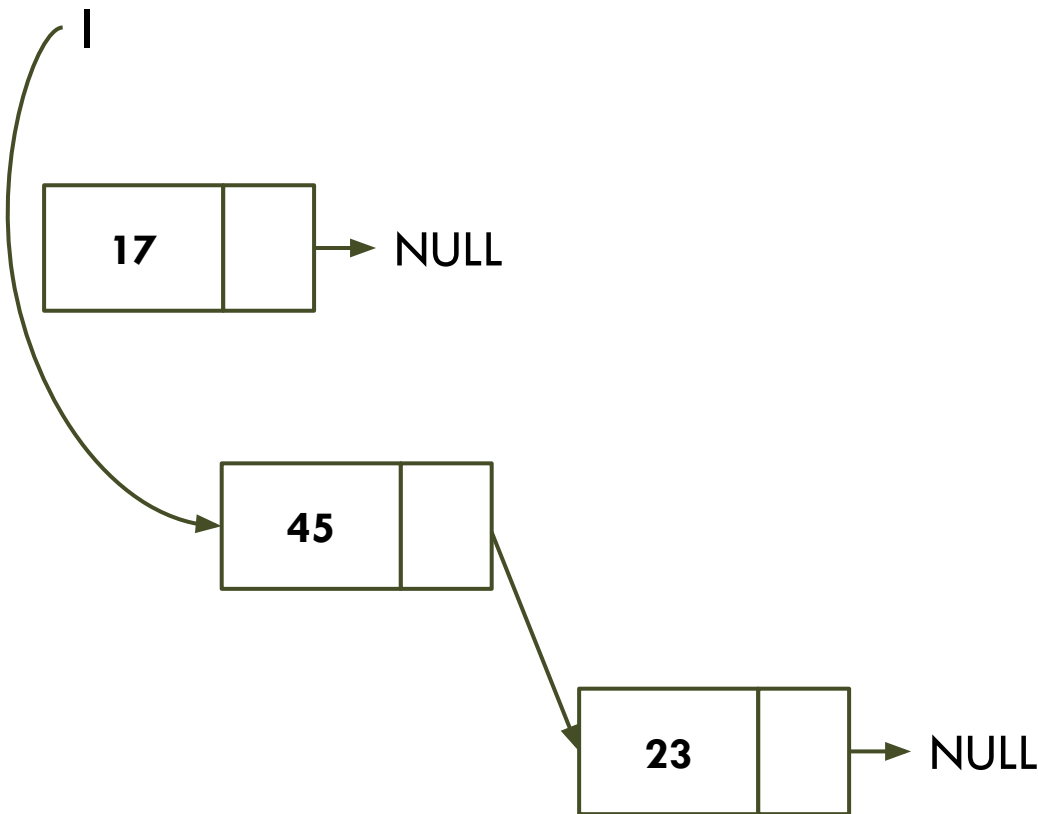
```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

Lista encadeada simples

Estudo de caso - insere

60



main()

```
lista = insere(lista, 17);
```

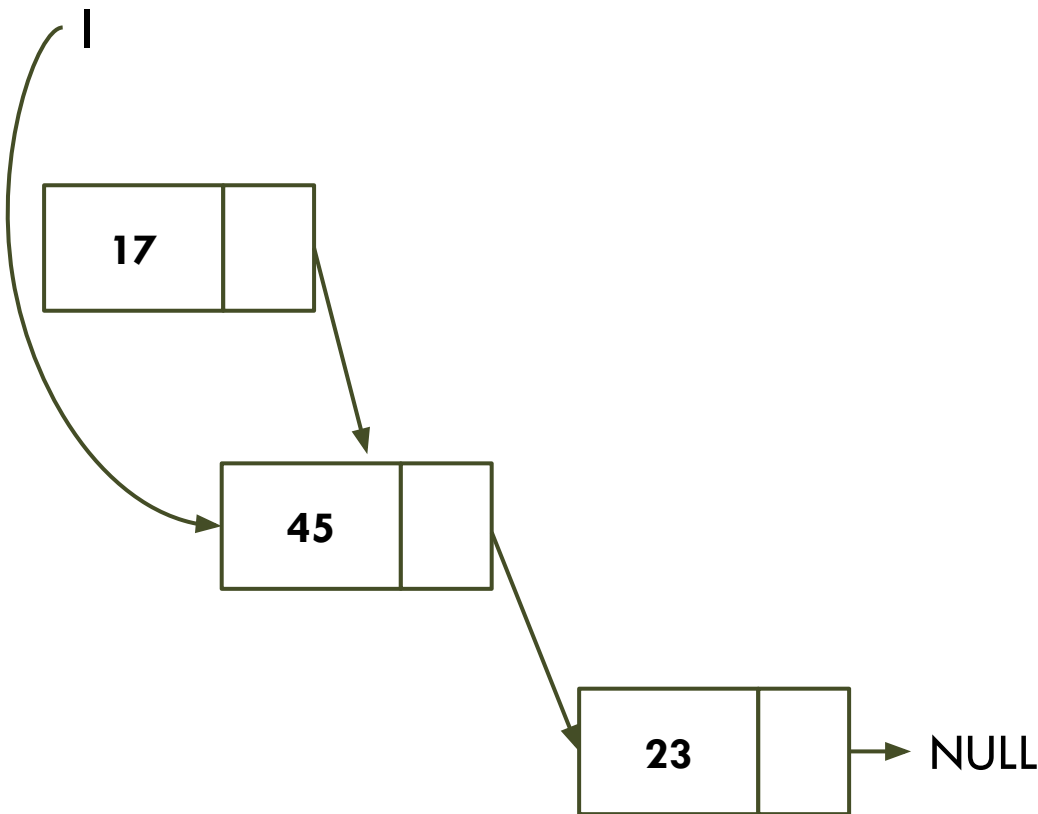
```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

Lista encadeada simples

Estudo de caso - insere

61



main()

```
lista = insere(lista, 17);
```

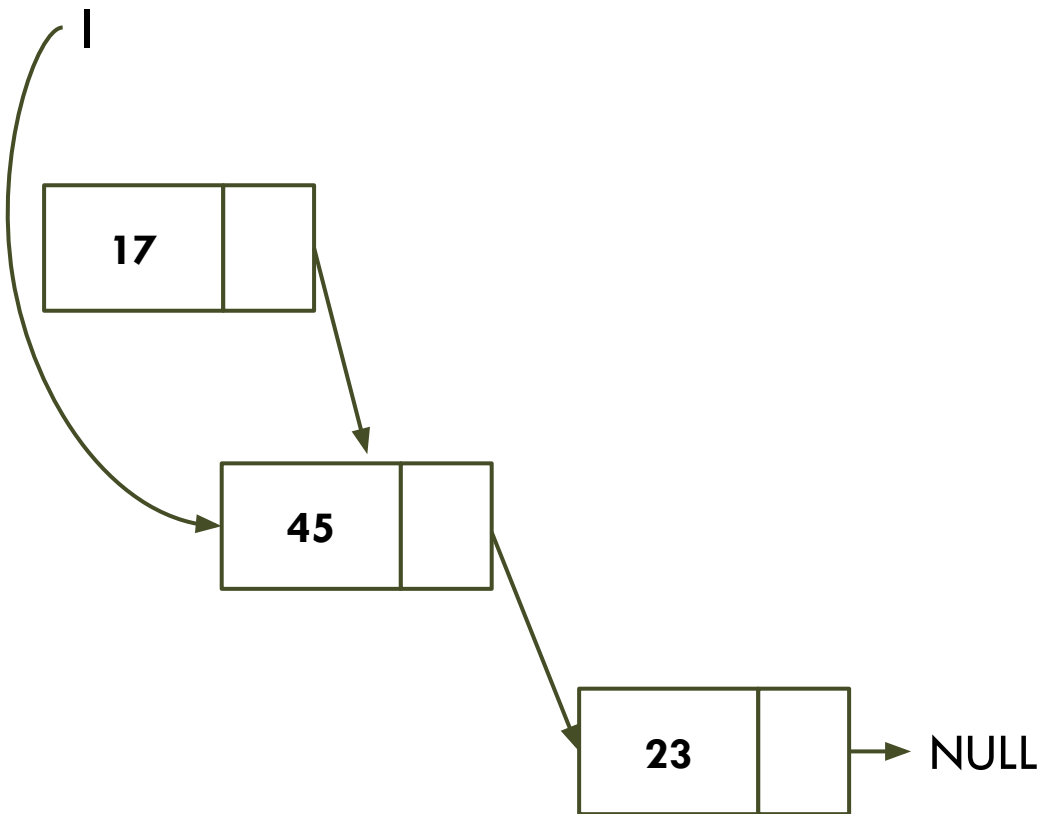
```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

Lista encadeada simples

Estudo de caso - insere

62



main()

```
lista = insere(lista, 17);
```

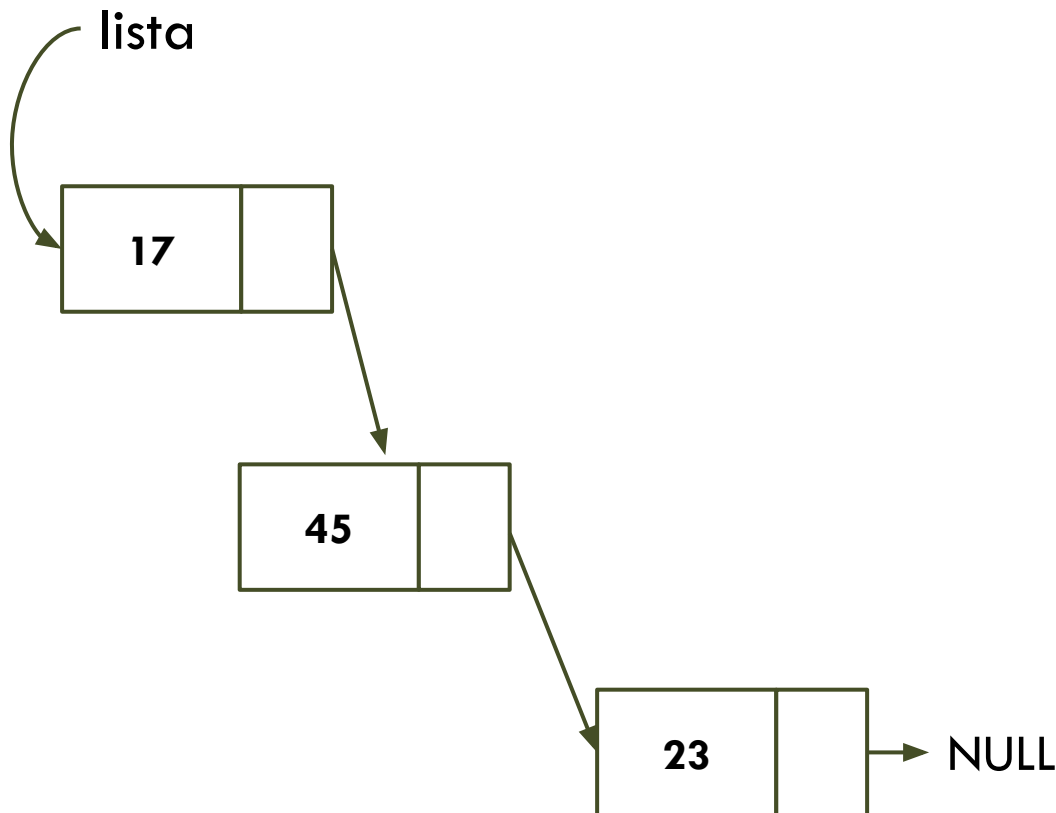
```
Lista* insere (Lista* l, int valor){  
    Lista* novo = alocaNo(valor);  
    novo->prox = l;  
    return novo;  
}
```

```
Lista* alocaNo(int valor){  
    Lista* no = (Lista*) malloc(sizeof(Lista));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

Lista encadeada simples

Estudo de caso - insere

63



main()

lista = insere(lista, 17);

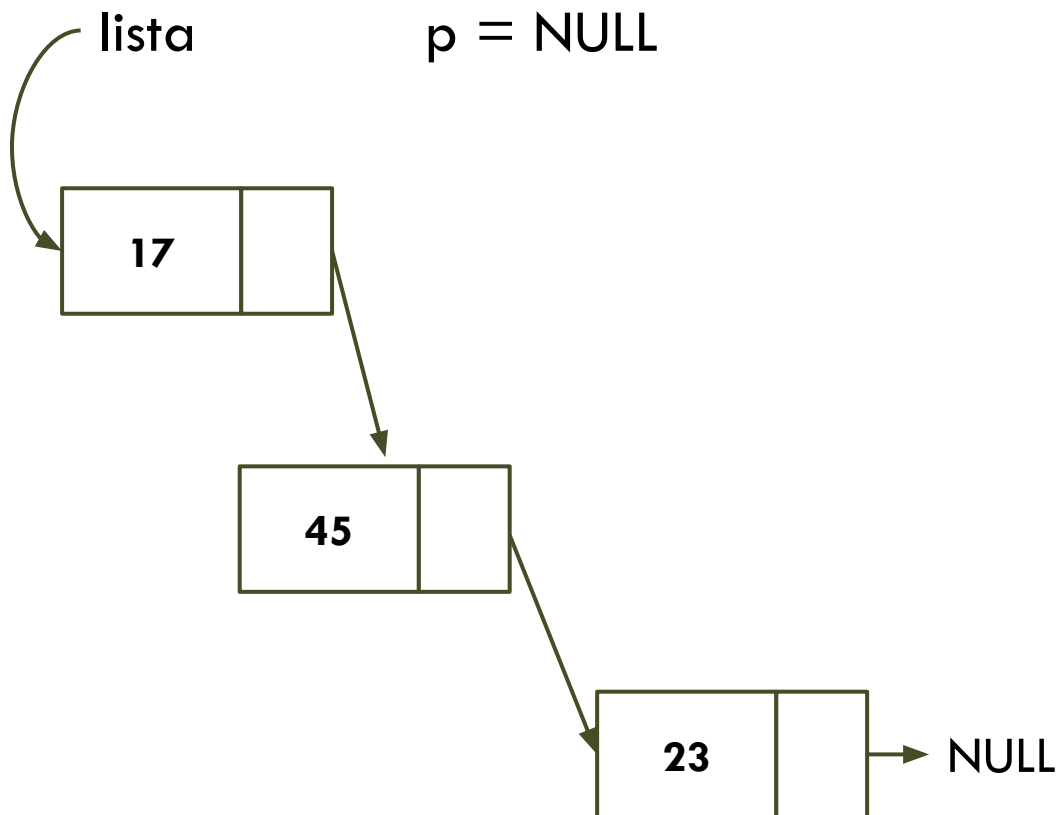
```
Lista* insere (Lista* l, int valor){
    Lista* novo = alocaNo(valor);
    novo->prox = l;
    return novo;
}
```

```
Lista* alocaNo(int valor){
    Lista* no = (Lista*) malloc(sizeof(Lista));
    no->info = valor;
    no->prox = NULL;
    return no;
}
```

Lista encadeada simples

Estudo de caso - imprime

64



main()

imprime(lista);

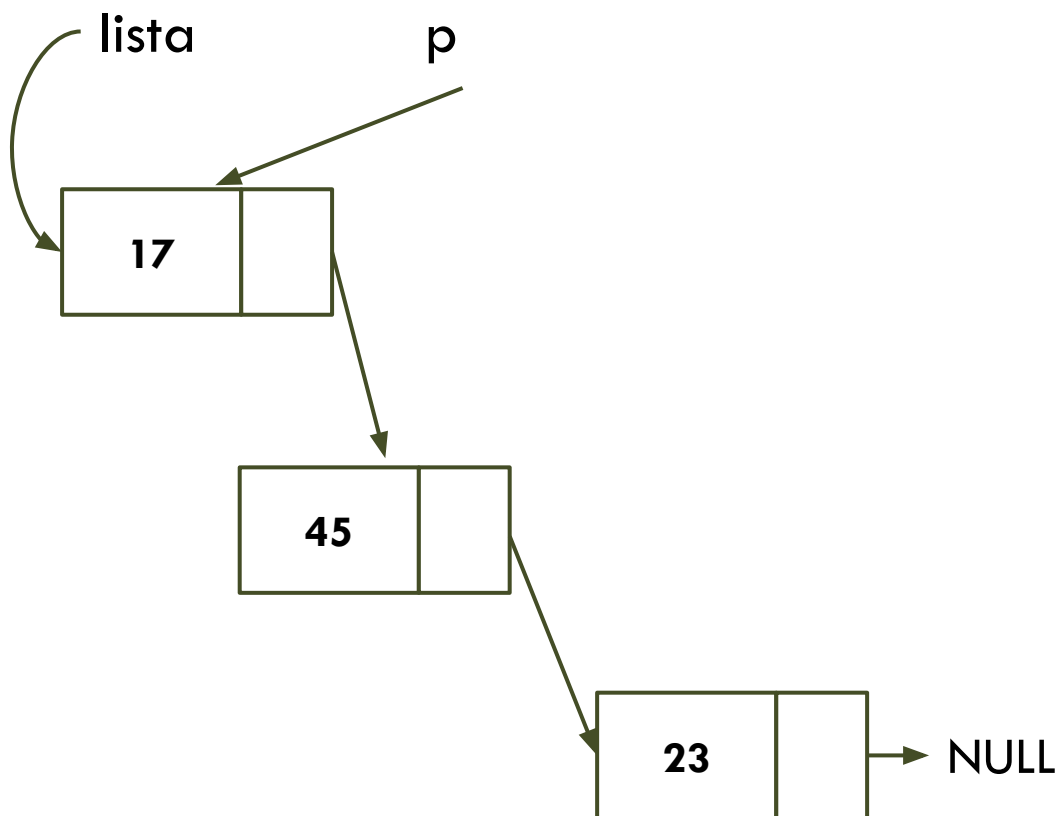
```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p!= NULL; p = p->prox){  
        printf(" %d ",p->info);  
    }  
}
```

Saída:

Lista encadeada simples

Estudo de caso - imprime

65



main()

imprime(lista);

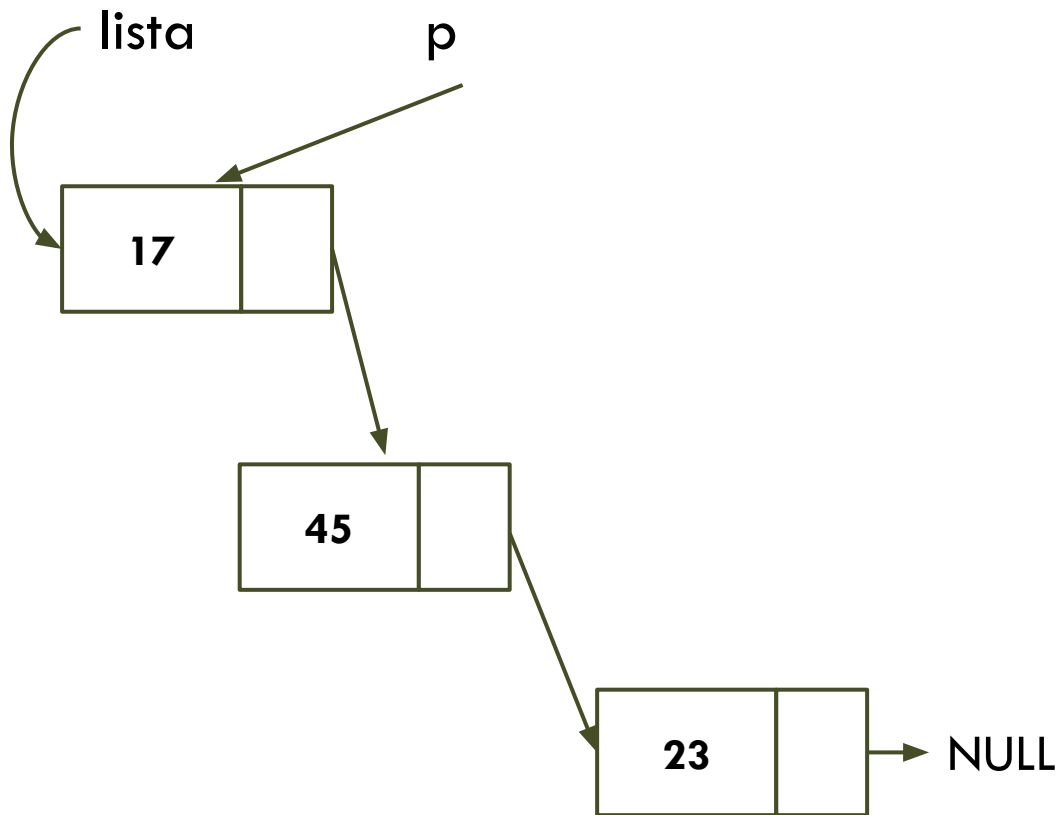
```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p!= NULL; p = p->prox){  
        printf(" %d ",p->info);  
    }  
}
```

Saída:

Lista encadeada simples

Estudo de caso - imprime

66



main()

imprime(lista);

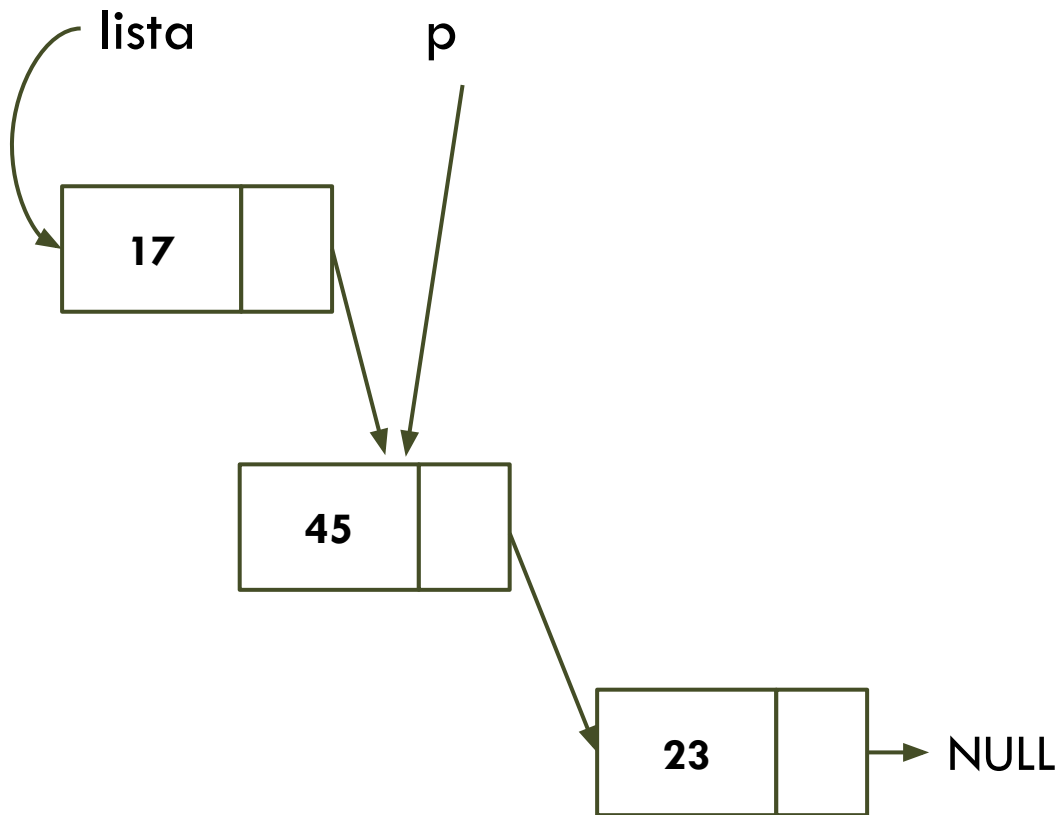
```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p!= NULL; p = p->prox){  
        printf(" %d ",p->info);  
    }  
}
```

Saída: 17

Lista encadeada simples

Estudo de caso - imprime

67



main()

imprime(lista);

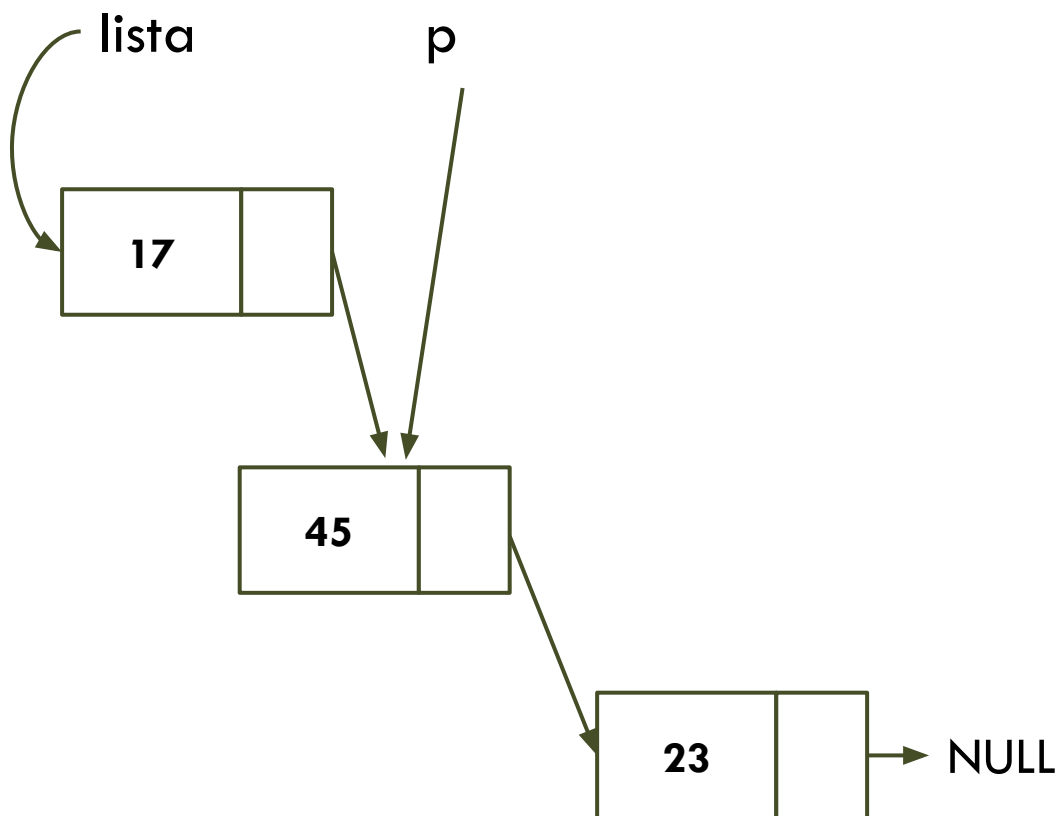
```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p!= NULL; p = p->prox){  
        printf(" %d ",p->info);  
    }  
}
```

Saída: 17

Lista encadeada simples

Estudo de caso - imprime

68



main()

imprime(lista);

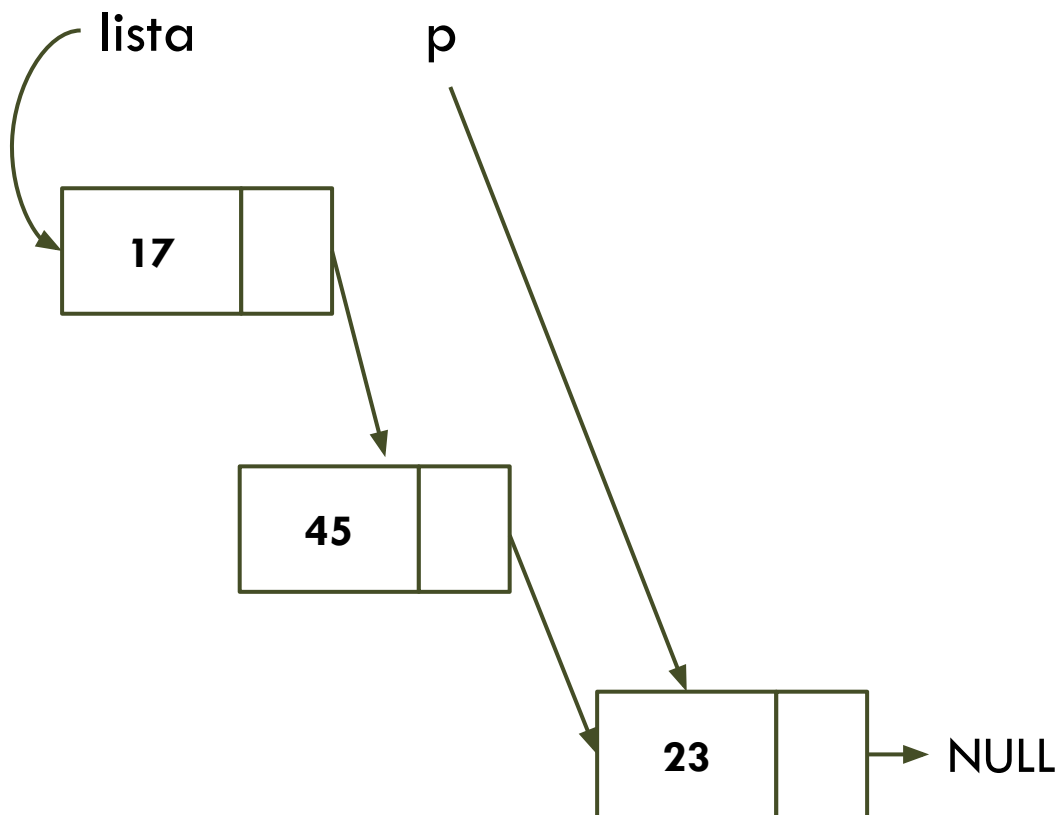
```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p!= NULL; p = p->prox){  
        printf(" %d ",p->info);  
    }  
}
```

Saída: 17 45

Lista encadeada simples

Estudo de caso - imprime

69



main()

imprime(lista);

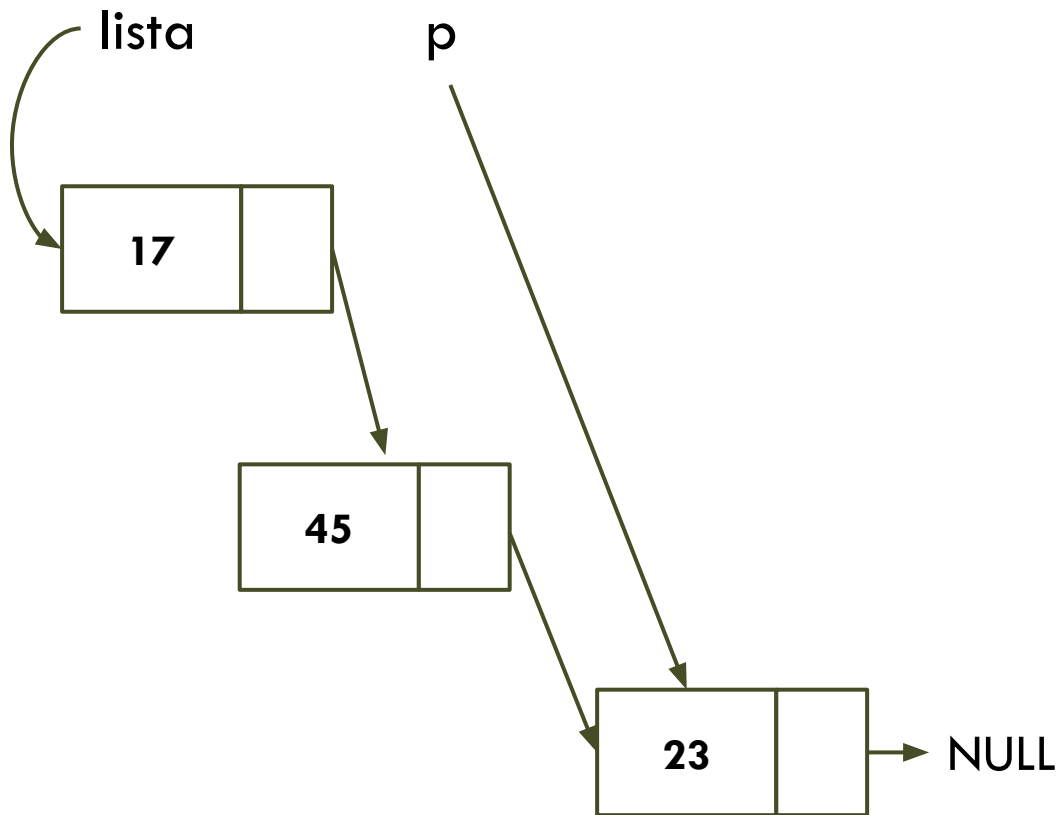
```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p!= NULL; p = p->prox){  
        printf(" %d ",p->info);  
    }  
}
```

Saída: 17 45

Lista encadeada simples

Estudo de caso - imprime

70



main()

imprime(lista);

```
void imprime(Lista* raiz){  
    Lista* p = NULL;  
    for (p = raiz; p!= NULL; p = p->prox){  
        printf(" %d ",p->info);  
    }  
}
```

Saída: 17 45 23

Lista encadeada simples

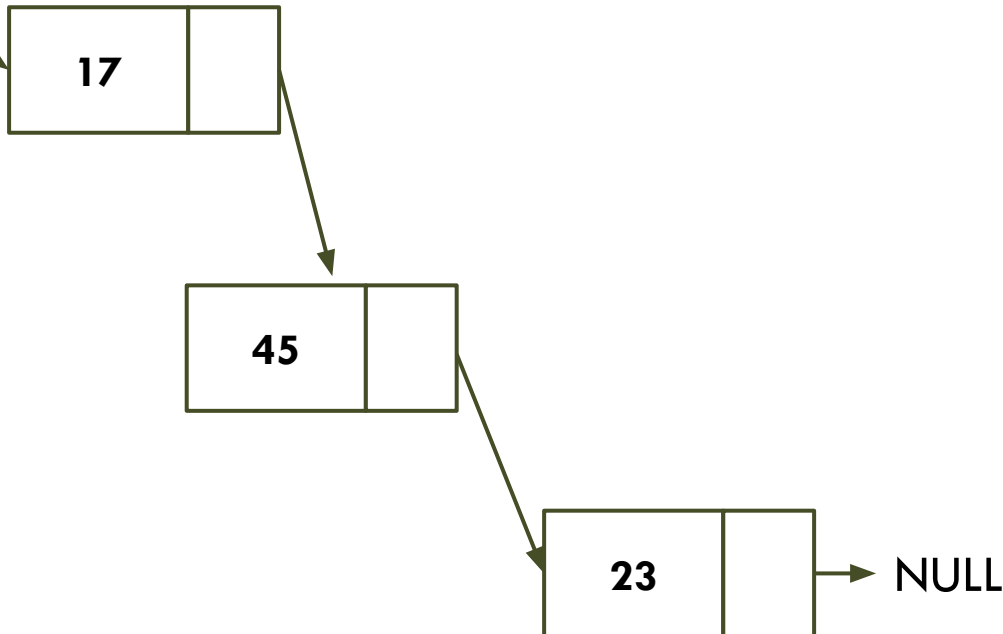
Estudo de caso - remove elemento do início

71

lista = #NO17

main()

lista = retira(lista, 17);

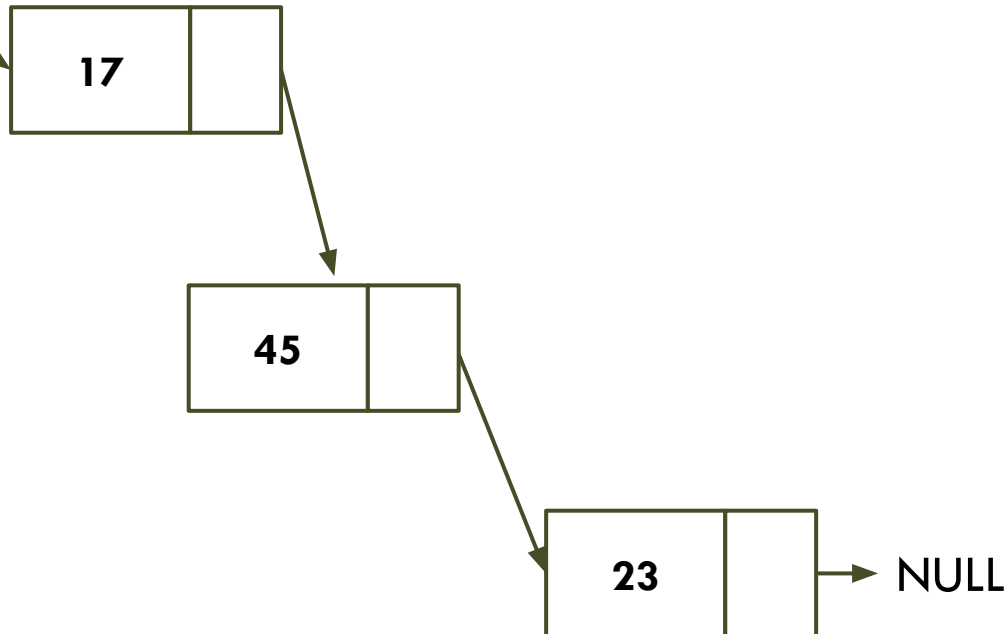


Lista encadeada simples

Estudo de caso - remove elemento do início

72

lista = #NO17



main()

lista = retira(lista, 17);

```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```


Lista encadeada simples

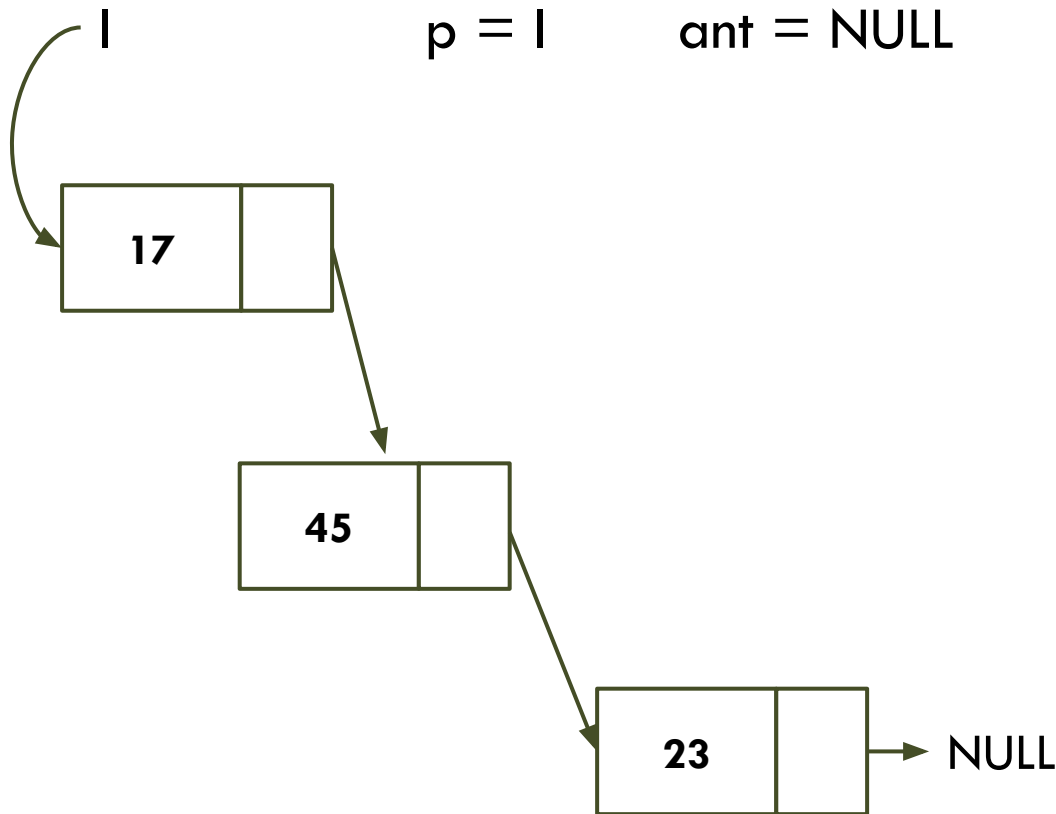
Estudo de caso - remove elemento do início

73

lista = #NO17

p = l

ant = NULL



main()

lista = retira(lista, 17);

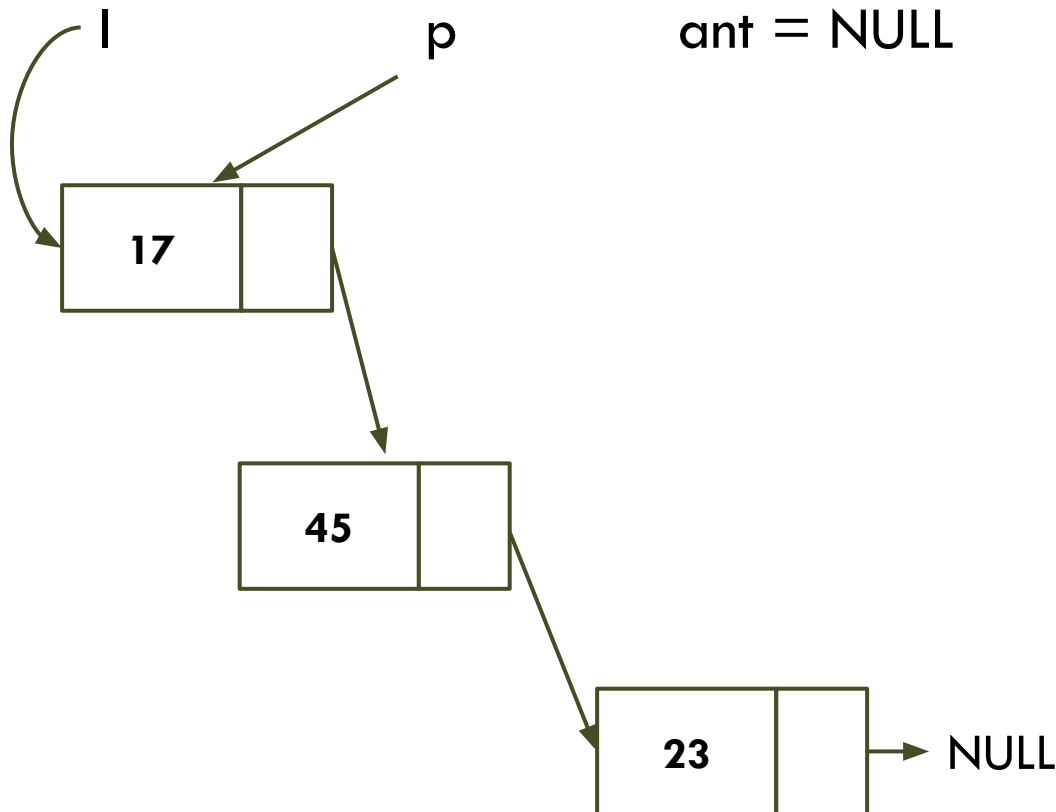
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do início

74

lista = #NO17



main()

lista = retira(lista, 17);

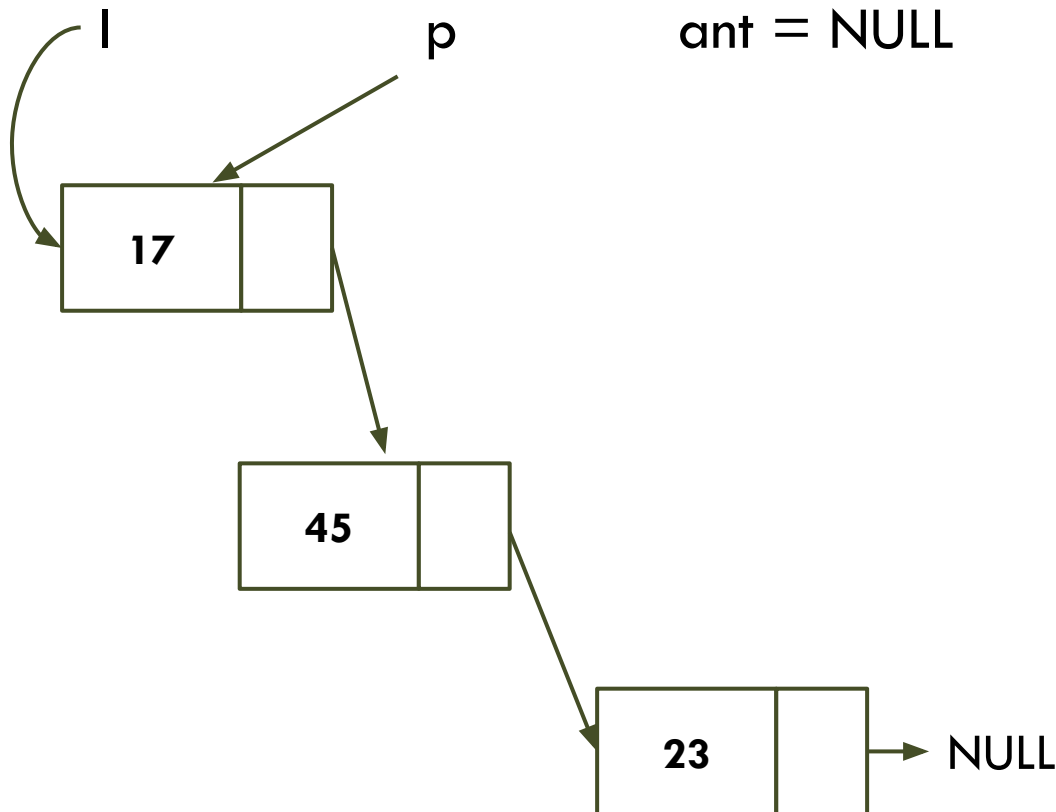
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do início

75

lista = #NO17



main()

lista = retira(lista, 17);

```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

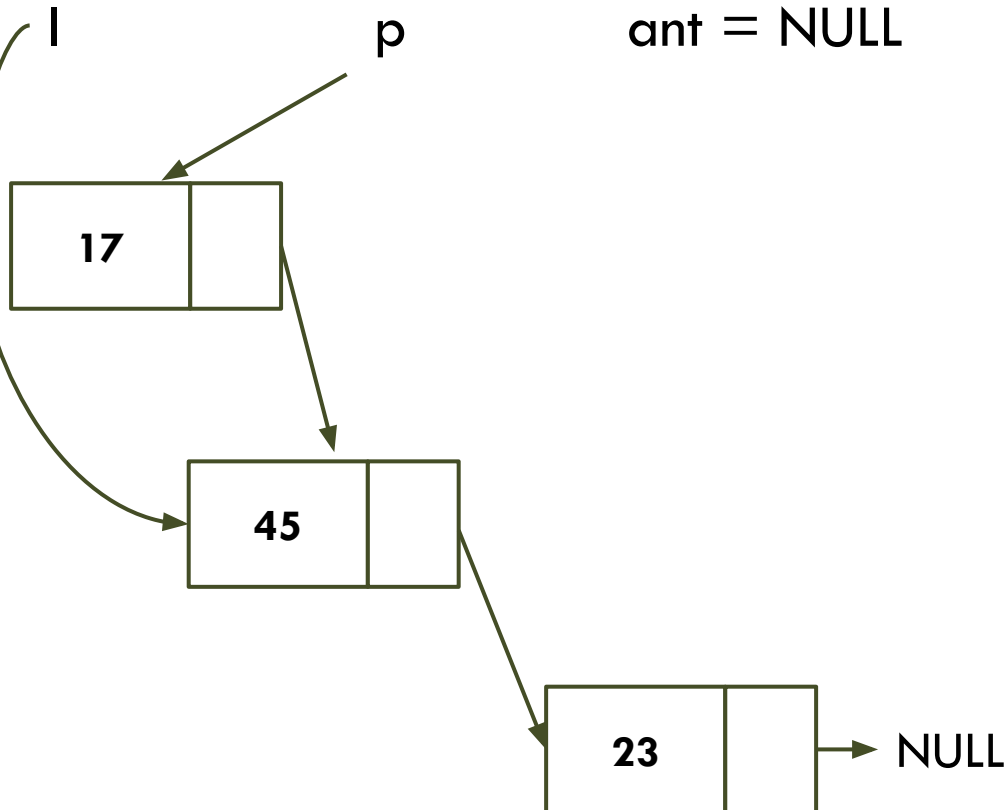
Estudo de caso - remove elemento do início

76

lista = #NO17

p

ant = NULL



main()

lista = retira(lista, 17);

```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

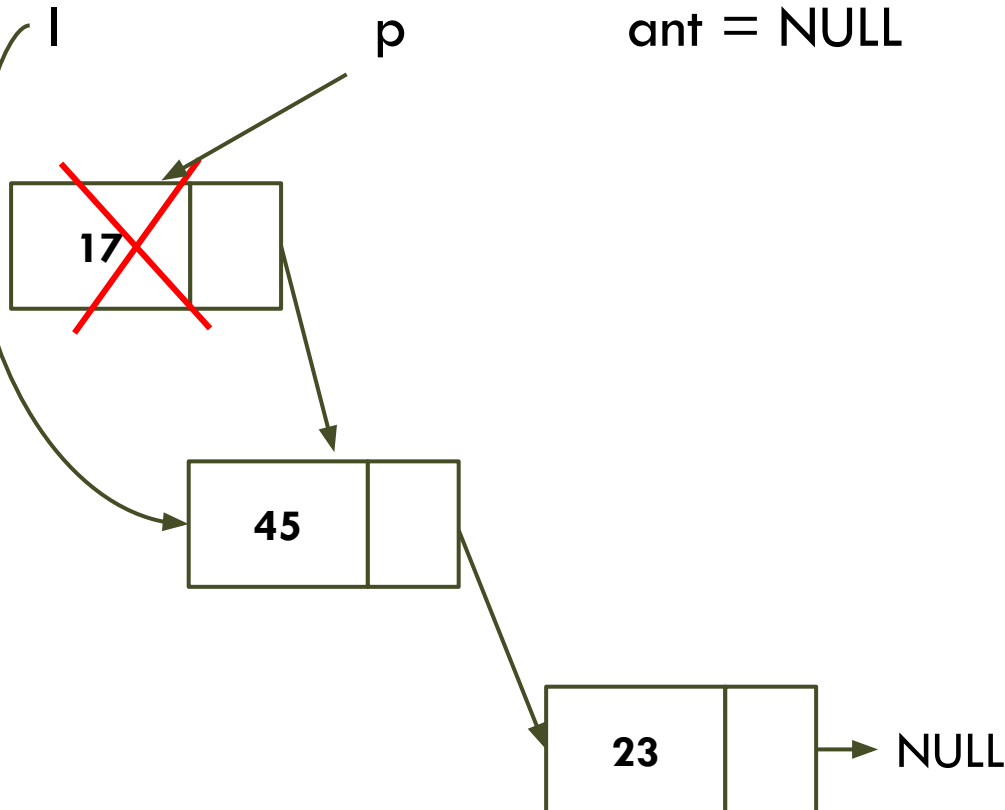
Estudo de caso - remove elemento do início

77

lista = #NO17

p

ant = NULL



main()

lista = retira(lista, 17);

```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

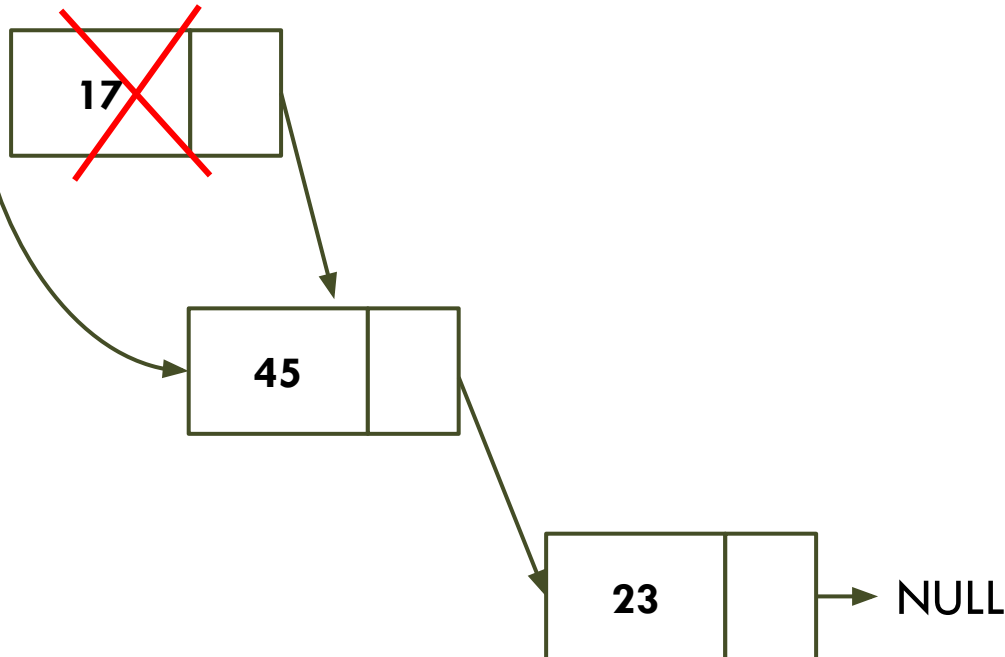
Estudo de caso - remove elemento do início

78

lista = #NO17

p

ant = NULL



main()

lista = retira(lista, 17);

```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

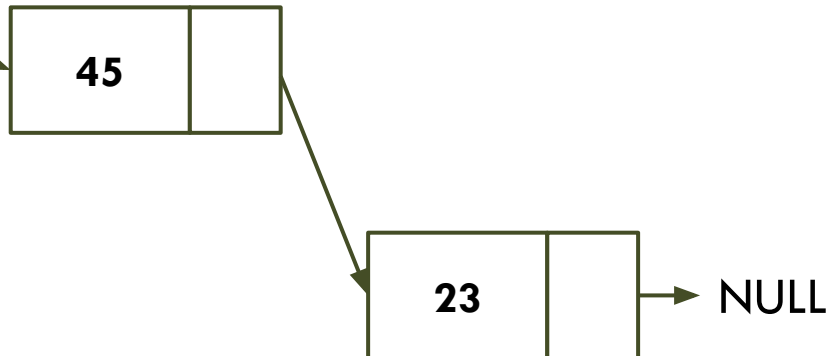
Estudo de caso - remove elemento do início

79

lista = #NO45

main()

lista = retira(lista, 17);



Lista encadeada simples

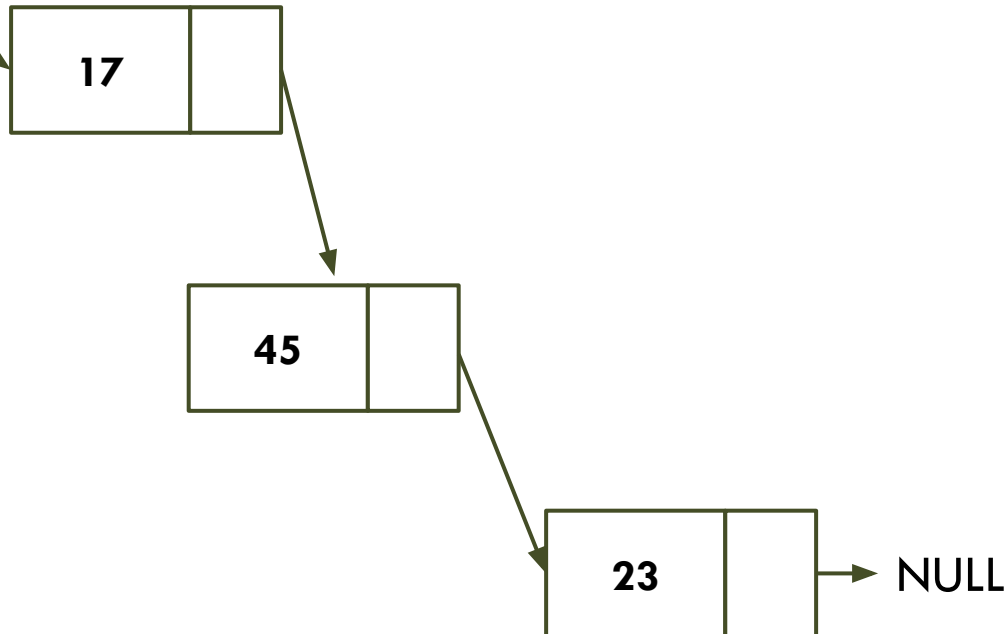
Estudo de caso - remove elemento do meio

80

lista = #NO17

main()

lista = retira(lista, 45);

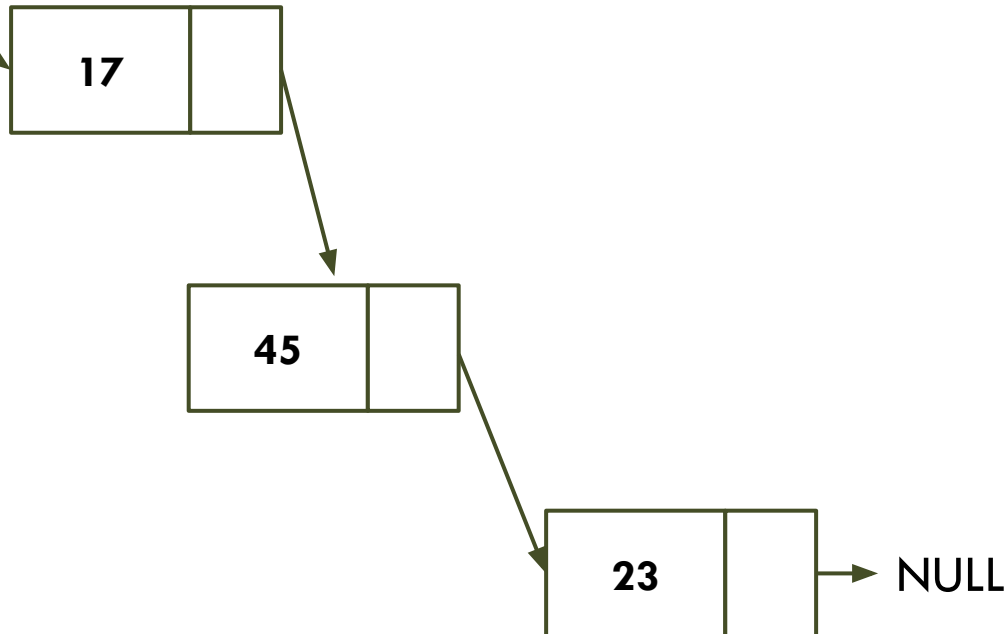


Lista encadeada simples

Estudo de caso - remove elemento do meio

81

lista = #NO17



main()

lista = retira(lista, 45);

```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

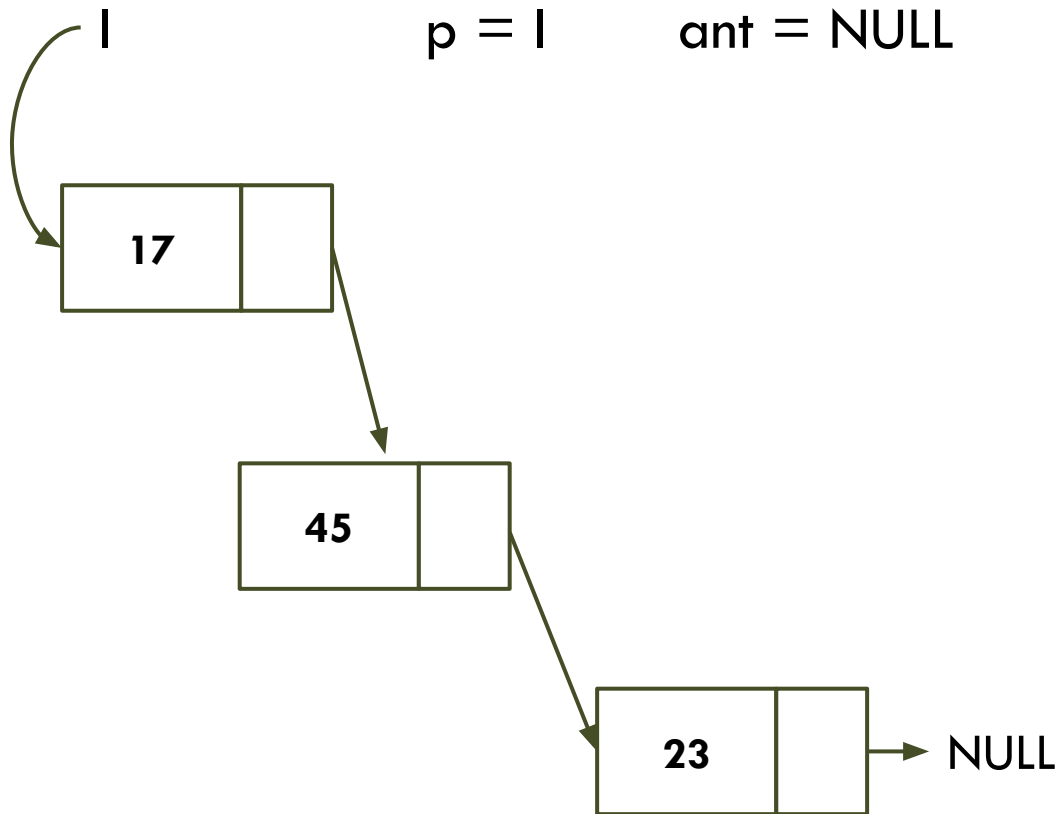
Estudo de caso - remove elemento do meio

82

lista = #NO17

p = l

ant = NULL



main()

lista = retira(lista, 45);

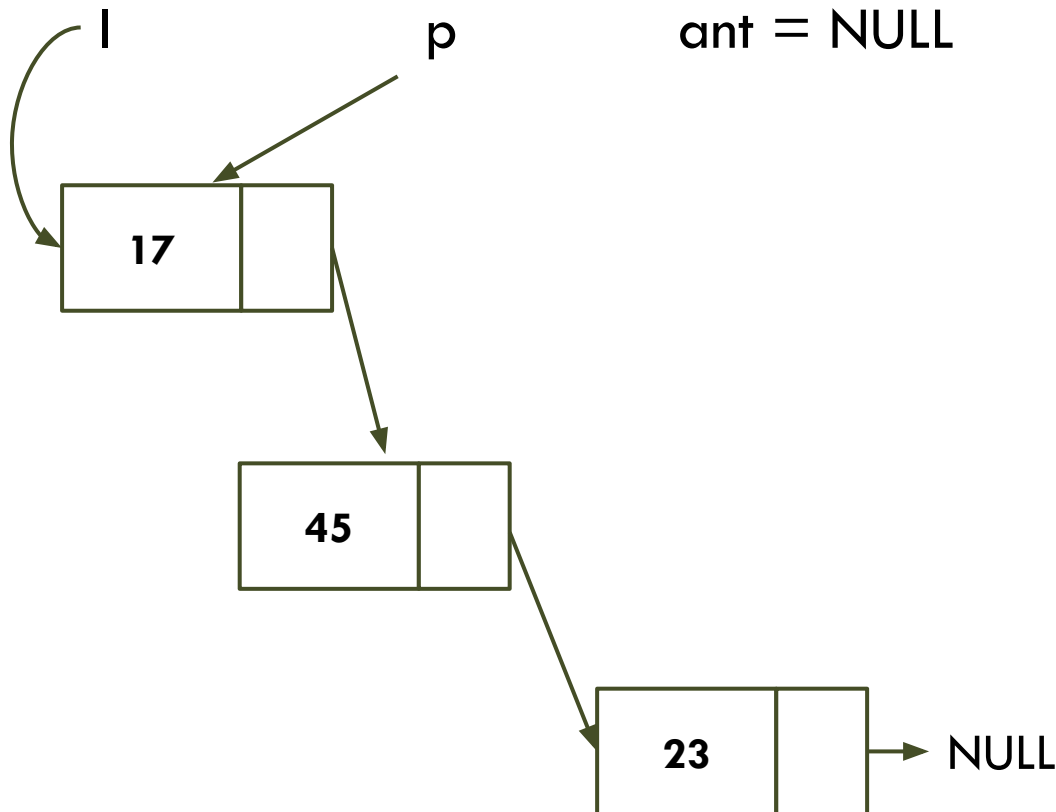
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do meio

83

lista = #NO17



main()

lista = retira(lista, 45);

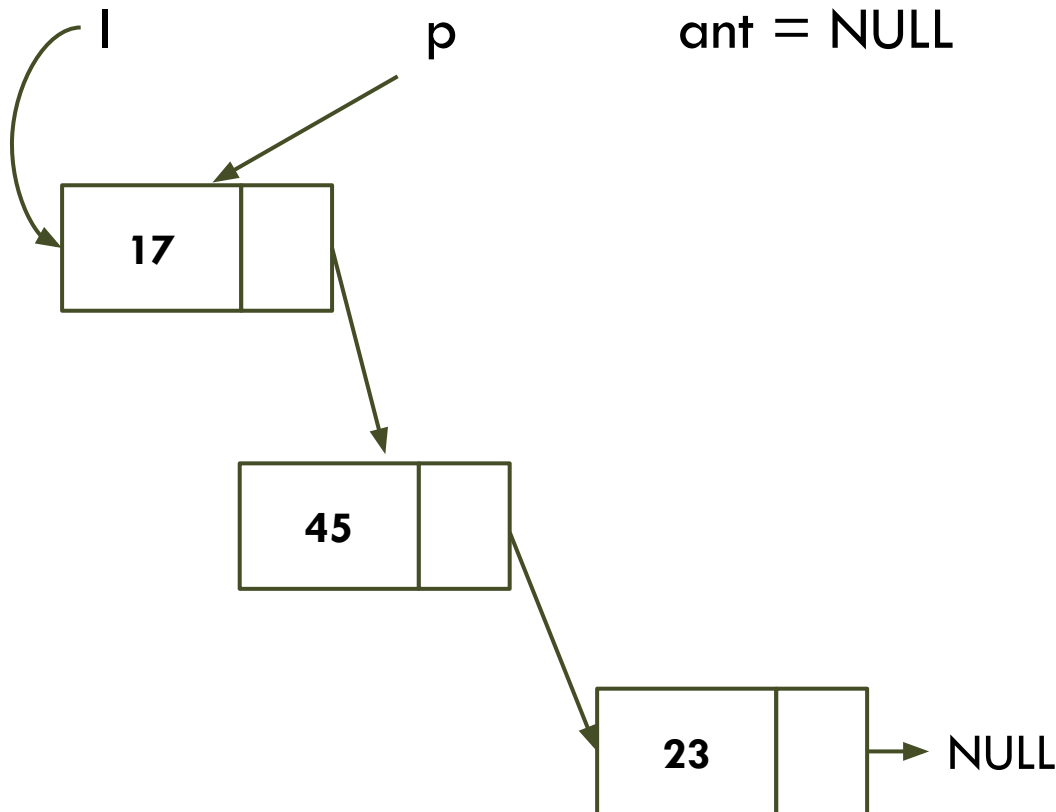
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do meio

84

lista = #NO17



main()

lista = retira(lista, 45);

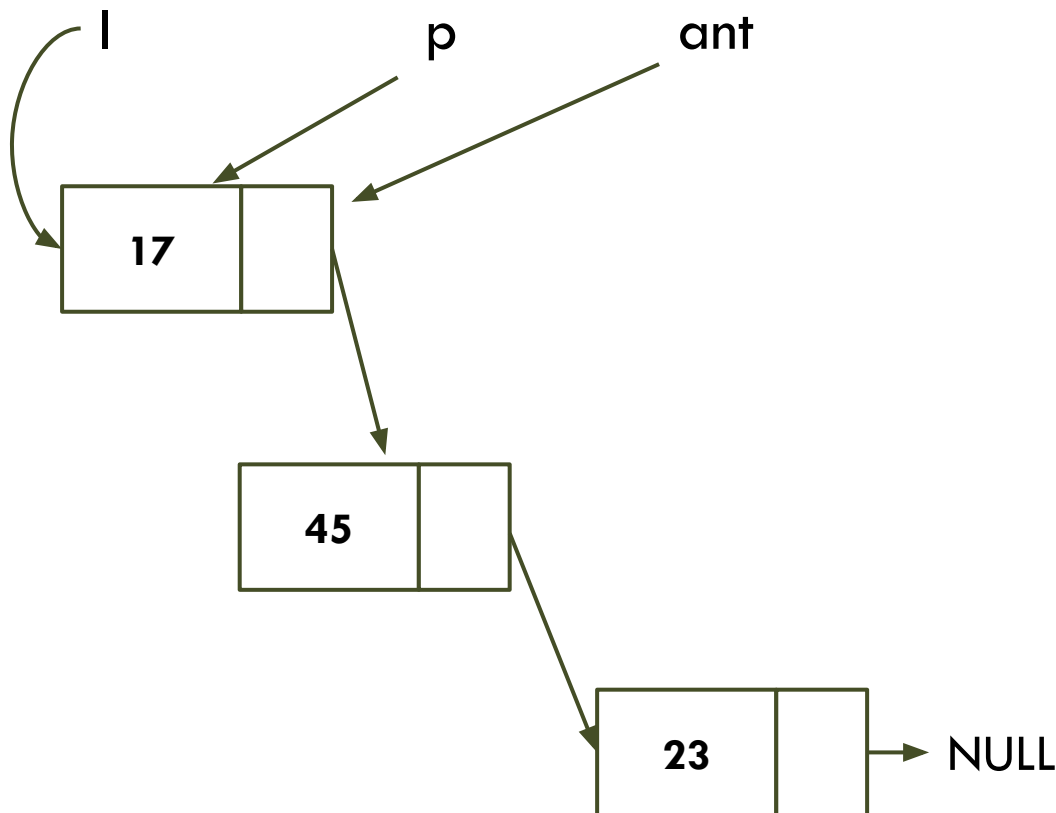
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do meio

85

lista = #NO17



main()

lista = retira(lista, 45);

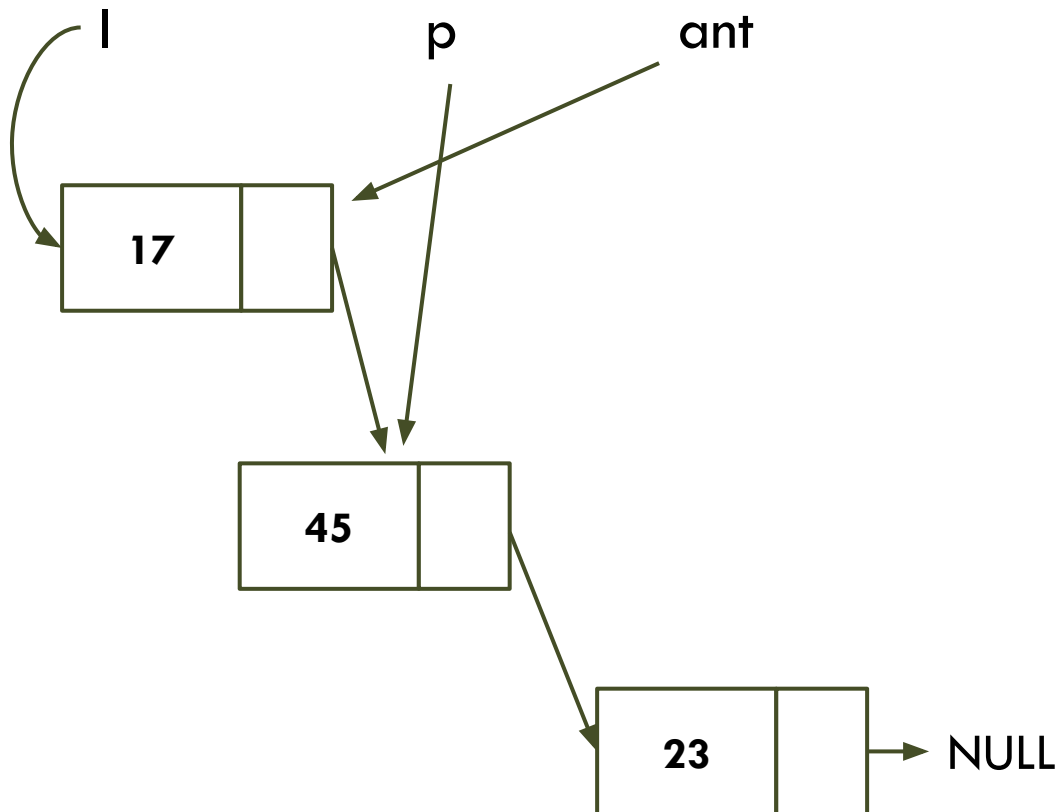
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do meio

86

lista = #NO17



main()

lista = retira(lista, 45);

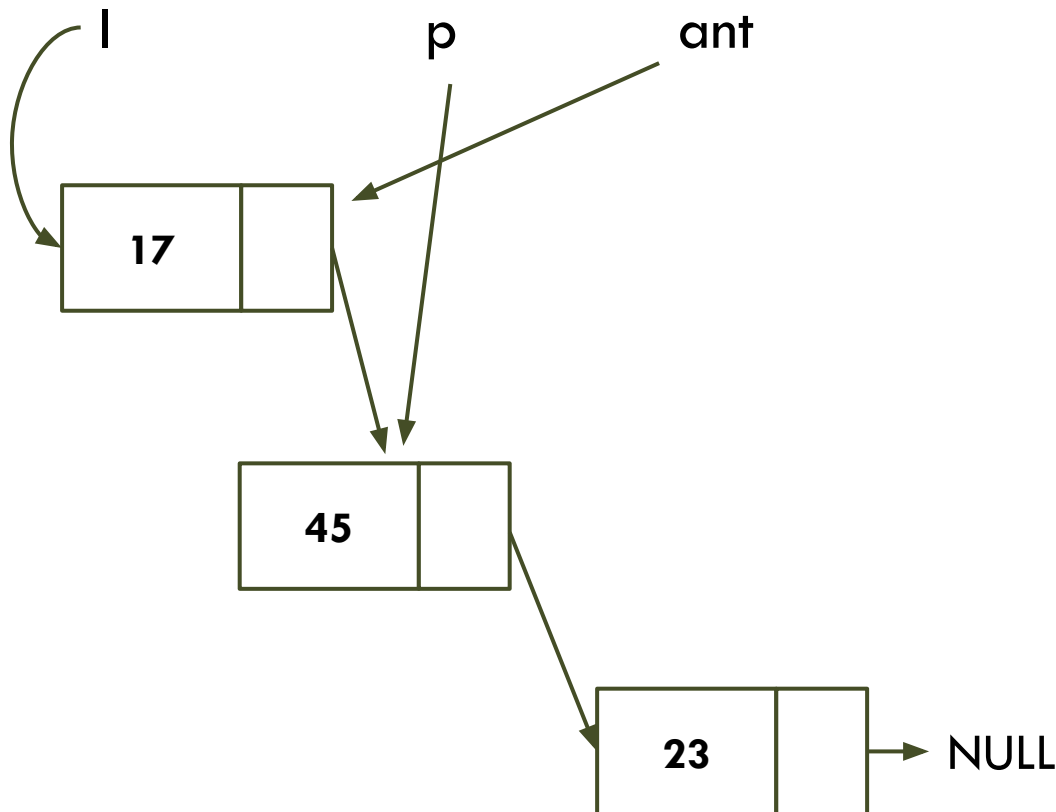
```
Lista* retira (Lista* l, int v){  
    Lista* ant = NULL;  
    Lista* p = l;  
    for (p=l; p!=NULL; p = p->prox) {  
        if (p->info == v){  
            break;  
        }  
        ant = p;  
    }  
    if (p == NULL)  
        return l;  
    if (ant == NULL){  
        l = p->prox;  
    }else {  
        ant->prox = p->prox;  
    }  
    free(p);  
    return l;  
}
```

Lista encadeada simples

Estudo de caso - remove elemento do meio

87

lista = #NO17



main()

lista = retira(lista, 45);

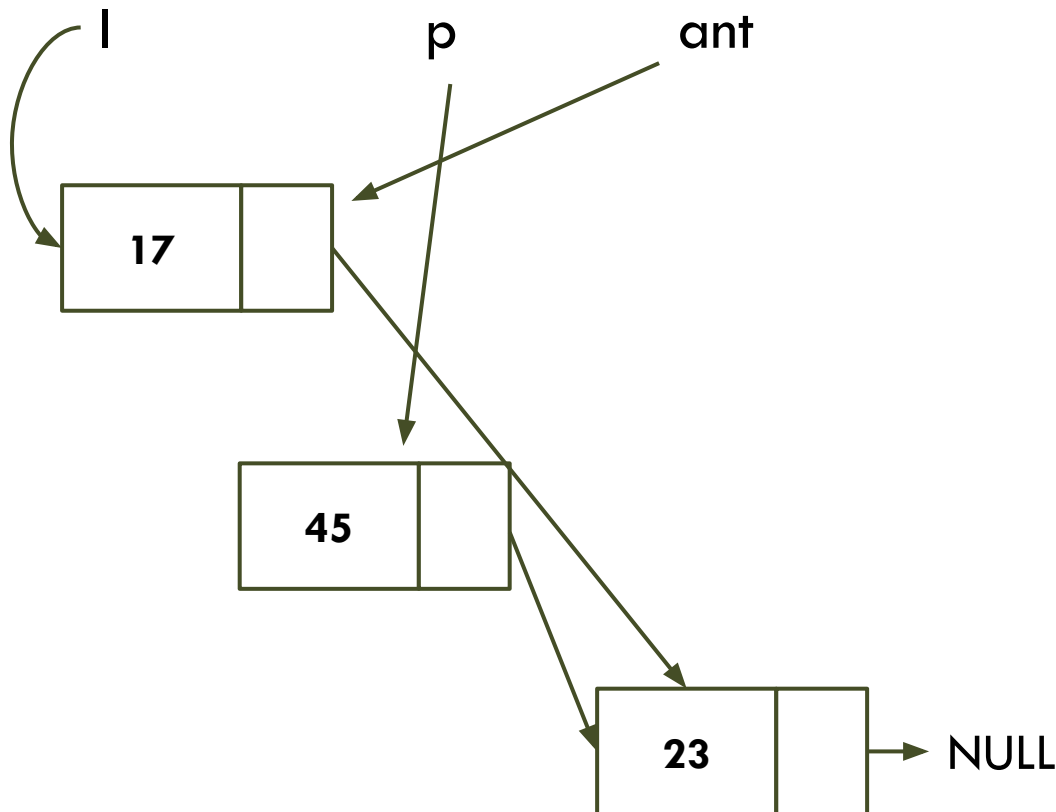
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do meio

88

lista = #NO17



main()

lista = retira(lista, 45);

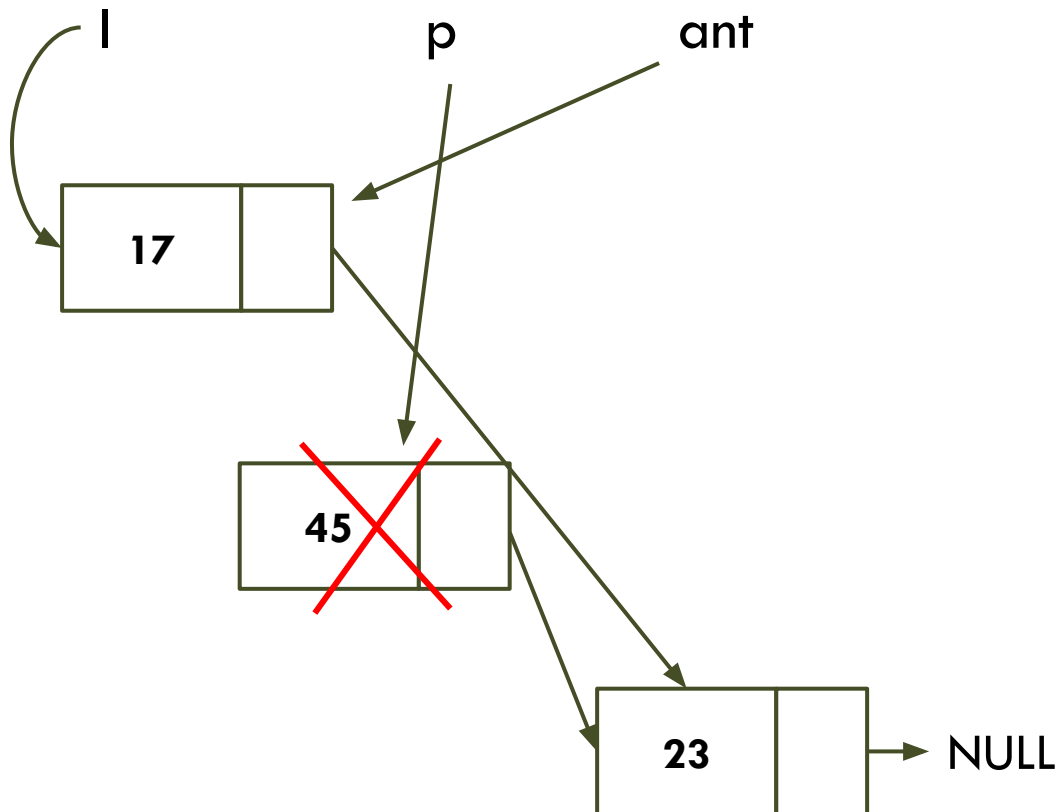
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```


Lista encadeada simples

Estudo de caso - remove elemento do meio

89

lista = #NO17



main()

lista = retira(lista, 45);

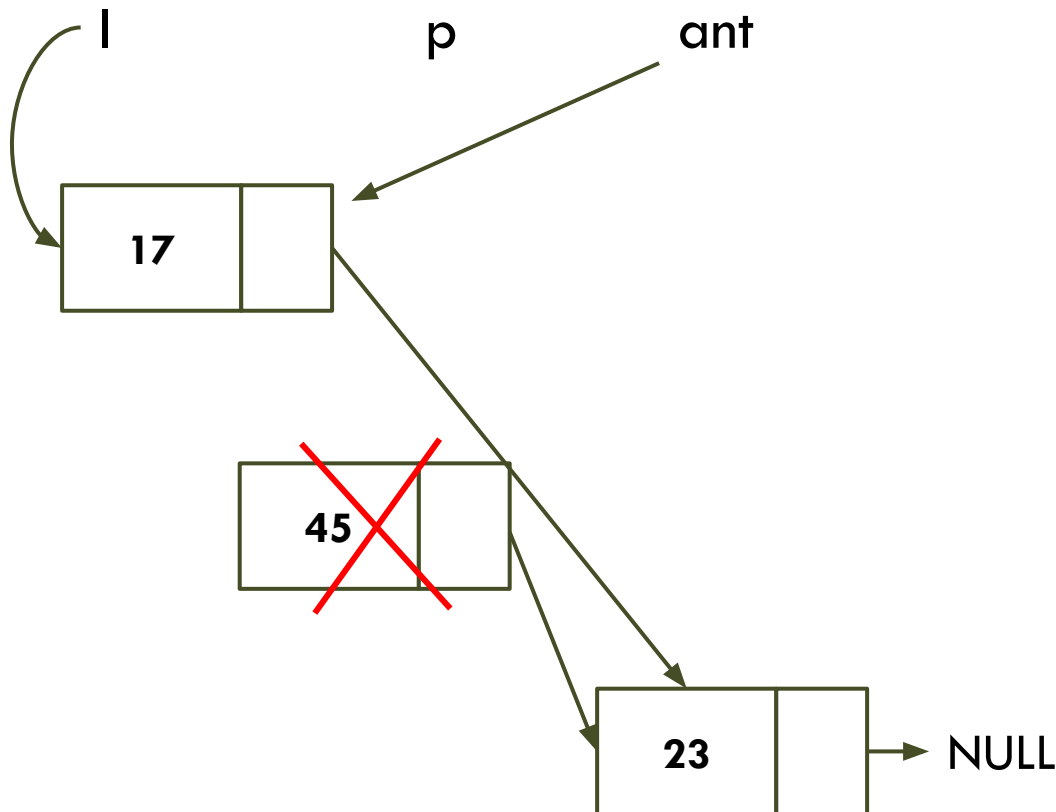
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do meio

90

lista = #NO17



main()

lista = retira(lista, 45);

```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

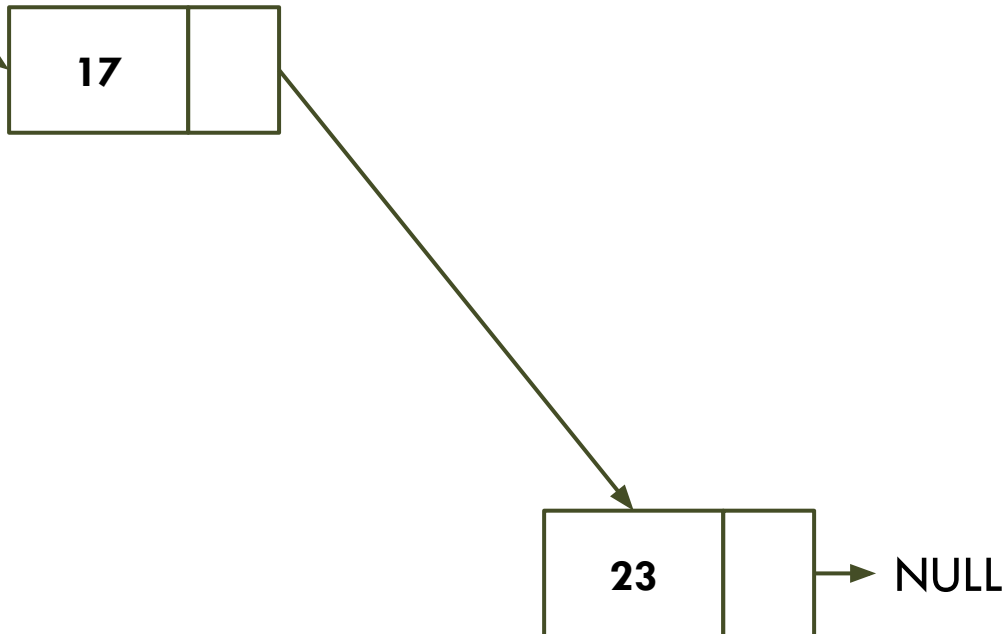
Estudo de caso - remove elemento do meio

91

lista = #NO17

main()

lista = retira(lista, 45);



Lista encadeada simples

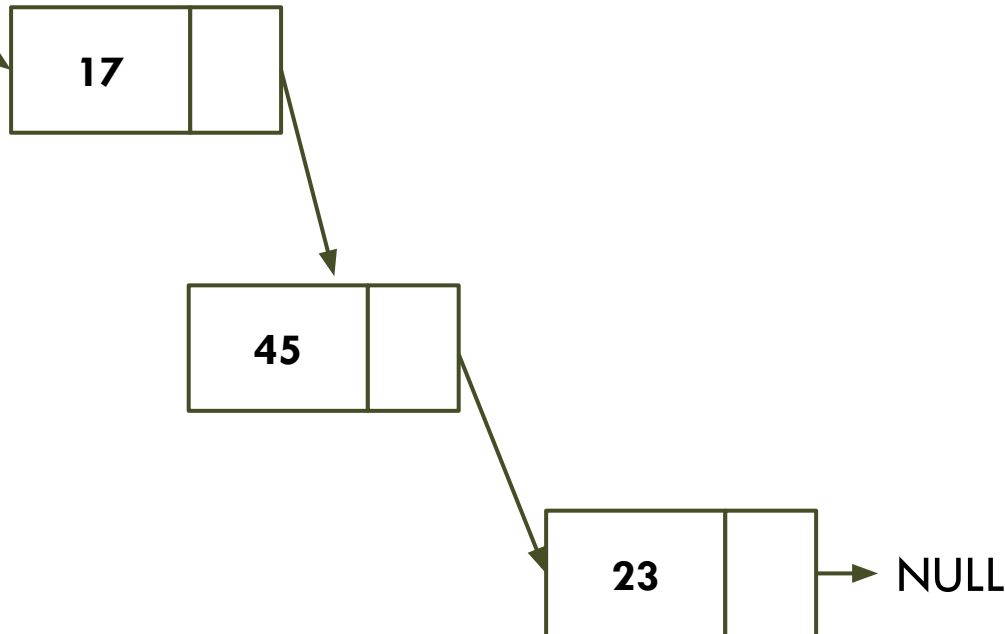
Estudo de caso - remove elemento do fim

92

lista = #NO17

main()

lista = retira(lista, 23);

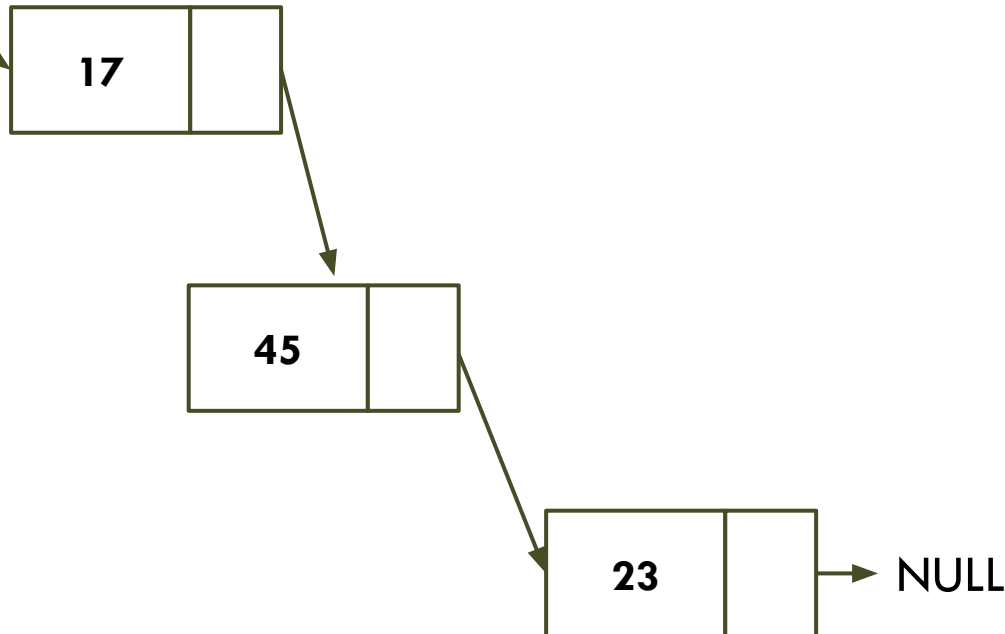


Lista encadeada simples

Estudo de caso - remove elemento do fim

93

lista = #NO17



main()

lista = retira(lista, 23);

```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

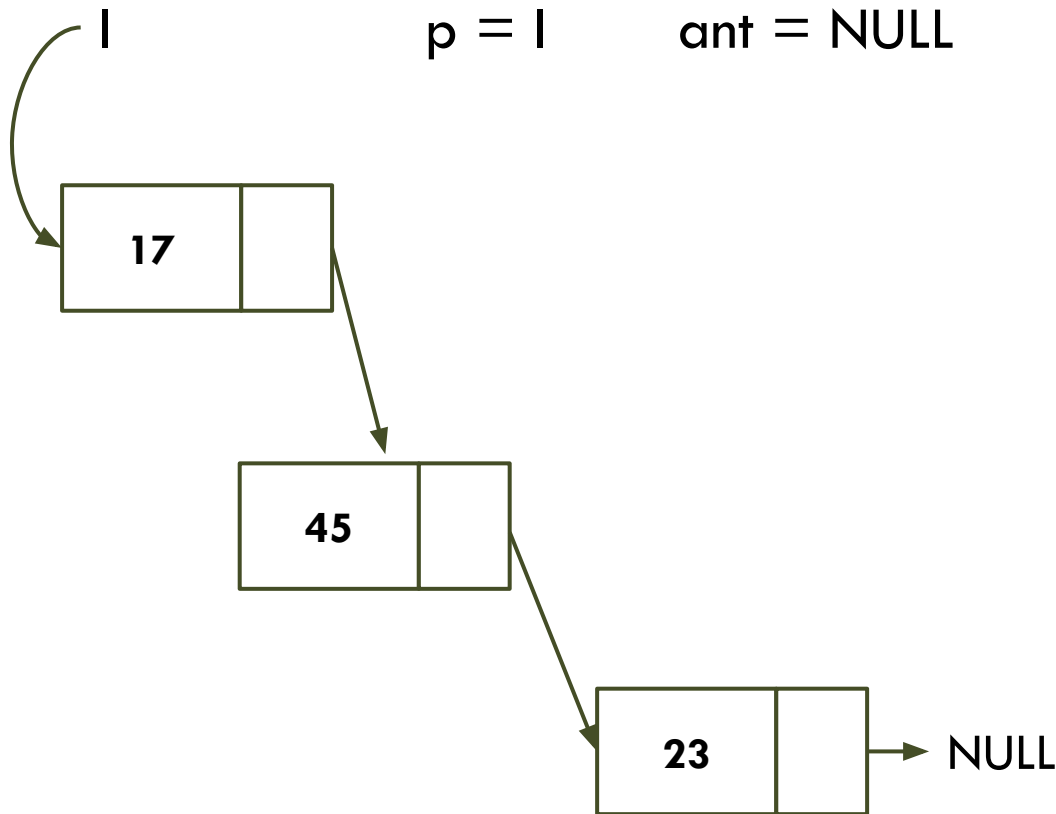
Estudo de caso - remove elemento do fim

94

lista = #NO17

p = l

ant = NULL



main()

lista = retira(lista, 23);

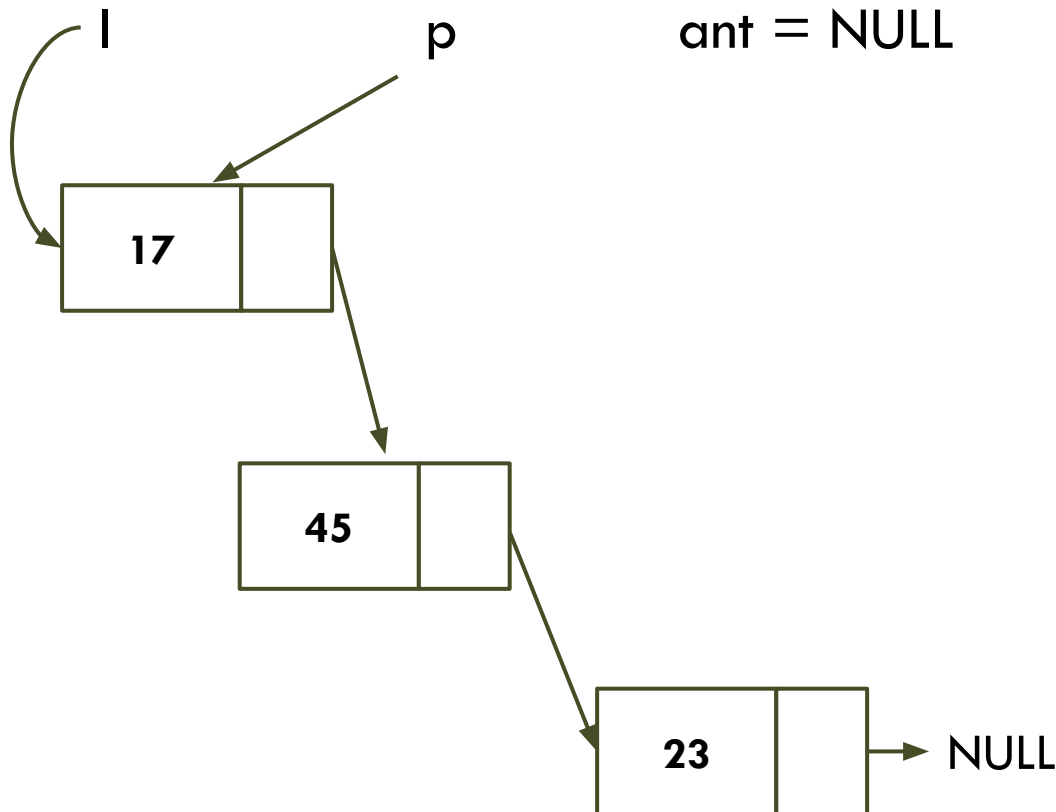
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do fim

95

lista = #NO17



main()

lista = retira(lista, 23);

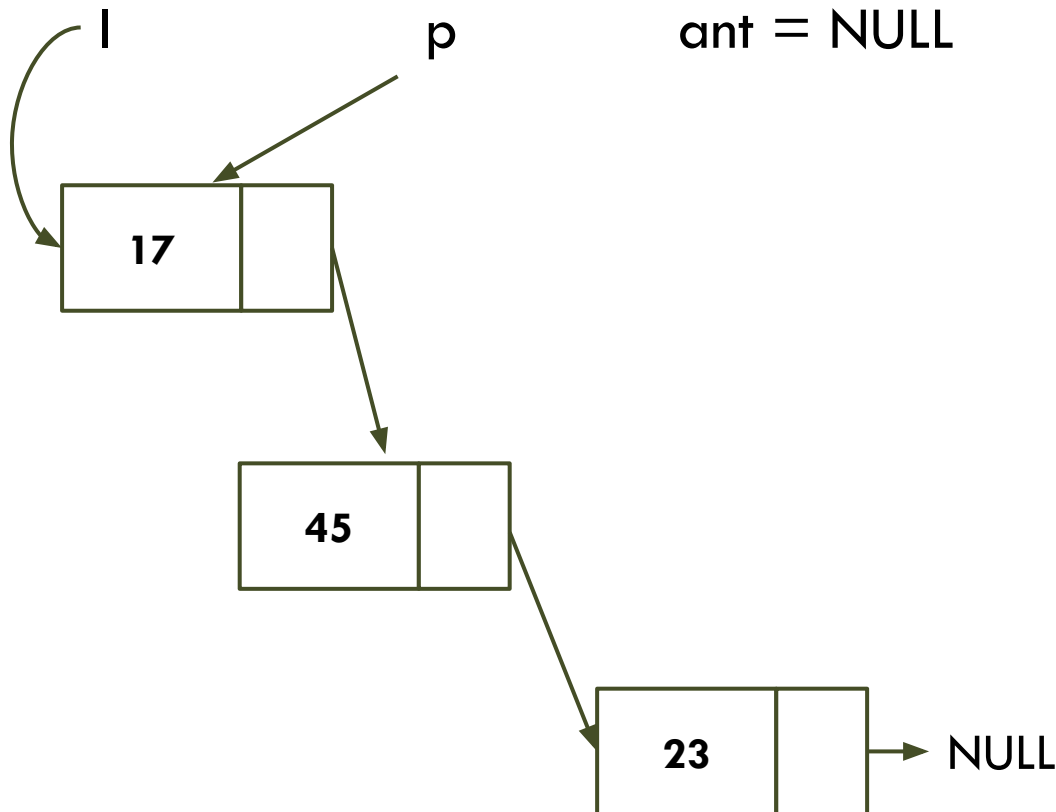
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do fim

96

lista = #NO17



main()

lista = retira(lista, 23);

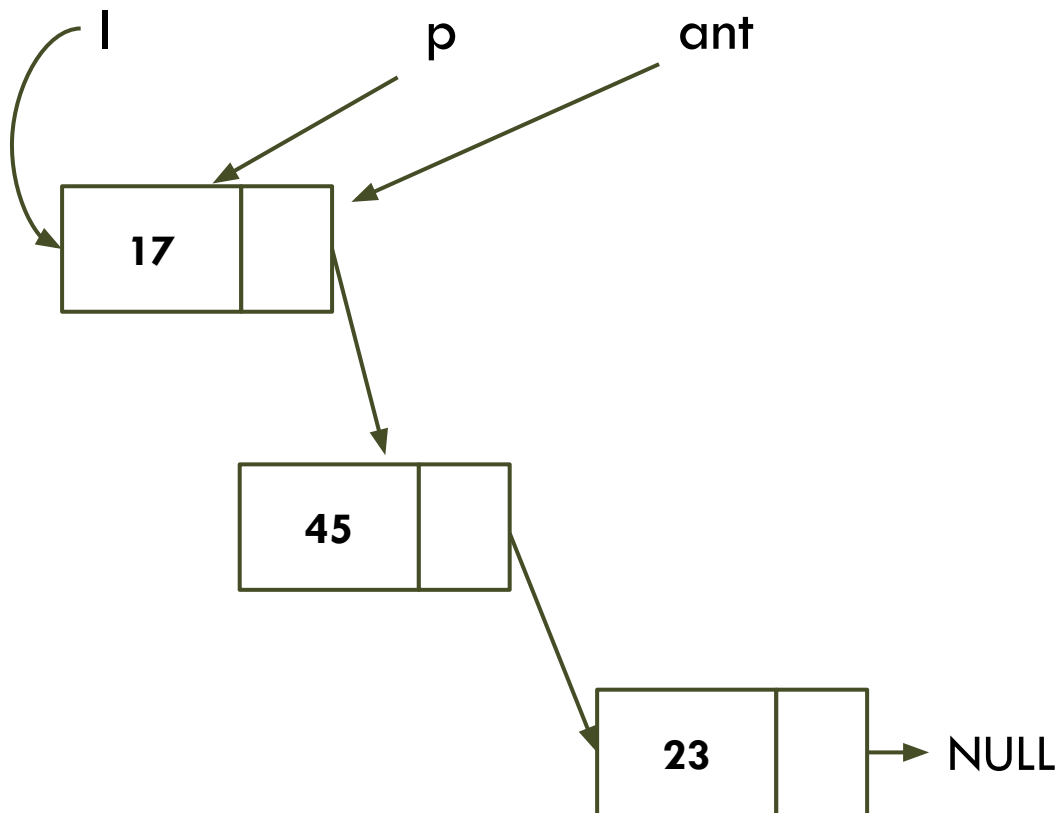
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```


Lista encadeada simples

Estudo de caso - remove elemento do fim

97

lista = #NO17



main()

lista = retira(lista, 23);

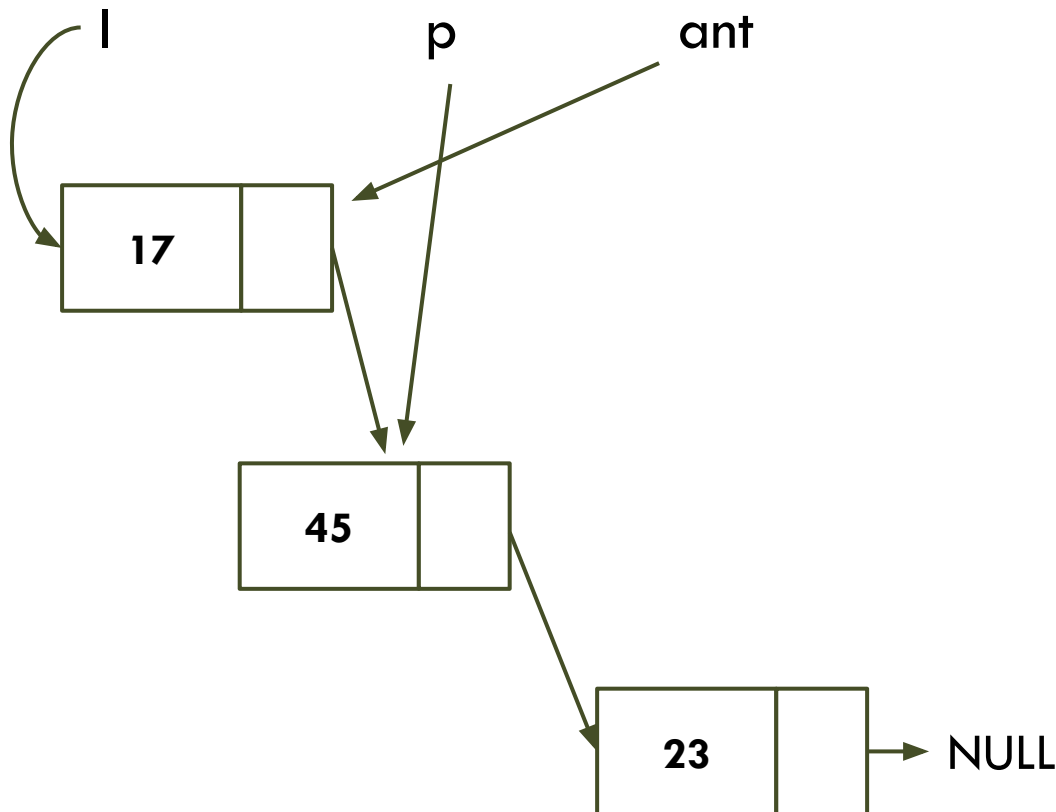
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do fim

98

lista = #NO17



main()

lista = retira(lista, 23);

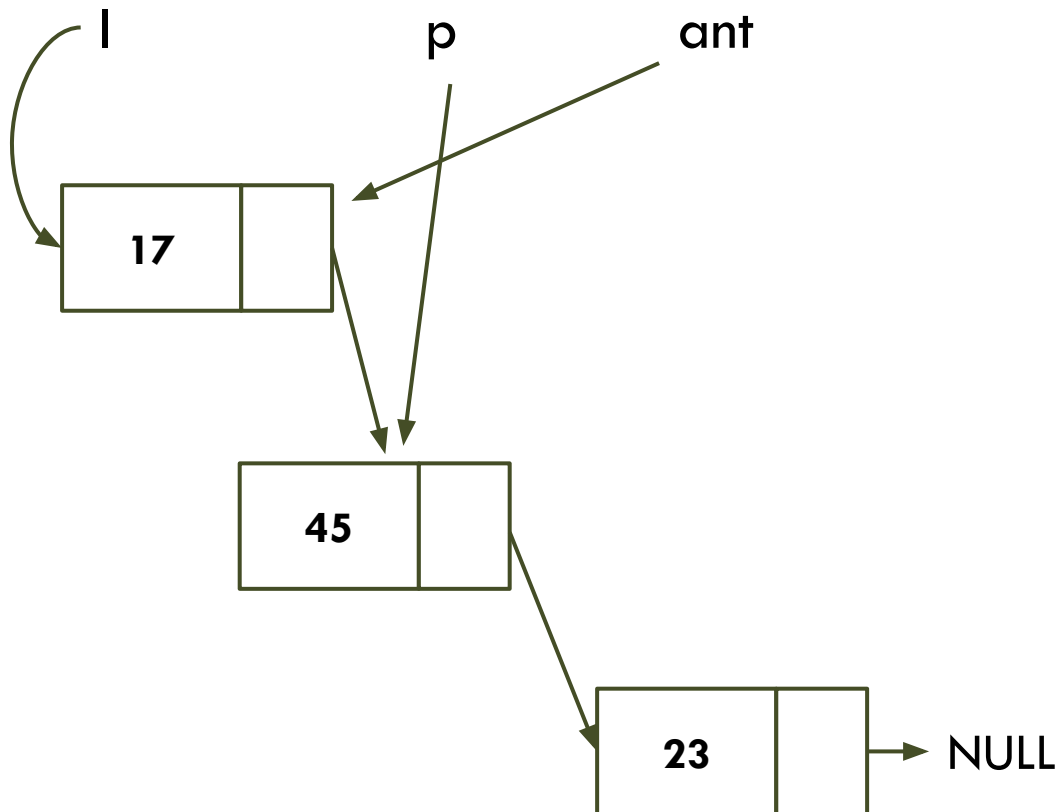
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do fim

99

lista = #NO17



main()

lista = retira(lista, 23);

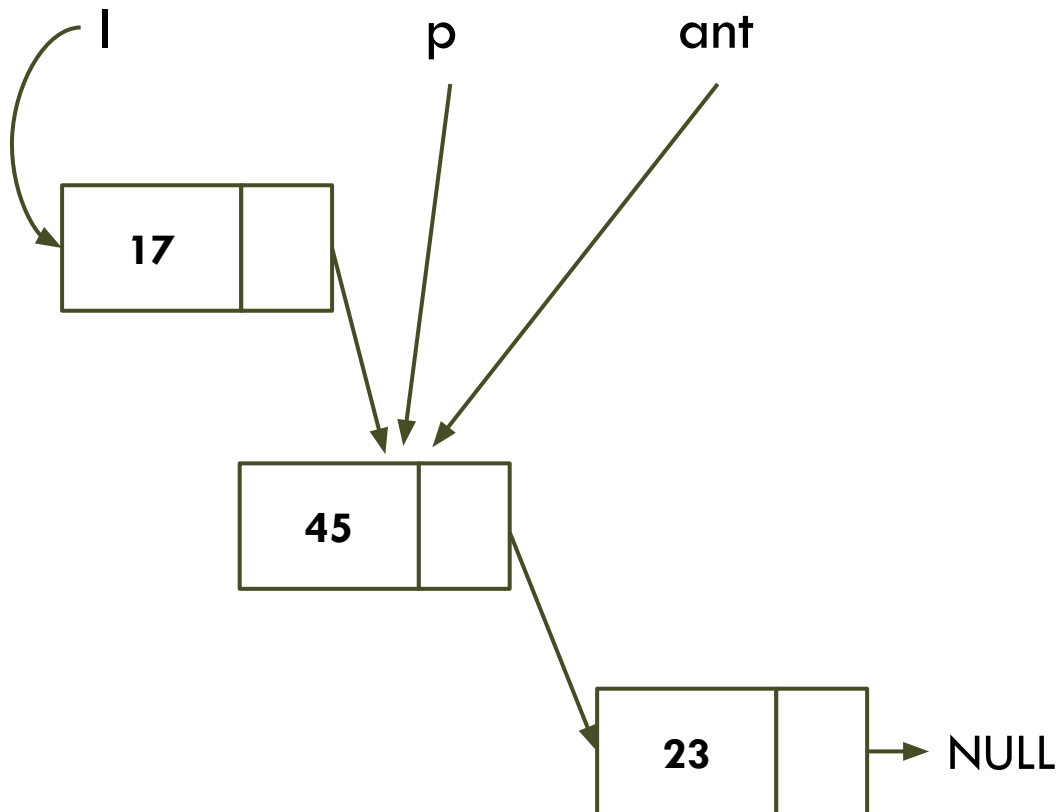
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do fim

100

lista = #NO17



main()

lista = retira(lista, 23);

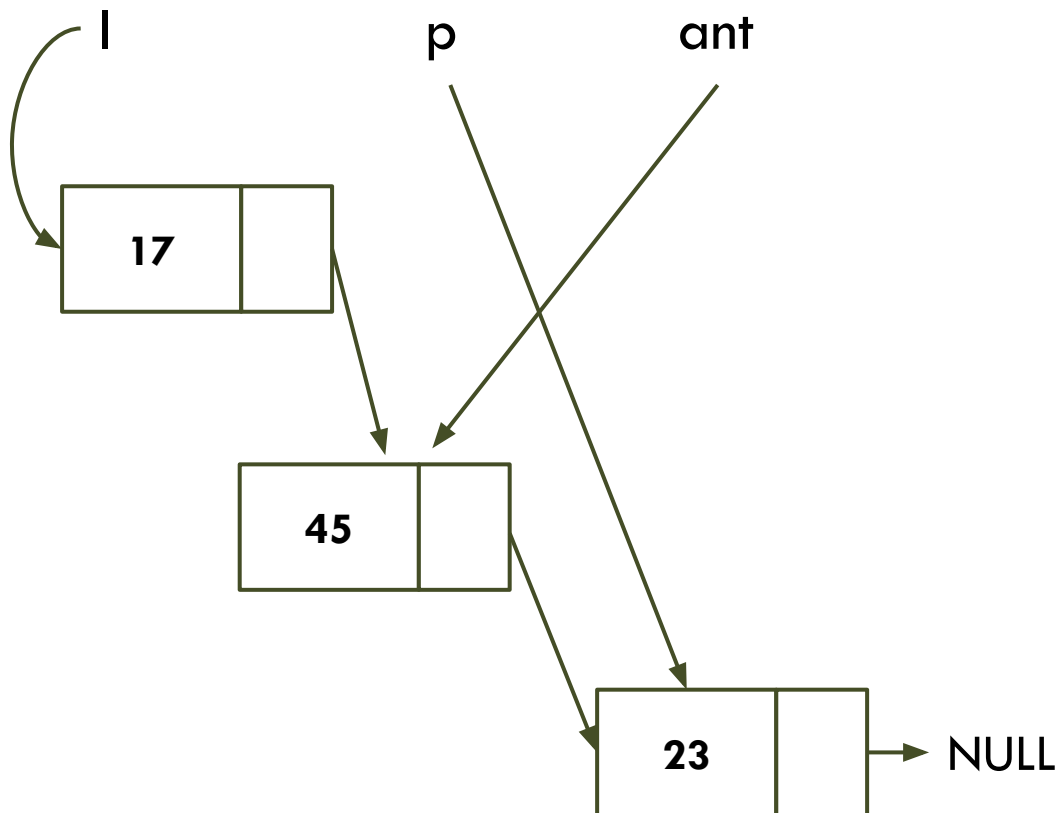
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do fim

101

lista = #NO17



main()

lista = retira(lista, 23);

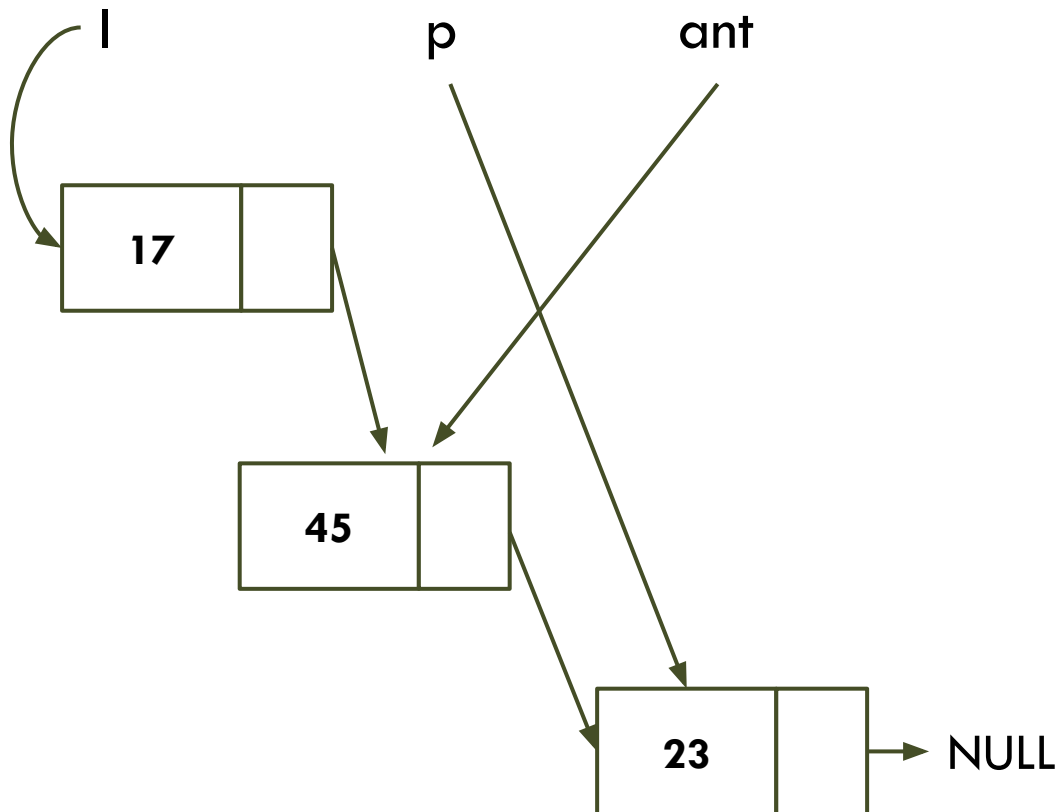
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do fim

102

lista = #NO17



main()

lista = retira(lista, 23);

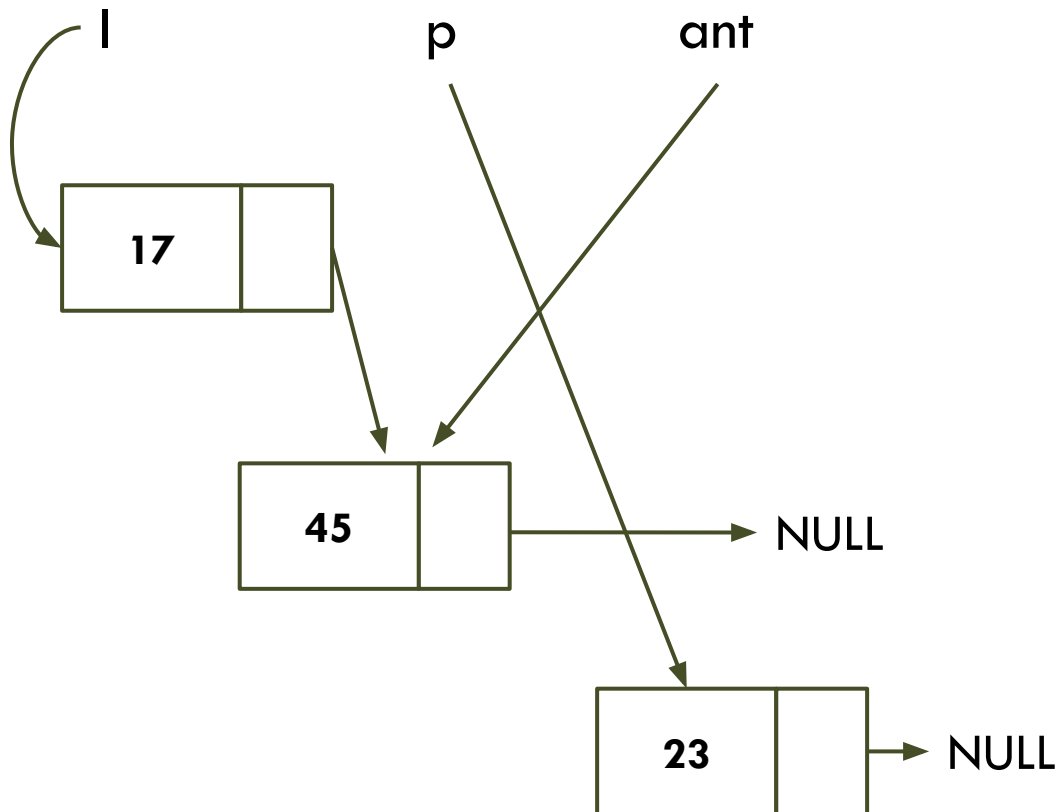
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do fim

103

lista = #NO17



main()

lista = retira(lista, 23);

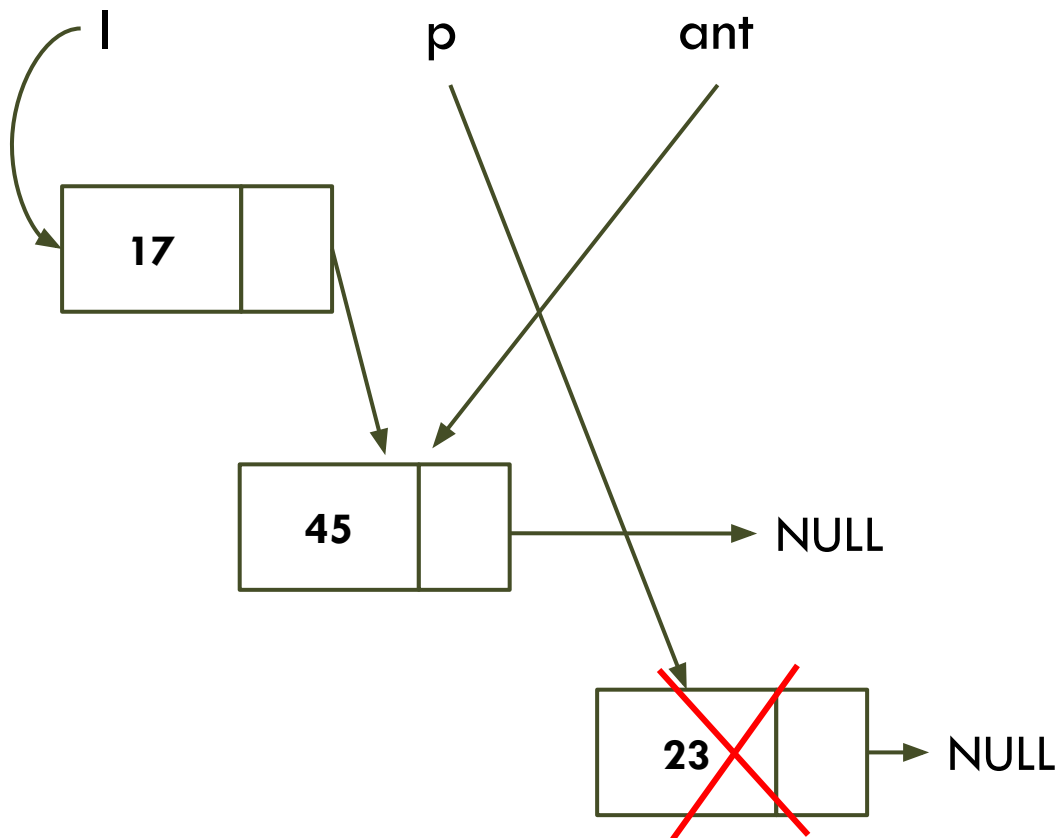
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do fim

104

lista = #NO17



main()

lista = retira(lista, 23);

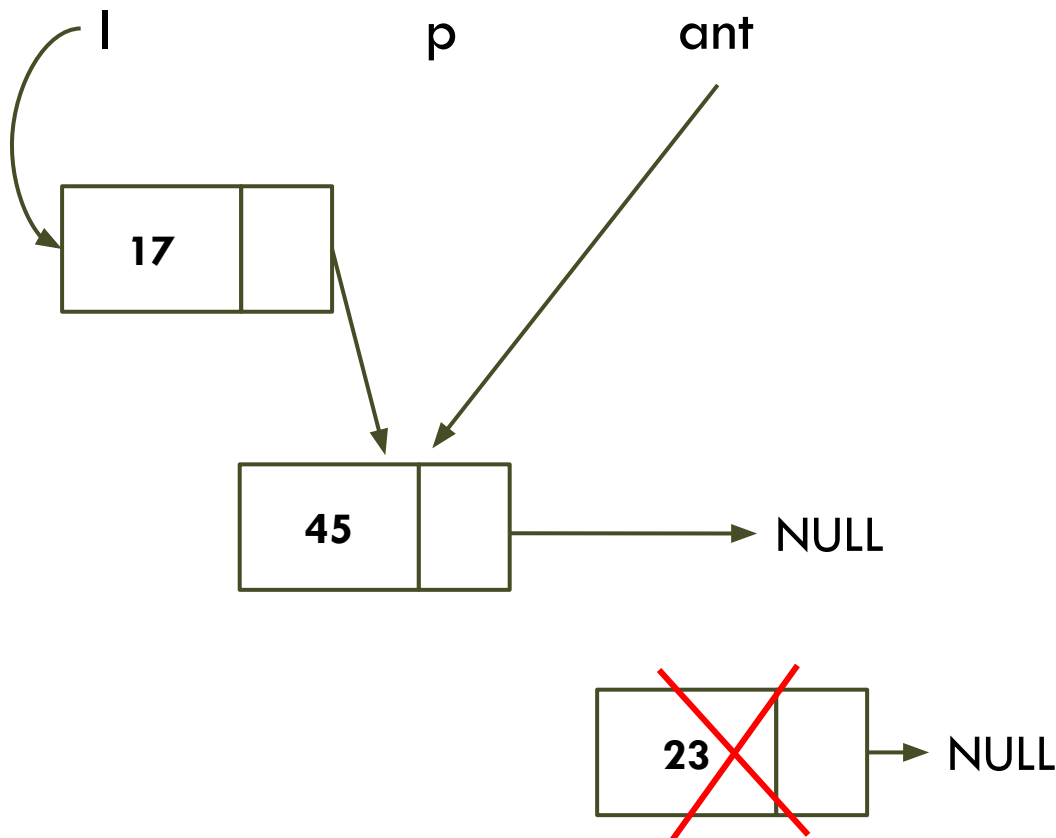
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```


Lista encadeada simples

Estudo de caso - remove elemento do fim

105

lista = #NO17



main()

lista = retira(lista, 23);

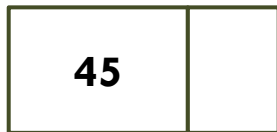
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada simples

Estudo de caso - remove elemento do fim

106

lista = #NO17



NULL

main()

lista = retira(lista, 23);

107

Lista encadeada circular

Lista encadeada circular

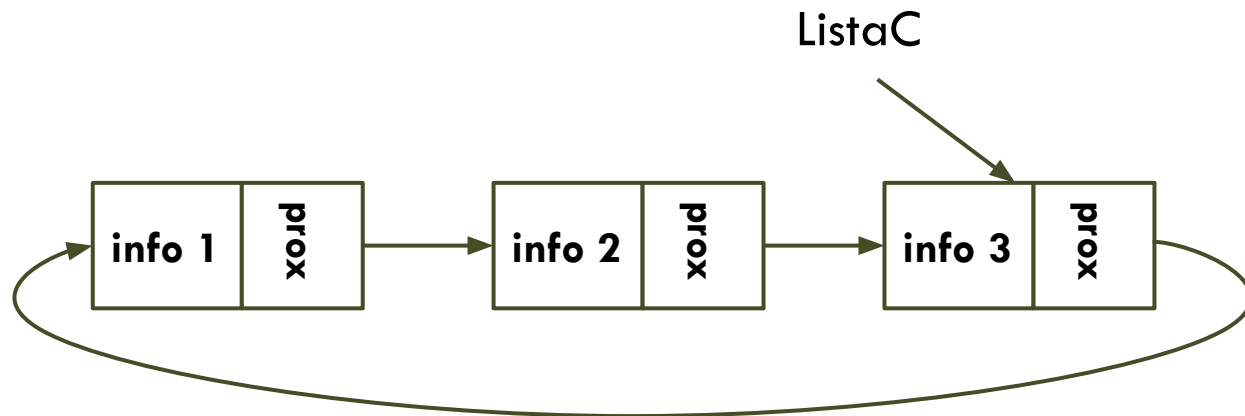
108

- ❏ Considere uma lista simples, e um ponteiro “p” que aponta para algum dos elementos da lista
 - **Problema:** a partir de “p”, não podemos acessar os elementos anteriores a “p”
 - **Solução:** último nó aponta para o primeiro
 - Definição de lista circular!

Lista encadeada circular

109

- ❑ O último elemento tem como próximo o primeiro elemento da lista, formando um ciclo
- ❑ A lista pode ser representada por um ponteiro para um elemento inicial qualquer da lista



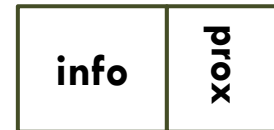
Lista encadeada circular

110

❏ Exemplo

- Lista encadeada circular armazenando valores inteiros

```
typedef struct listaC ListaC;  
  
struct listaC{  
    int info;  
    ListaC *prox;  
};
```



Lista encadeada circular

111

❏ **Função inicializa**

- Cria uma lista vazia, representada pelo ponteiro NULL

```
ListaC* inicializa (void){  
    return NULL;  
}
```

ListaC → NULL

Lista encadeada circular

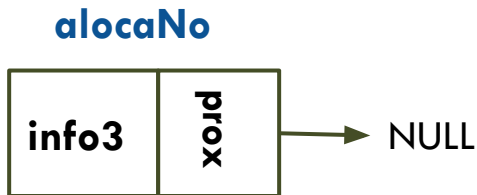
112

❏ Função insere

- Aloca memória para armazenar o elemento
- Encadeia o elemento na lista existente

```
ListaC* alocaNo(int valor){  
    ListaC* no = (ListaC*) malloc(sizeof(ListaC));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

```
ListaC* insereCircular (ListaC* l, int valor){  
    ListaC* novo = alocaNo(valor);  
    if (l == NULL){  
        novo->prox = novo;  
        l = novo;  
    }else{  
        novo->prox = l->prox;  
        l->prox = novo;  
    }  
    return l;  
}
```



Lista encadeada circular

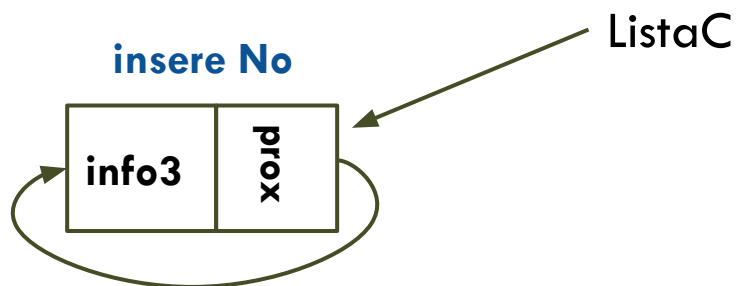
113

❏ Função insere

- Aloca memória para armazenar o elemento
- Encadeia o elemento na lista existente

```
ListaC* alocaNo(int valor){  
    ListaC* no = (ListaC*) malloc(sizeof(ListaC));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

```
ListaC* insereCircular (ListaC* l, int valor){  
    ListaC* novo = alocaNo(valor);  
    if (l == NULL){  
        novo->prox = novo;  
        l = novo;  
    }else{  
        novo->prox = l->prox;  
        l->prox = novo;  
    }  
    return l;  
}
```



Lista encadeada circular

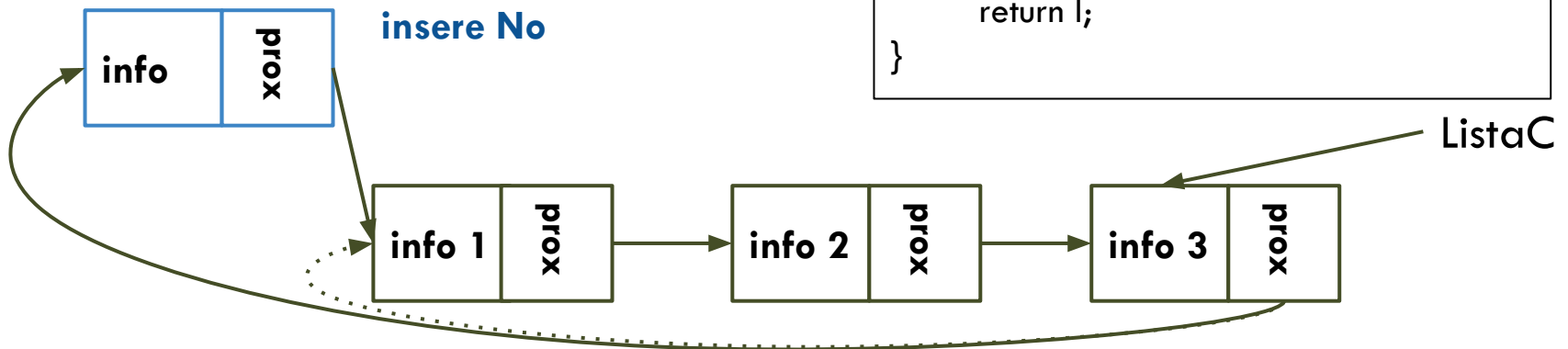
114

❏ Função insere

- Aloca memória para armazenar o elemento
- Encadeia o elemento na lista existente

```
ListaC* alocaNo(int valor){  
    ListaC* no = (ListaC*) malloc(sizeof(ListaC));  
    no->info = valor;  
    no->prox = NULL;  
    return no;  
}
```

```
ListaC* insereCircular (ListaC* l, int valor){  
    ListaC* novo = alocaNo(valor);  
    if (l == NULL){  
        novo->prox = novo;  
        l = novo;  
    }else{  
        novo->prox = l->prox;  
        l->prox = novo;  
    }  
    return l;  
}
```



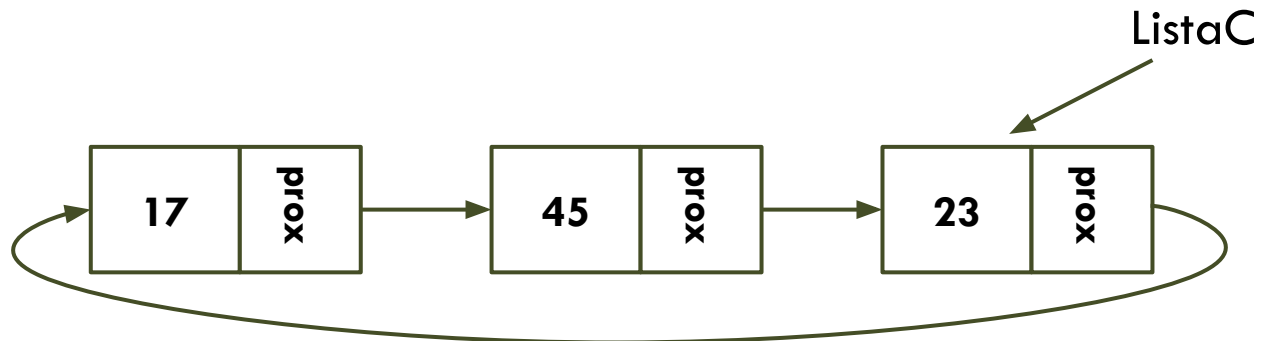
Lista encadeada circular

115

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprimeCircular (ListaC* l){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            printf(" %d ", p->info);  
        } while (p != l);  
    }  
}
```



Lista encadeada circular

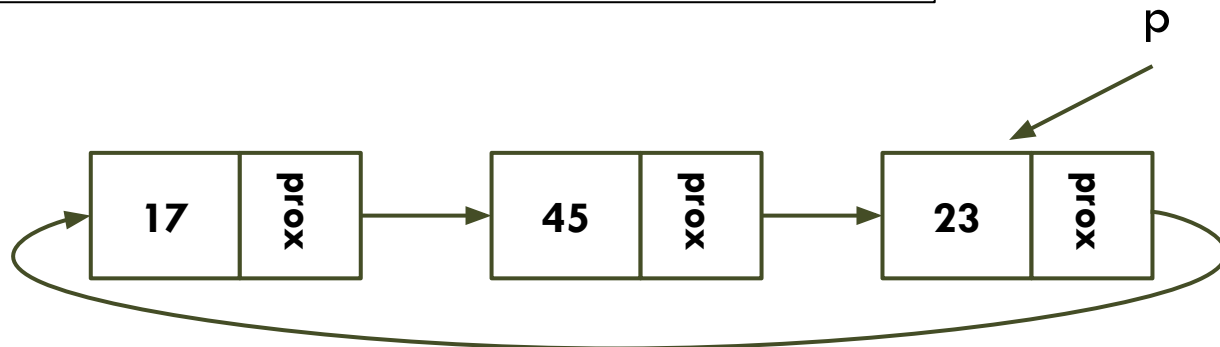
116

❑ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprimeCircular (ListaC* l){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            printf(" %d ", p->info);  
        } while (p != l);  
    }  
}
```

Saída:



Lista encadeada circular

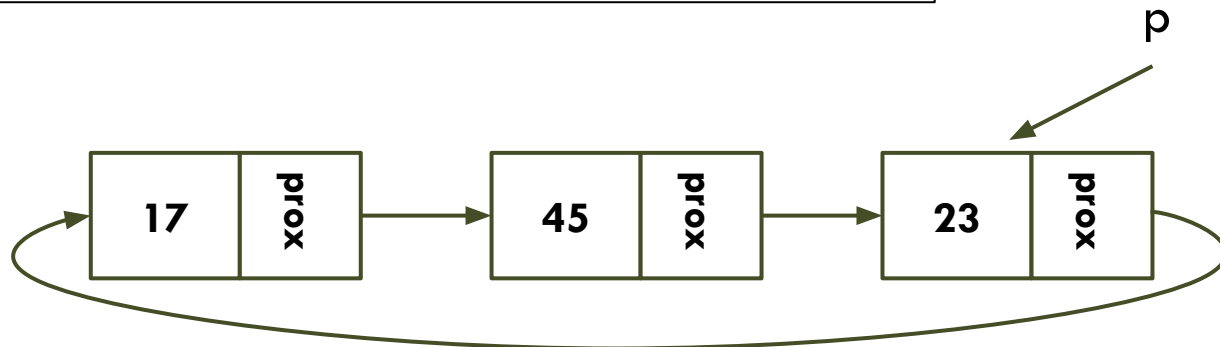
117

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprimeCircular (ListaC* l){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            printf(" %d ", p->info);  
        } while (p != l);  
    }  
}
```

Saída:



Lista encadeada circular

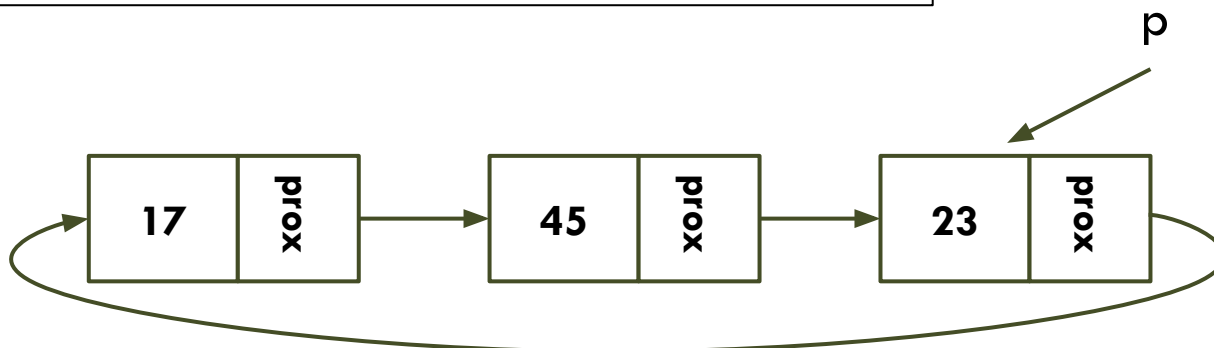
118

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprimeCircular (ListaC* l){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            printf(" %d ", p->info);  
        } while (p != l);  
    }  
}
```

Saída:



Lista encadeada circular

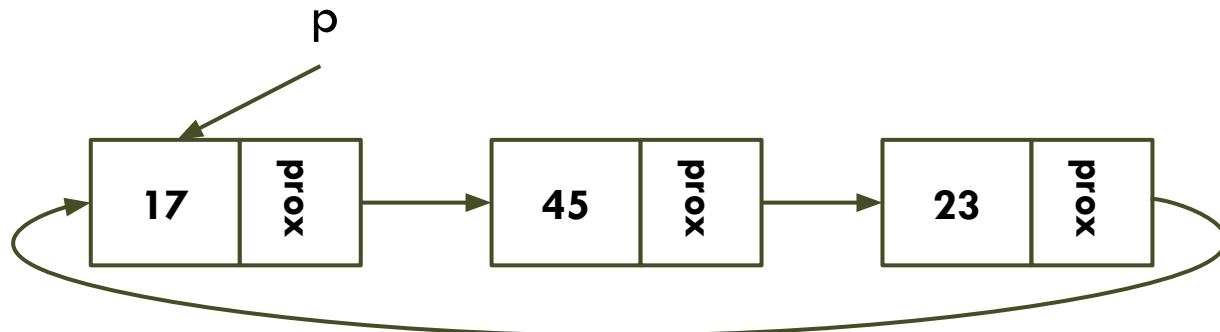
119

❑ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprimeCircular (ListaC* l){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            printf(" %d ", p->info);  
        } while (p != l);  
    }  
}
```

Saída:



Lista encadeada circular

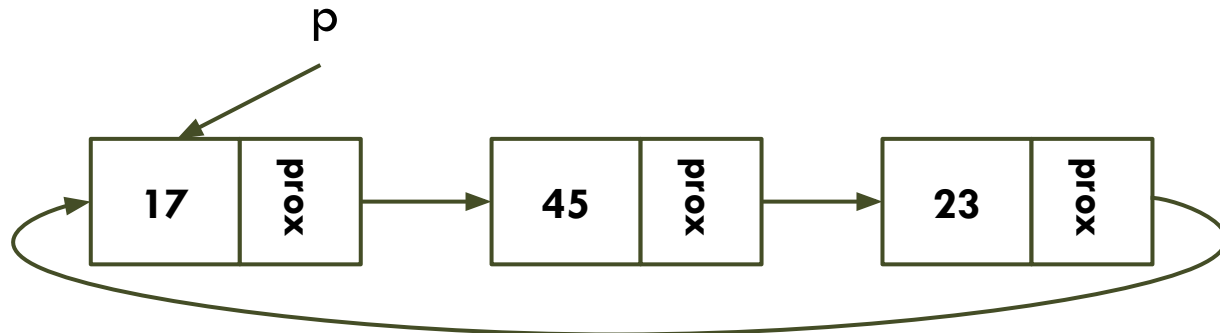
120

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprimeCircular (ListaC* l){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            printf(" %d ", p->info);  
        } while (p != l);  
    }  
}
```

Saída: 17



Lista encadeada circular

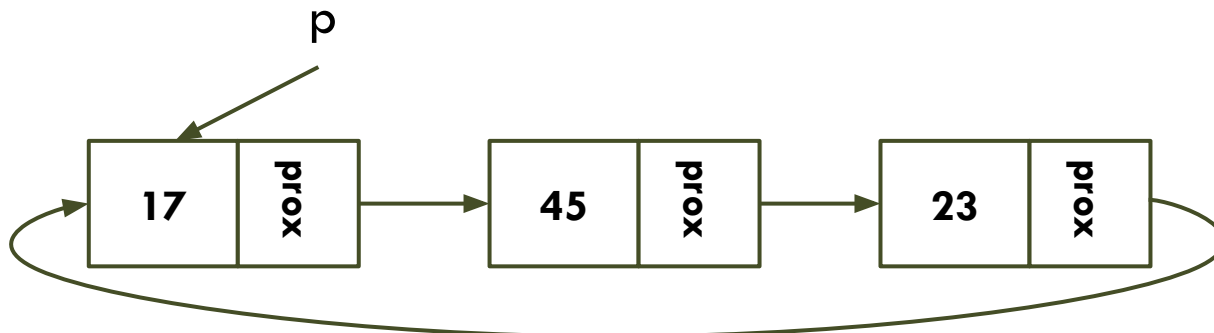
121

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprimeCircular (ListaC* l){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            printf(" %d ", p->info);  
        } while (p != l);  
    }  
}
```

Saída: 17



Lista encadeada circular

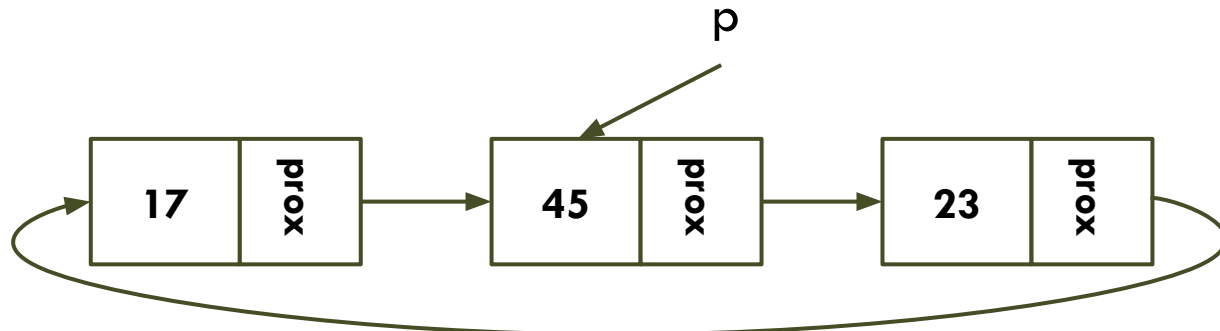
122

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprimeCircular (ListaC* l){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            printf(" %d ", p->info);  
        } while (p != l);  
    }  
}
```

Saída: 17



Lista encadeada circular

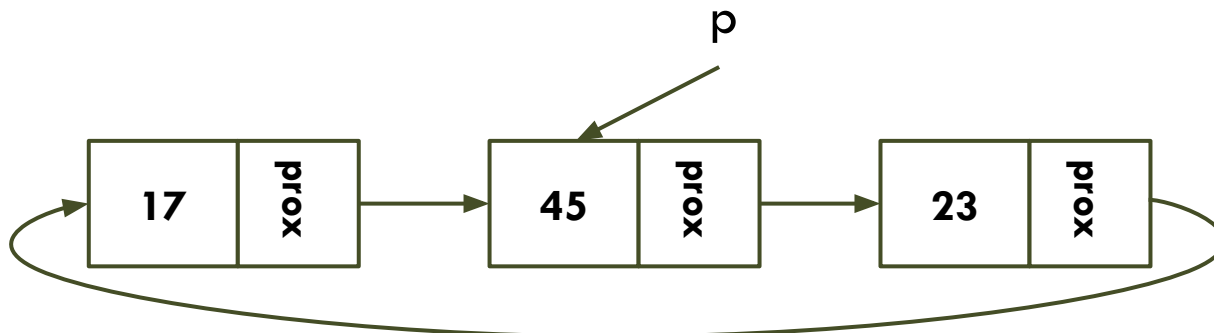
123

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprimeCircular (ListaC* l){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            printf(" %d ", p->info);  
        } while (p != l);  
    }  
}
```

Saída: 17 45



Lista encadeada circular

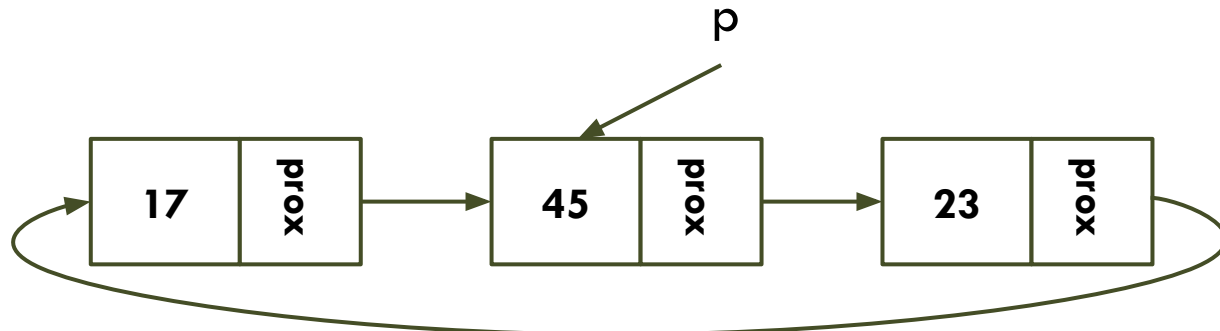
124

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprimeCircular (ListaC* l){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            printf(" %d ", p->info);  
        } while (p != l);  
    }  
}
```

Saída: 17 45



Lista encadeada circular

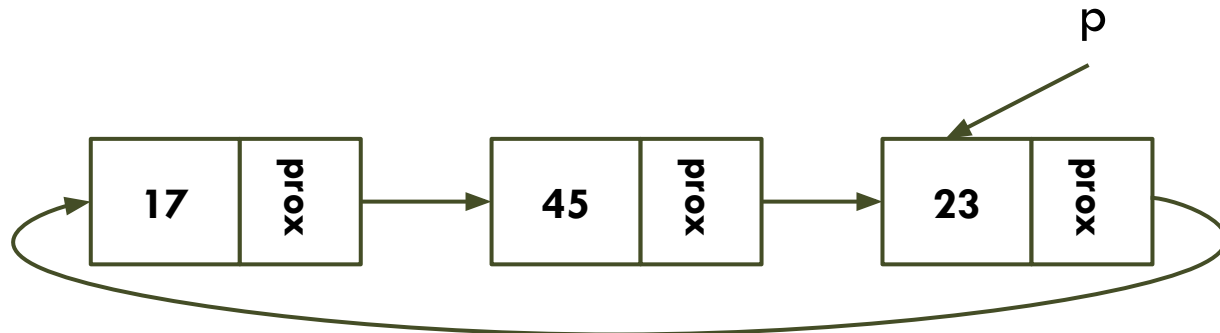
125

❑ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprimeCircular (ListaC* l){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            printf(" %d ", p->info);  
        } while (p != l);  
    }  
}
```

Saída: 17 45



Lista encadeada circular

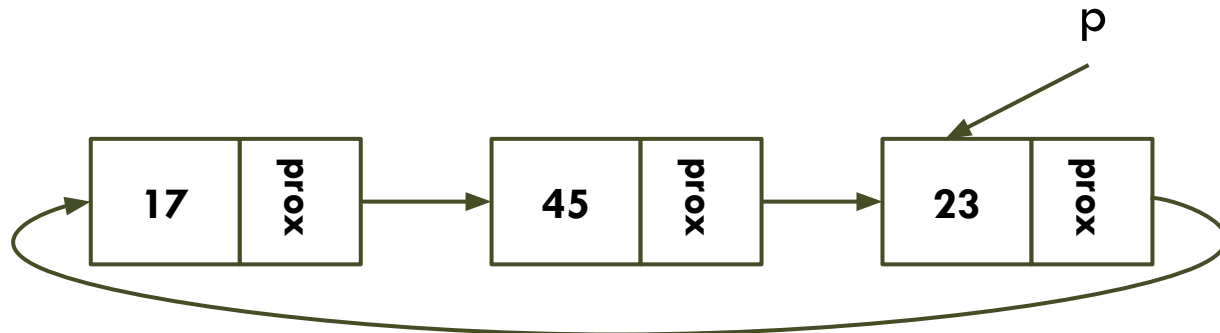
126

❑ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprimeCircular (ListaC* l){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            printf(" %d ", p->info);  
        } while (p != l);  
    }  
}
```

Saída: 17 45 23



Lista encadeada circular

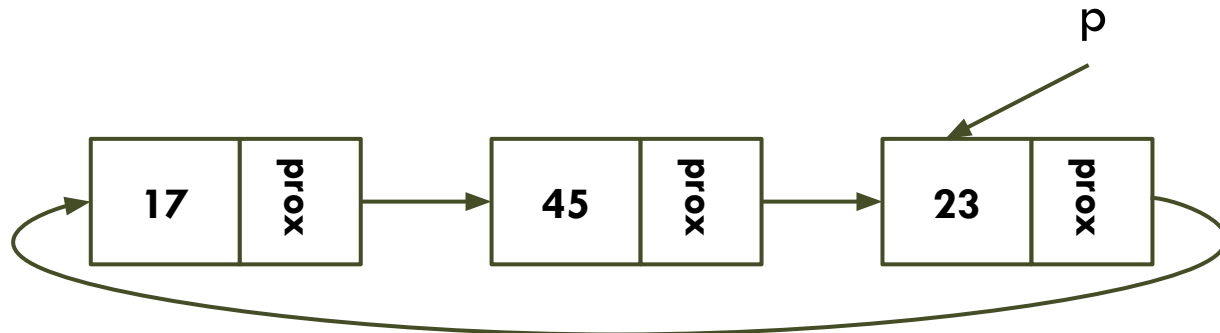
127

❏ Função imprime

- Imprime os valores dos elementos armazenados

```
void imprimeCircular (ListaC* l){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            printf(" %d ", p->info);  
        } while (p != l);  
    }  
}
```

Saída: 17 45 23



Lista encadeada circular

128

❏ **Função busca**

- Recebe como entrada a lista e o valor do elemento a retirar
- Se existe um único elemento na lista, remove o elemento e retorna NULL

```
ListaC* busca (ListaC* l, int v){
    ListaC* p = l;
    if (p){
        do {
            p = p->prox; /* avança para o próximo nó */
            if(p->info == v){
                return p;
            }
        } while (p != l);
    }
    return NULL;
}
```

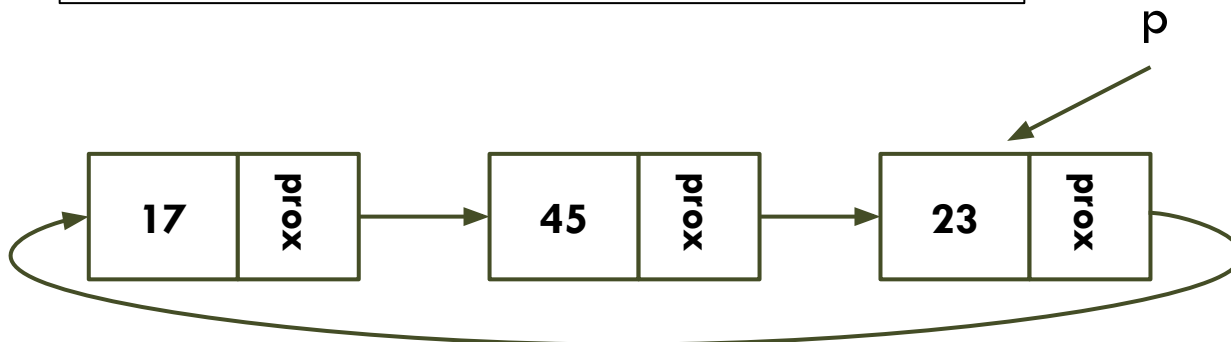

Lista encadeada circular

129

❏ Função busca

```
ListaC* busca (ListaC* l, int v){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            if(p->info == v){  
                return p;  
            }  
        } while (p != l);  
    }  
    return NULL;  
}
```

Buscando: 45



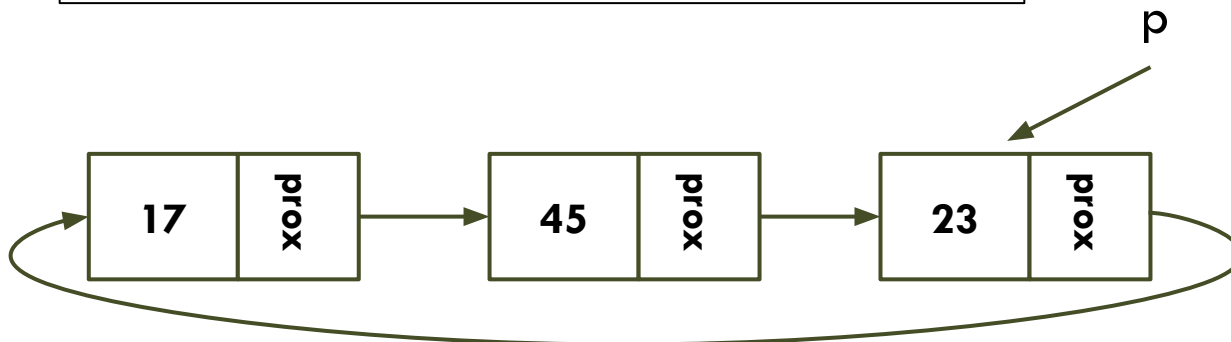
Lista encadeada circular

130

❏ Função busca

```
ListaC* busca (ListaC* l, int v){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            if(p->info == v){  
                return p;  
            }  
        } while (p != l);  
    }  
    return NULL;  
}
```

Buscando: 45



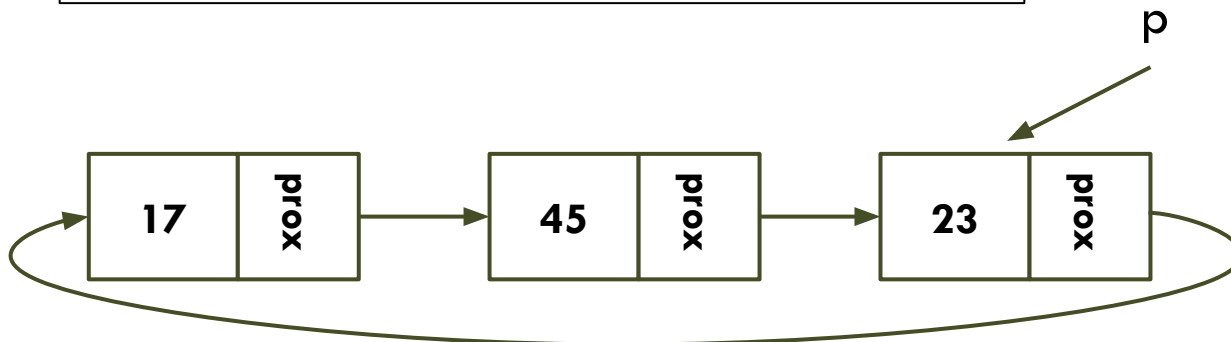
Lista encadeada circular

131

❑ Função busca

```
ListaC* busca (ListaC* l, int v){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            if(p->info == v){  
                return p;  
            }  
        } while (p != l);  
    }  
    return NULL;  
}
```

Buscando: 45



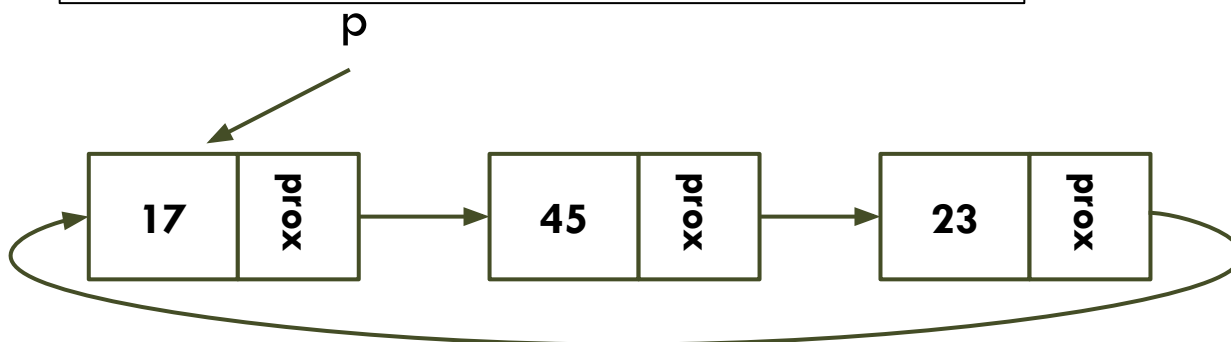
Lista encadeada circular

132

❏ Função busca

```
ListaC* busca (ListaC* l, int v){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            if(p->info == v){  
                return p;  
            }  
        } while (p != l);  
    }  
    return NULL;  
}
```

Buscando: 45



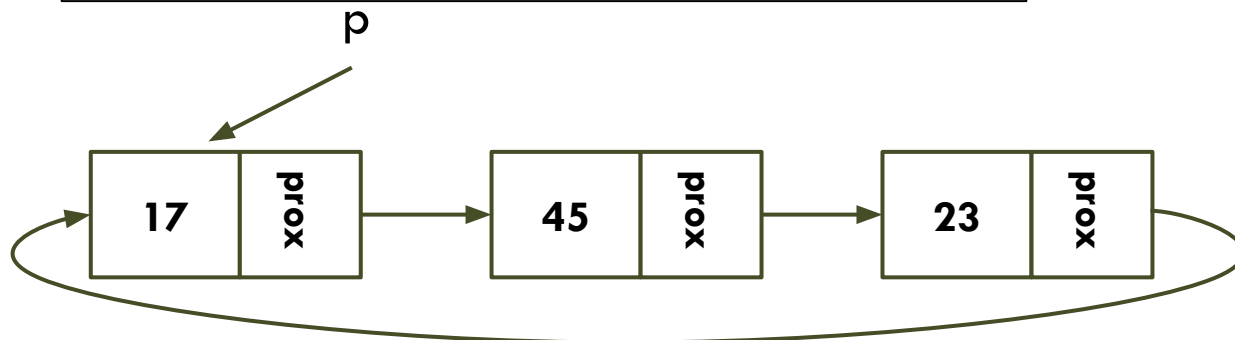
Lista encadeada circular

133

❏ Função busca

```
ListaC* busca (ListaC* l, int v){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            if(p->info == v){  
                return p;  
            }  
        } while (p != l);  
    }  
    return NULL;  
}
```

Buscando: 45



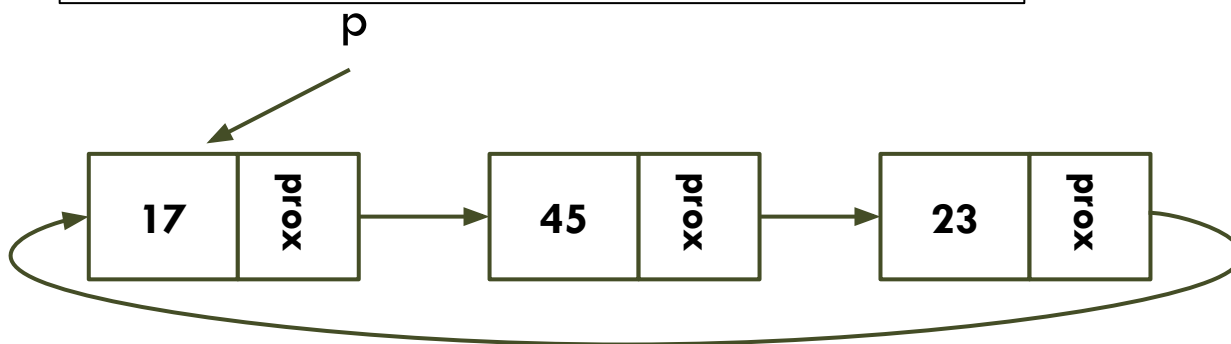
Lista encadeada circular

134

❏ Função busca

```
ListaC* busca (ListaC* l, int v){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            if(p->info == v){  
                return p;  
            }  
        } while (p != l);  
    }  
    return NULL;  
}
```

Buscando: 45



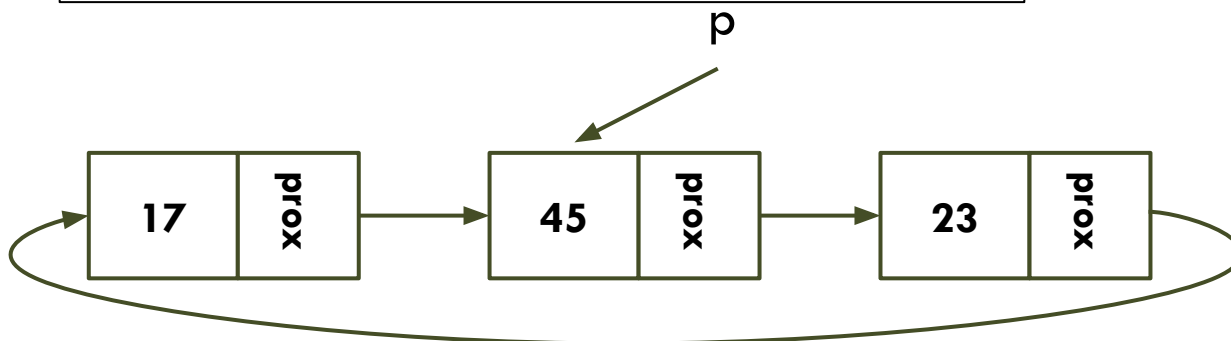
Lista encadeada circular

135

❏ Função busca

```
ListaC* busca (ListaC* l, int v){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            if(p->info == v){  
                return p;  
            }  
        } while (p != l);  
    }  
    return NULL;  
}
```

Buscando: 45



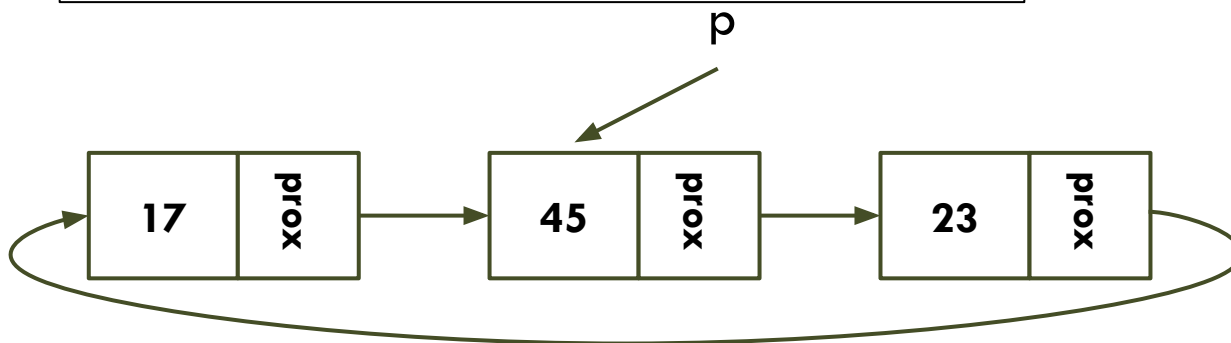
Lista encadeada circular

136

❏ Função busca

```
ListaC* busca (ListaC* l, int v){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            if(p->info == v){  
                return p;  
            }  
        } while (p != l);  
    }  
    return NULL;  
}
```

Buscando: 45



Lista encadeada circular

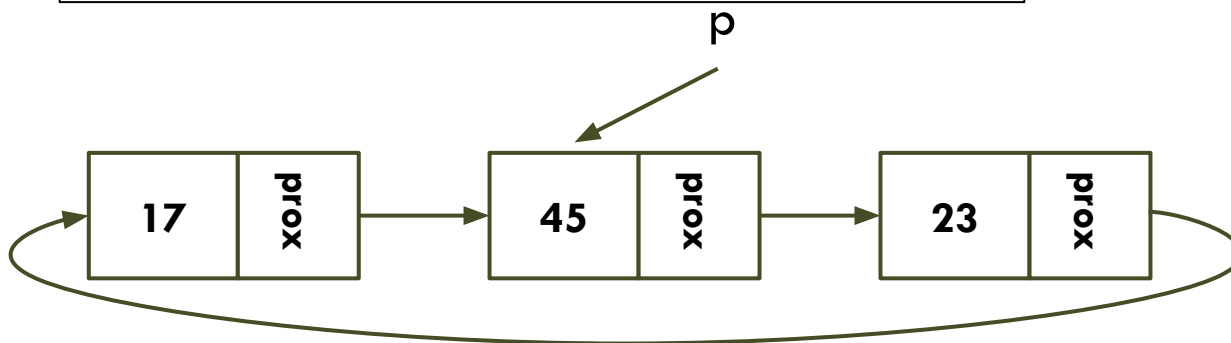
137

❏ Função busca

```
ListaC* busca (ListaC* l, int v){  
    ListaC* p = l;  
    if (p){  
        do {  
            p = p->prox; /* avança para o próximo nó */  
            if(p->info == v){  
                return p;  
            }  
        } while (p != l);  
    }  
    return NULL;  
}
```

Buscando: 45

Retorna p

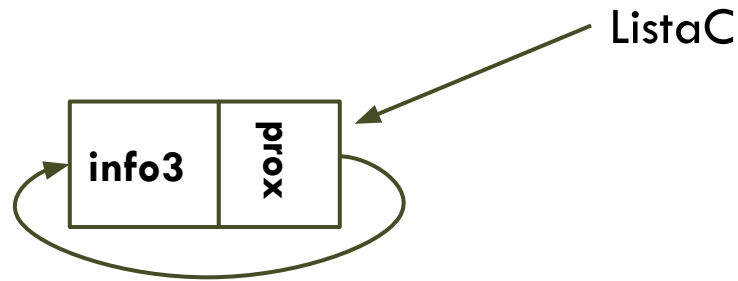


Lista encadeada circular

138

❏ **Função retira**

- Recebe como entrada a lista e o valor do elemento a retirar
- Se existe um único elemento na lista, remove o elemento e retorna NULL

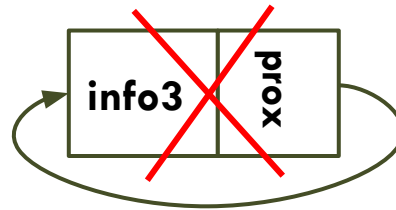


Lista encadeada circular

139

❏ **Função retira**

- Recebe como entrada a lista e o valor do elemento a retirar
- Se existe um único elemento na lista, remove o elemento e retorna NULL



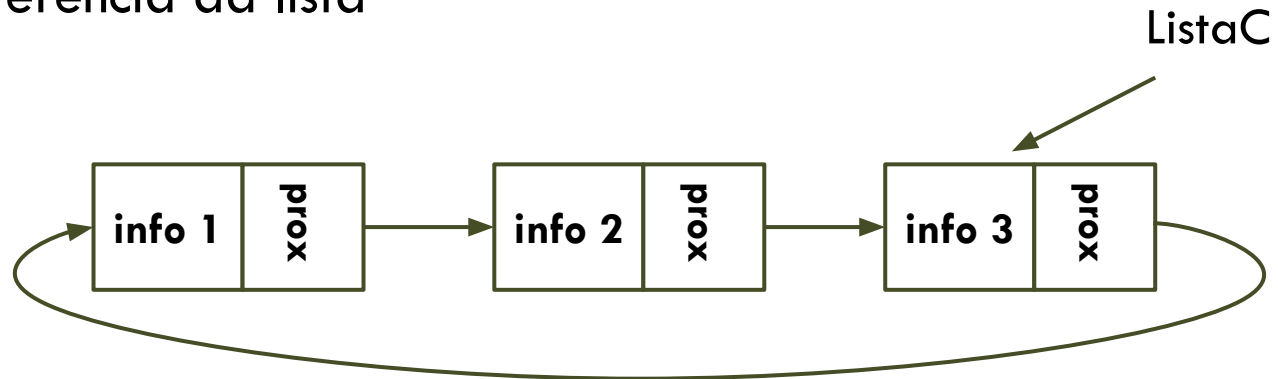
ListaC → NULL

Lista encadeada circular

140

❏ Função retira

- Recebe como entrada a lista e o valor do elemento a retirar
- Quando o elemento a remover é a referência da lista, atualiza o ponteiro anterior. E anterior será a nova referência da lista

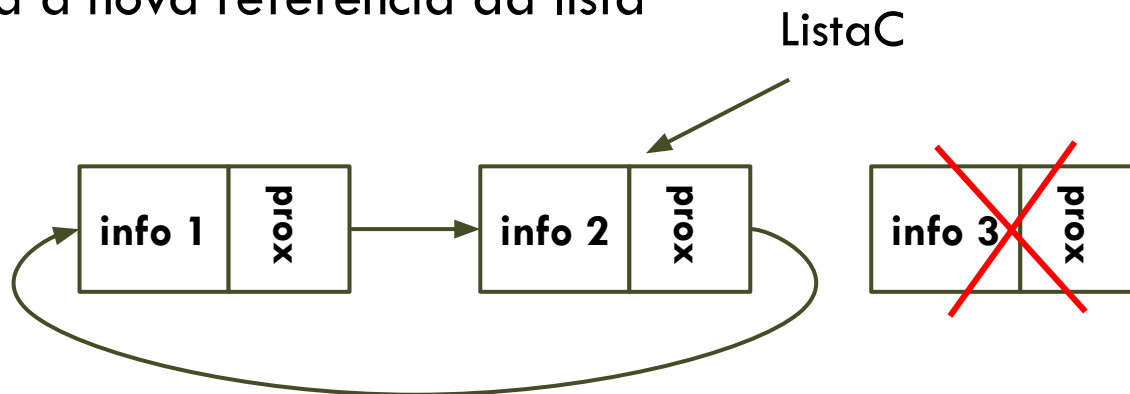


Lista encadeada circular

141

❏ Função retira

- Recebe como entrada a lista e o valor do elemento a retirar
- Quando o elemento a remover é a referência da lista, atualiza o ponteiro anterior, remove o elemento. E anterior será a nova referência da lista

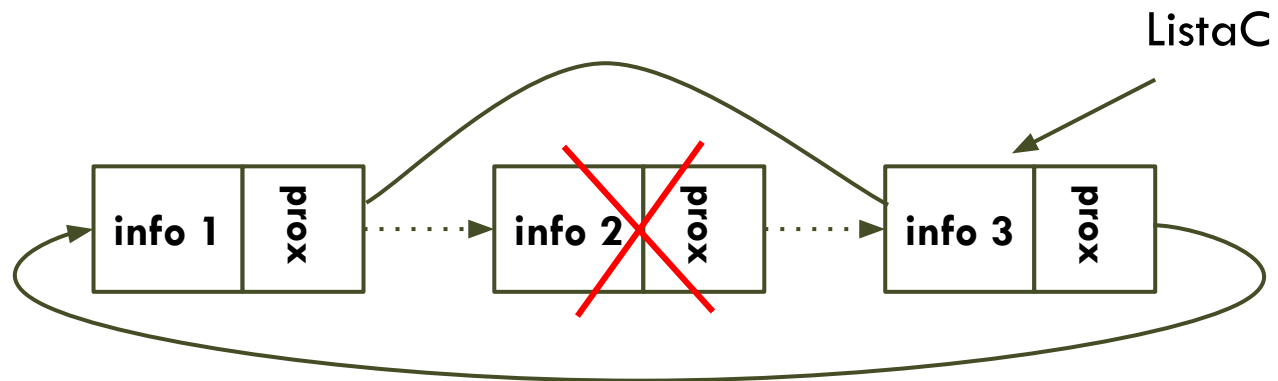


Lista encadeada circular

142

❏ Função retira

- Recebe como entrada a lista e o valor do elemento a retirar
- Quando é o elemento do meio, atualiza o ponteiro anterior e remove o elemento



Lista encadeada circular

143

Função retira

```
ListaC* retiraCircular(ListaC* l, int v){
    ListaC* ant = NULL;
    ListaC* p = l;
    int achou = 0;

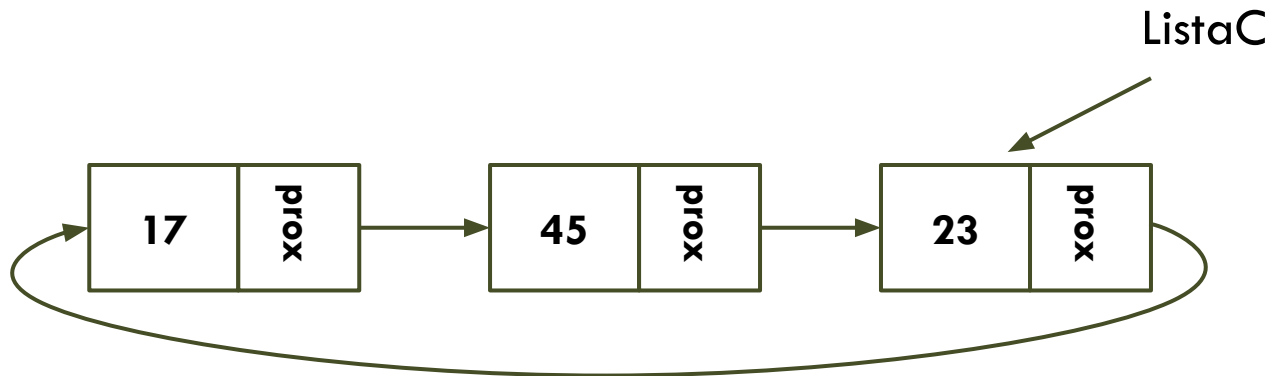
    if (p == NULL) // Se a lista for nula
        return NULL;
    if (p){ // descobrir quem é o anterior
        do {
            ant = p;
            p = p->prox; /* avança para o próximo nó */
            if (p->info == v){
                achou = 1;
                break;
            }
        } while (p != l);
    }
    if(!achou) // Se não achar o elemento
        return l;
    if (p->prox == p){ // Quando há apenas um único elemento da lista
        free(p);
        return NULL;
    }
    if (p == l){ // Caso quando a remoção é a referência da lista
        ant->prox = l->prox;
        l = ant;
    }else {
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Lista encadeada circular

144

❏ Função libera

- Libera todos os elementos (nós) alocados



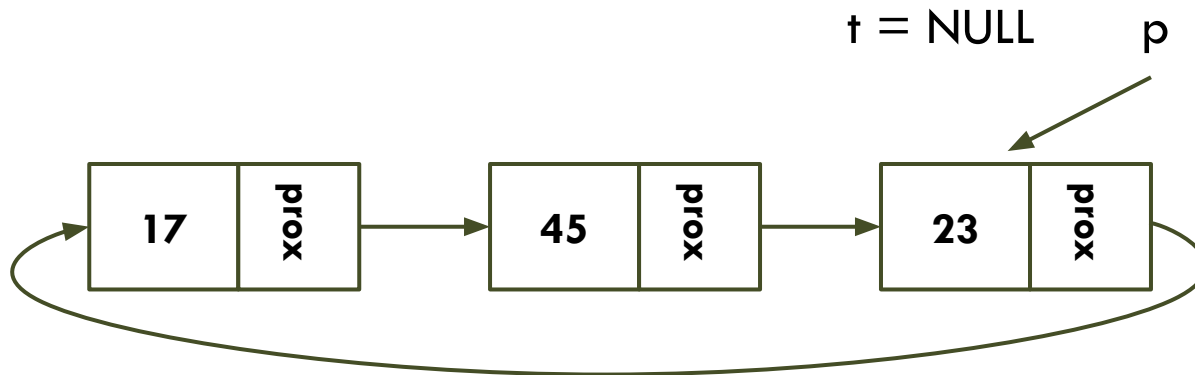
```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```


Lista encadeada circular

145

❏ Função libera

- Libera todos os elementos (nós) alocados



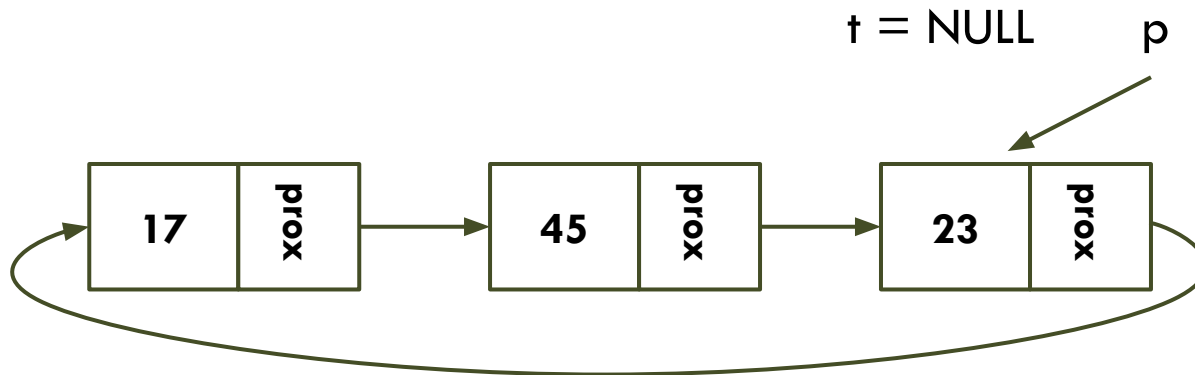
```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```

Lista encadeada circular

146

❏ Função libera

- Libera todos os elementos (nós) alocados



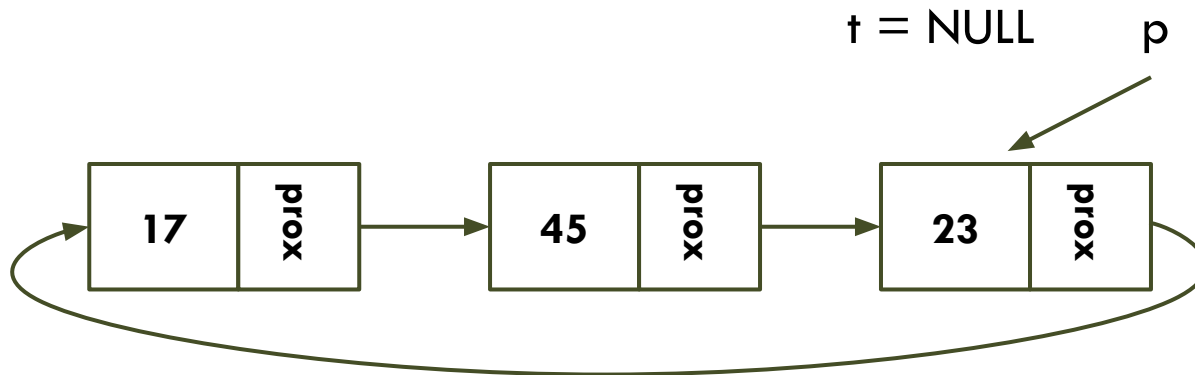
```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```

Lista encadeada circular

147

❏ Função libera

- Libera todos os elementos (nós) alocados



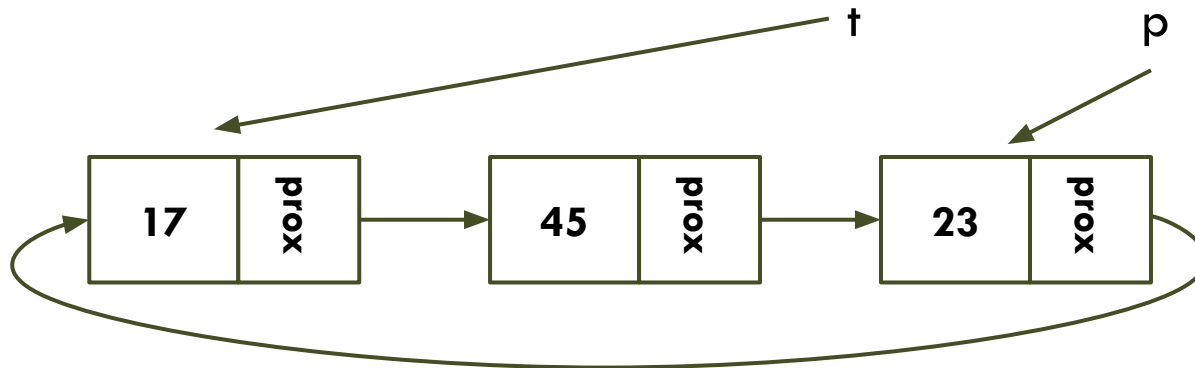
```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```

Lista encadeada circular

148

❏ Função libera

- Libera todos os elementos (nós) alocados



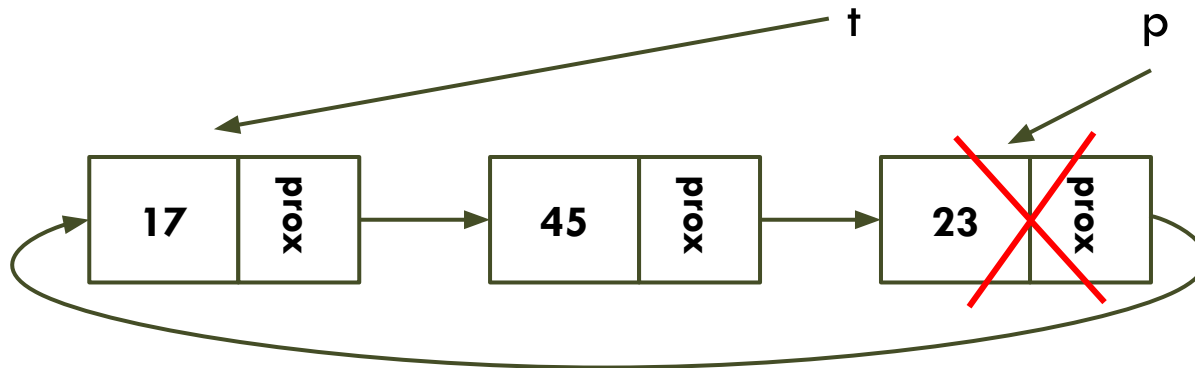
```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```

Lista encadeada circular

149

❏ Função libera

- Libera todos os elementos (nós) alocados



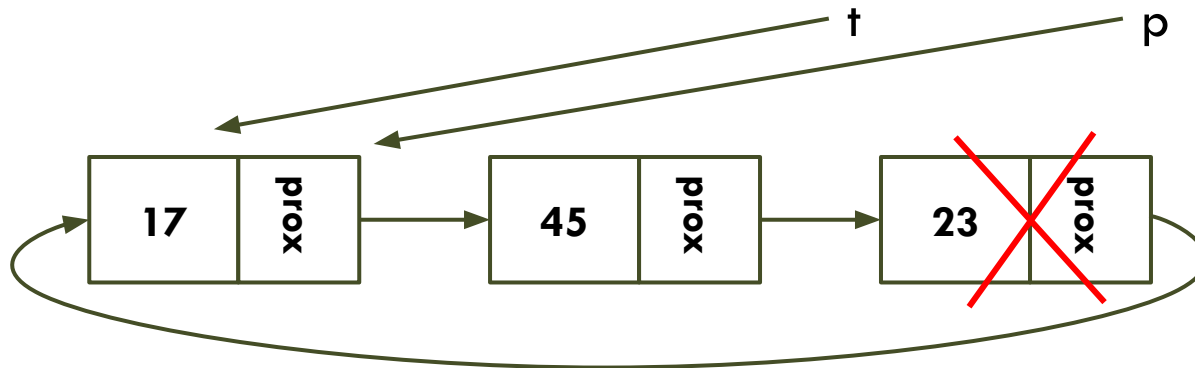
```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```

Lista encadeada circular

150

❏ Função libera

- Libera todos os elementos (nós) alocados



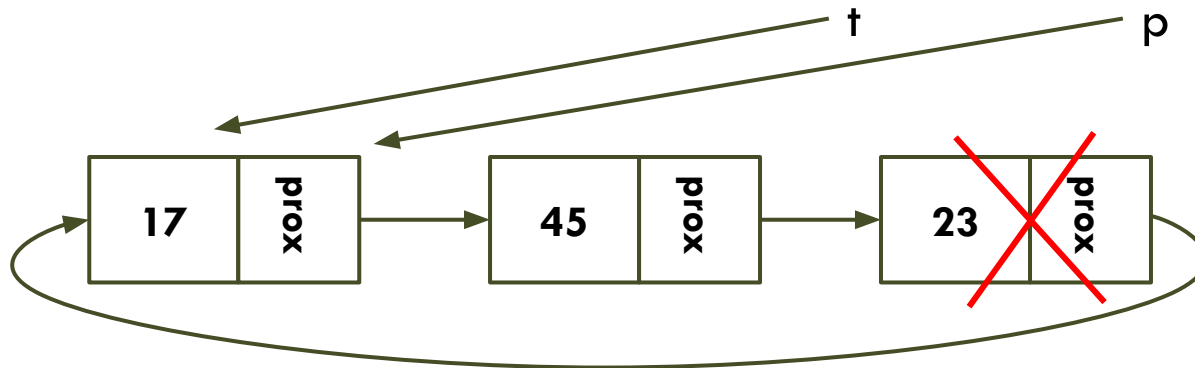
```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```

Lista encadeada circular

151

❏ Função libera

- Libera todos os elementos (nós) alocados



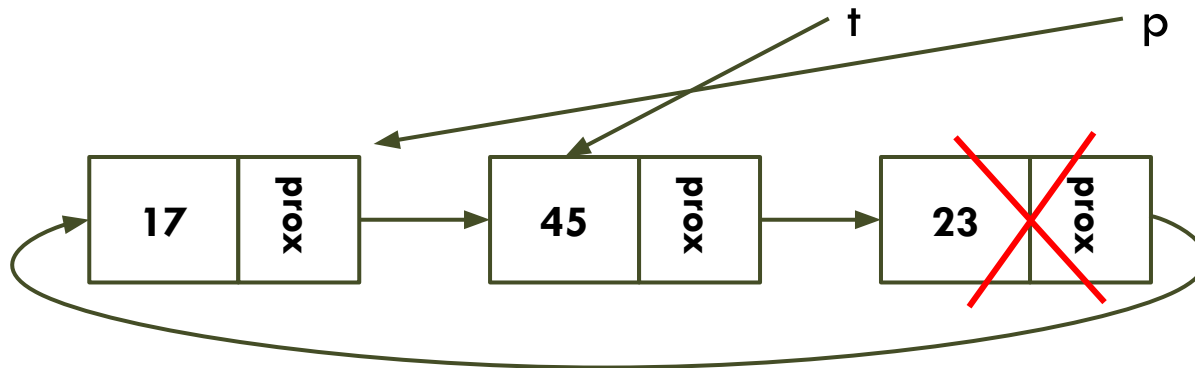
```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```

Lista encadeada circular

152

❏ Função libera

- Libera todos os elementos (nós) alocados



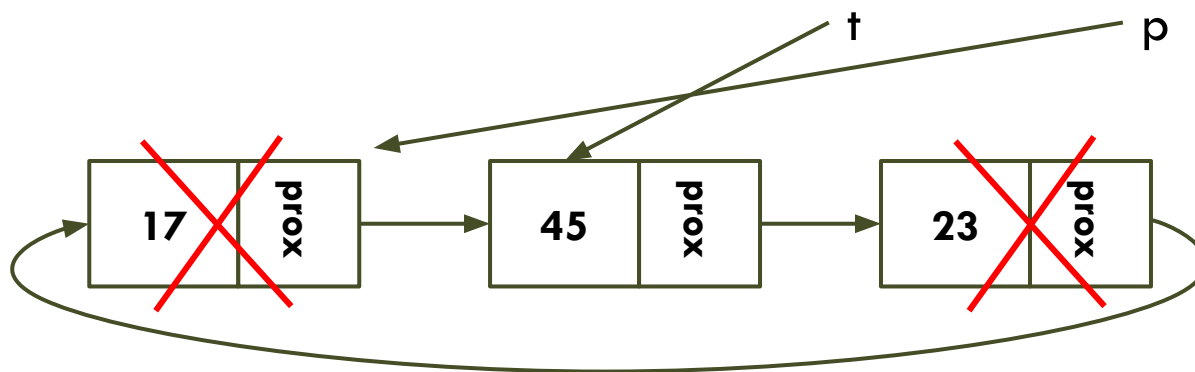
```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```


Lista encadeada circular

153

❏ Função libera

- Libera todos os elementos (nós) alocados



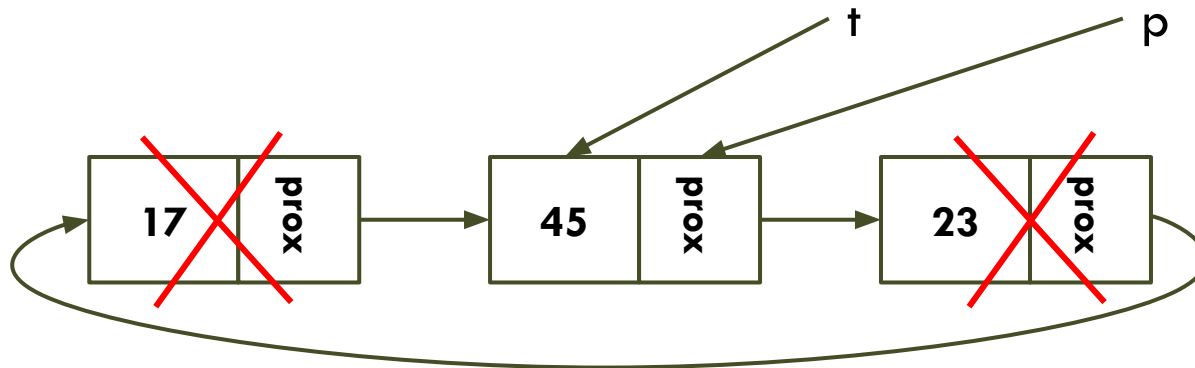
```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```

Lista encadeada circular

154

❏ Função libera

- Libera todos os elementos (nós) alocados



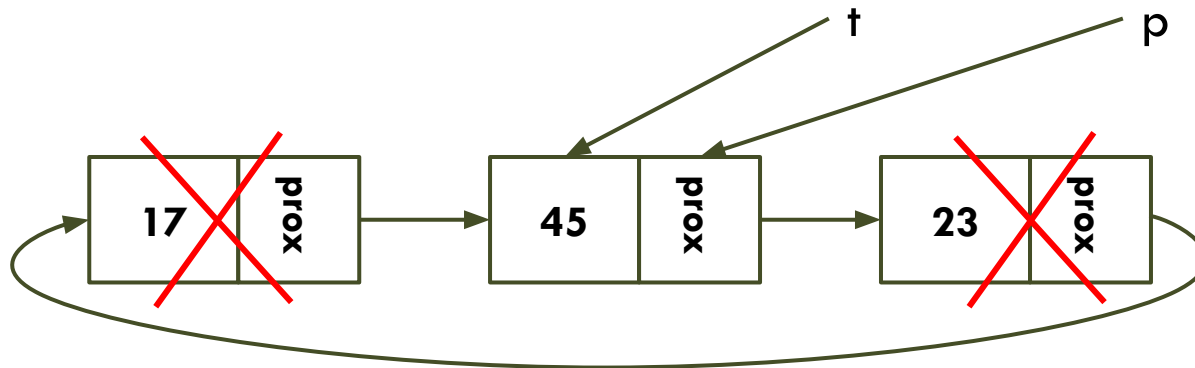
```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```

Lista encadeada circular

155

❏ Função libera

- Libera todos os elementos (nós) alocados



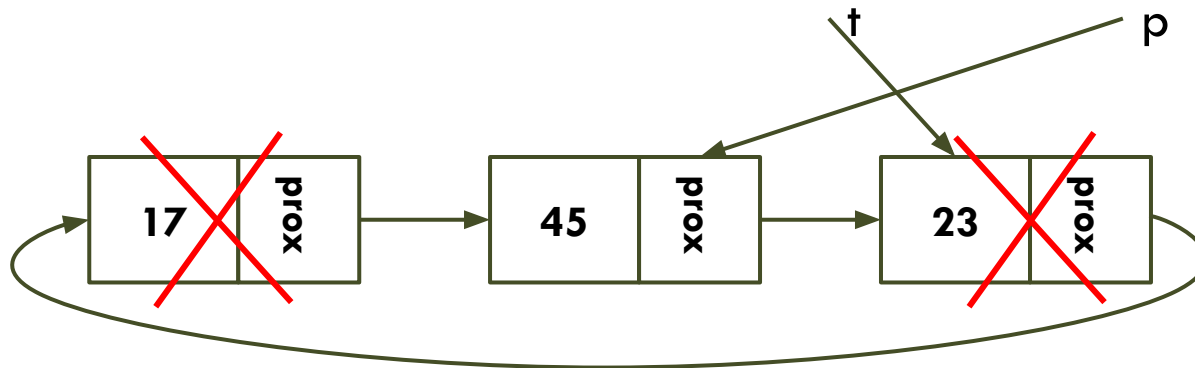
```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```

Lista encadeada circular

156

❏ Função libera

- Libera todos os elementos (nós) alocados



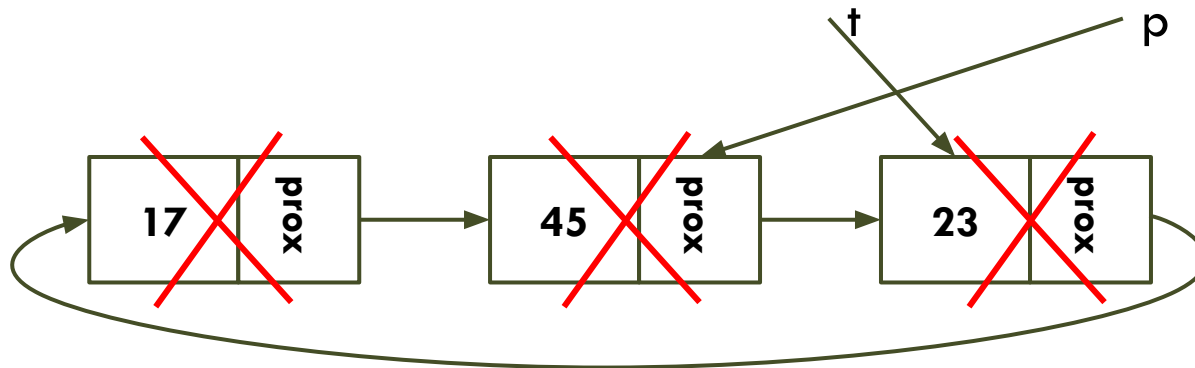
```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```

Lista encadeada circular

157

❏ Função libera

- Libera todos os elementos (nós) alocados



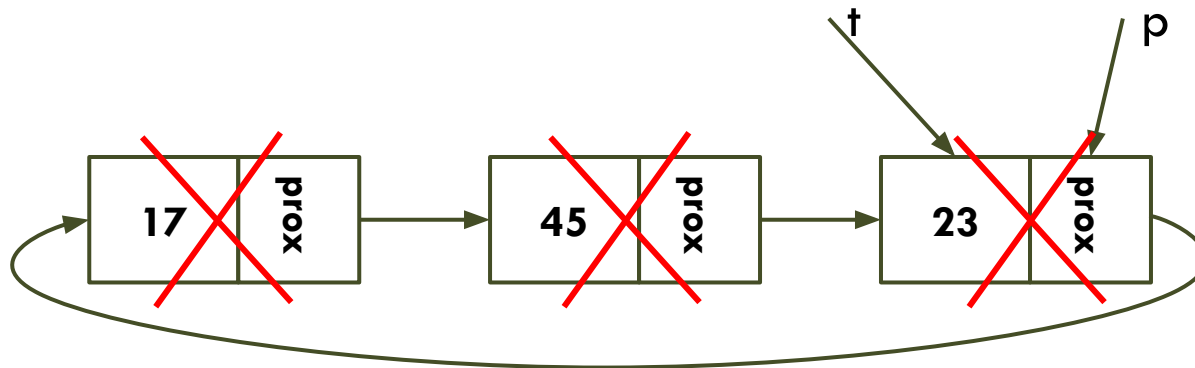
```
ListaC* libera (ListaC* l){
    ListaC* p = l;
    ListaC* t = NULL;
    if(p){
        do{
            t = p->prox;
            free(p);
            p = t;
        }while(p != l);
    }
    return NULL;
}
```

Lista encadeada circular

158

❏ Função libera

- Libera todos os elementos (nós) alocados



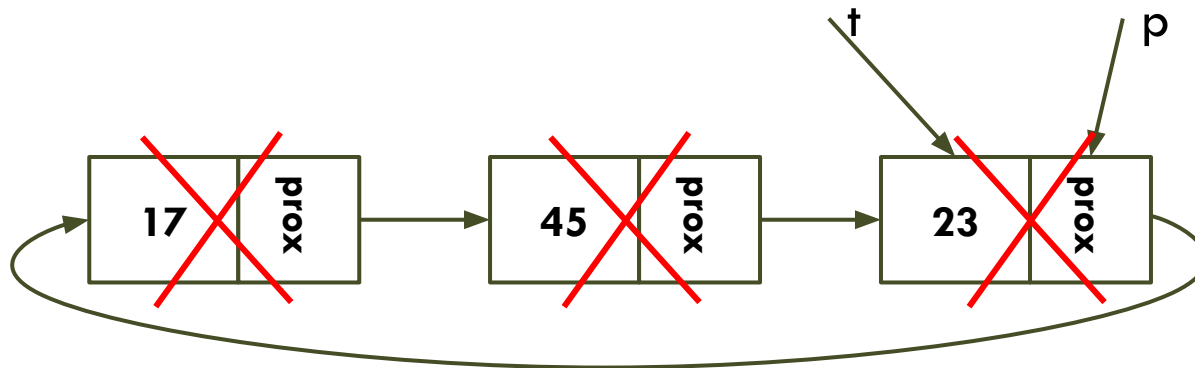
```
ListaC* libera (ListaC* l){
    ListaC* p = l;
    ListaC* t = NULL;
    if(p){
        do{
            t = p->prox;
            free(p);
            p = t;
        }while(p != l);
    }
    return NULL;
}
```

Lista encadeada circular

159

❏ Função libera

- Libera todos os elementos (nós) alocados



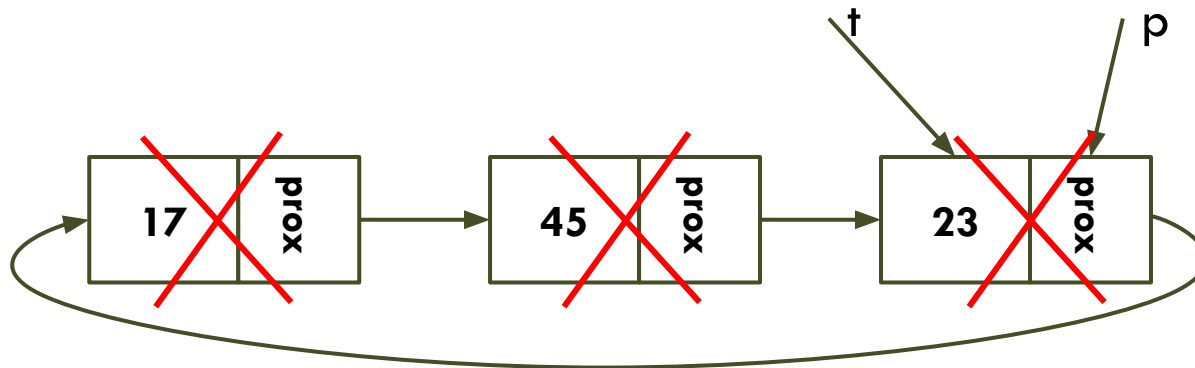
```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```

Lista encadeada circular

160

❏ Função libera

- Libera todos os elementos (nós) alocados



```
ListaC* libera (ListaC* l){  
    ListaC* p = l;  
    ListaC* t = NULL;  
    if(p){  
        do{  
            t = p->prox;  
            free(p);  
            p = t;  
        }while(p != l);  
    }  
    return NULL;  
}
```

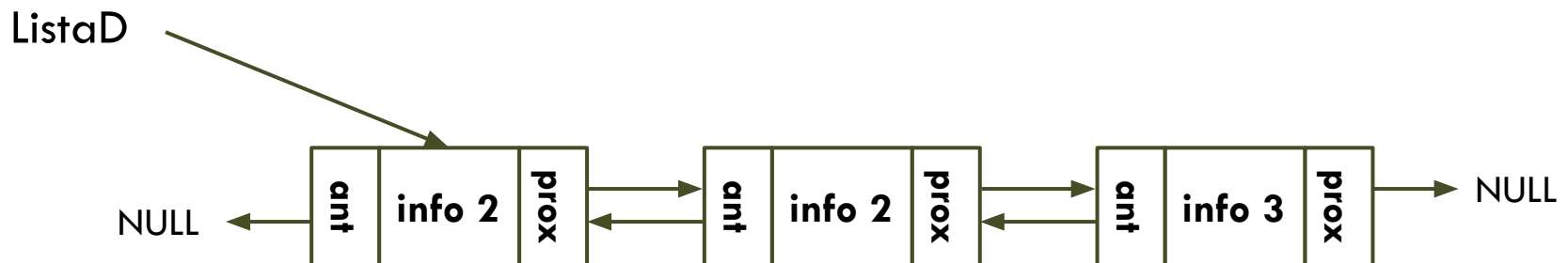

161

Lista duplamente encadeada e lista circular
duplamente encadeada

Lista duplamente encadeada

162

- ❑ Cada elemento tem um ponteiro para o próximo elemento e um ponteiro para o elemento anterior
 - Dado um elemento, é possível acessar o próximo e o anterior
 - Dado um ponteiro para o último elemento da lista, é possível percorrer a lista em ordem inversa



Lista duplamente encadeada

163

❏ Exemplo

- Lista duplamente encadeada armazenando valores inteiros

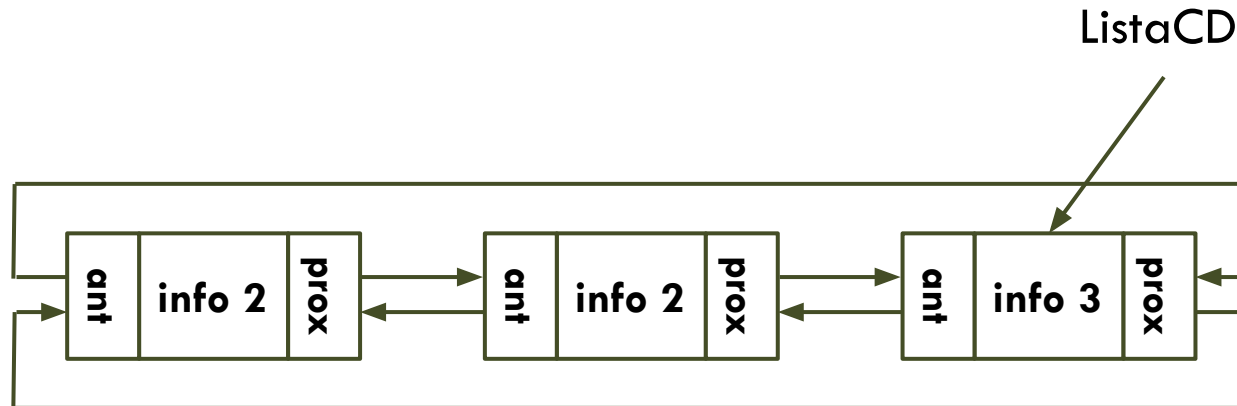
```
typedef struct listaD ListaD;  
  
struct listaD{  
    int info;  
    ListaD *prox;  
    ListaD *ant;  
};
```



Lista circular duplamente encadeada

164

- ❑ Cada elemento tem um ponteiro para o próximo elemento e um ponteiro para o elemento anterior
 - O último elemento tem como próximo o primeiro elemento da lista
 - E o primeiro elemento tem como anterior o último elemento da lista



Lista circular duplamente encadeada

165

❏ Exemplo

- Lista circular duplamente encadeada armazenando valores inteiros

```
typedef struct listaCD ListaCD;  
  
struct listaCD{  
    int info;  
    ListaCD *prox;  
    ListaCD *ant;  
};
```



Vantagens de listas

166

- ❑ Maior flexibilidade
 - Conjunto de dados pode crescer ou diminuir
- ❑ Evita o desperdício de memória
- ❑ Elementos podem ser inseridos e removidos em posições específicas

Atividade

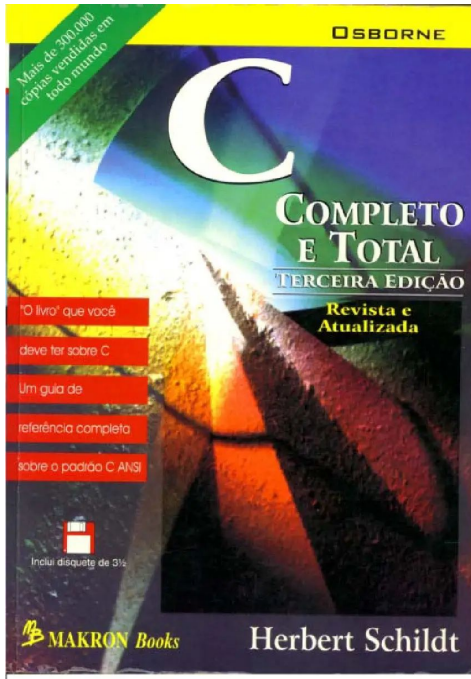
167

- ❑ Implementar as TADs para as listas duplamente encadeadas e listas circulares duplamente encadeadas
 - Inicializa
 - Insere
 - Imprime
 - Busca
 - Remove
 - Libera

Referências



168



SCHILD, Herbert. **C completo e total**. Makron, 3ª edição revista e atualizada, 1997.



SZWARCHFITER, J. **Estruturas de Dados e seus algoritmos**. 3 ed. Rio de Janeiro: LTC, 2010.