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

Novembro - 2021

Roteiro

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

Introdução

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

int vet[10];

Introdução

- **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

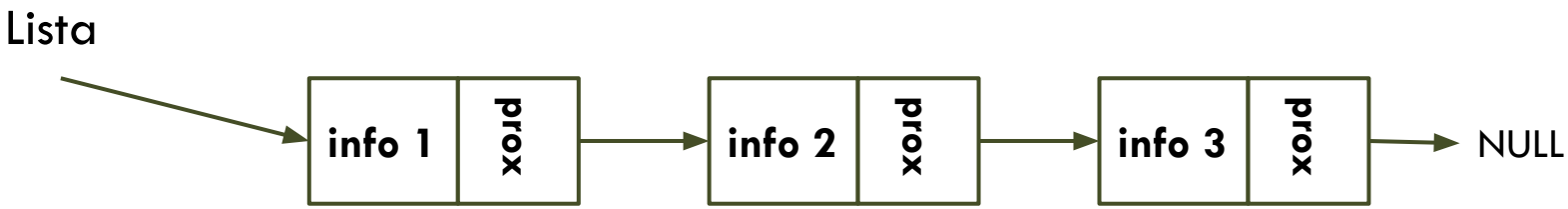
Introdução

- **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

- 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

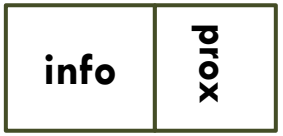


Lista encadeada simples

- **Exemplo**
  - Lista encadeada armazenando valores inteiros

```
typedef struct lista Lista;

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



Lista encadeada simples

- **Função inicializa**
  - Cria uma lista vazia, representada pelo ponteiro NULL

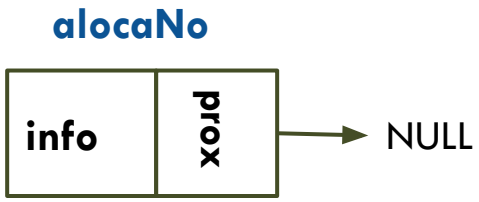
```
Lista* inicializa (void){
        return NULL;
}
```

Lista encadeada simples

- **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;
}
```

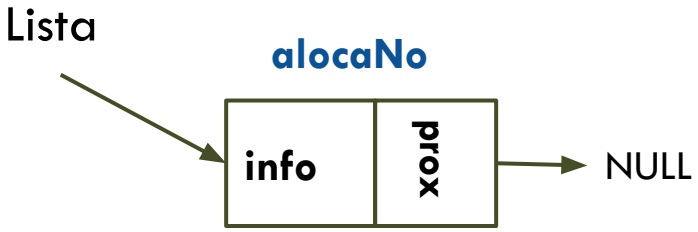


Lista encadeada simples

- **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;
}
```

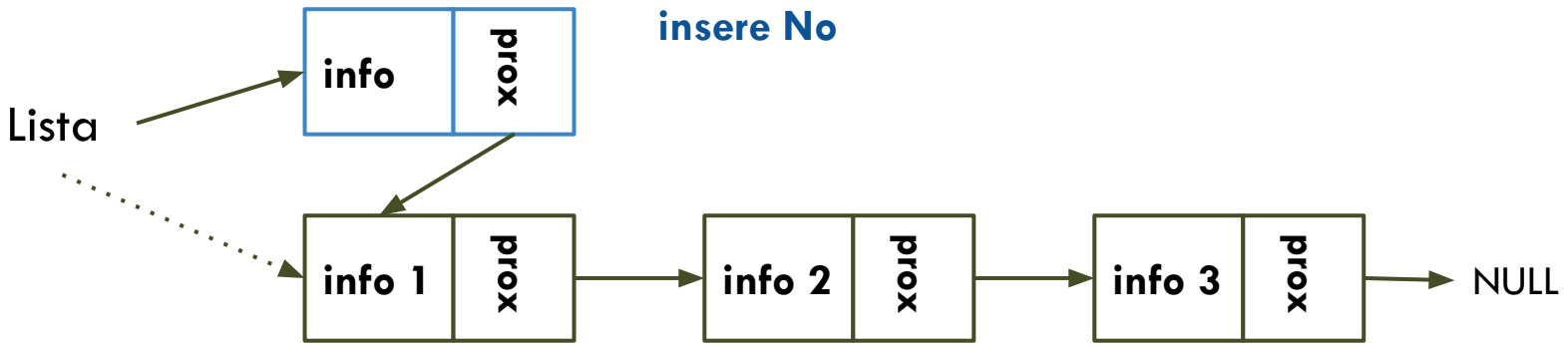


Lista encadeada simples

- **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;
}
```

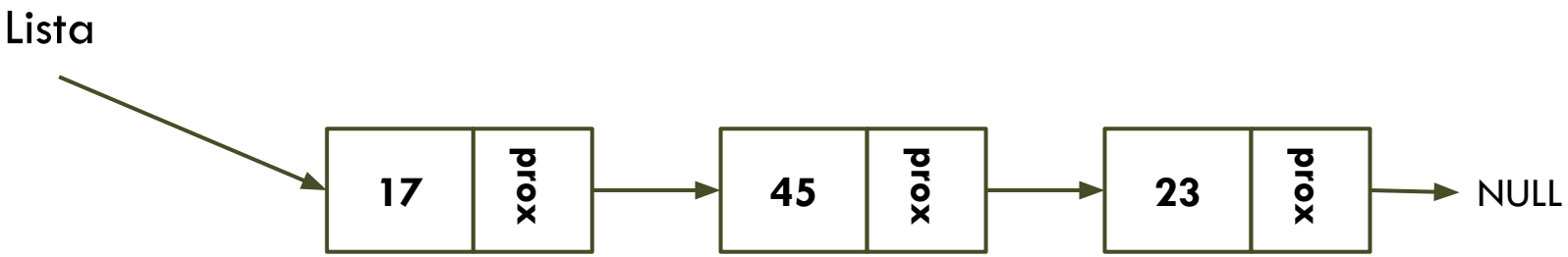
```
Lista* insere (Lista* l, int valor){
        Lista* novo = alocaNo(valor);
        novo->prox = l;
        return novo;
}
```



Lista encadeada simples

- **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);
    }
}
```

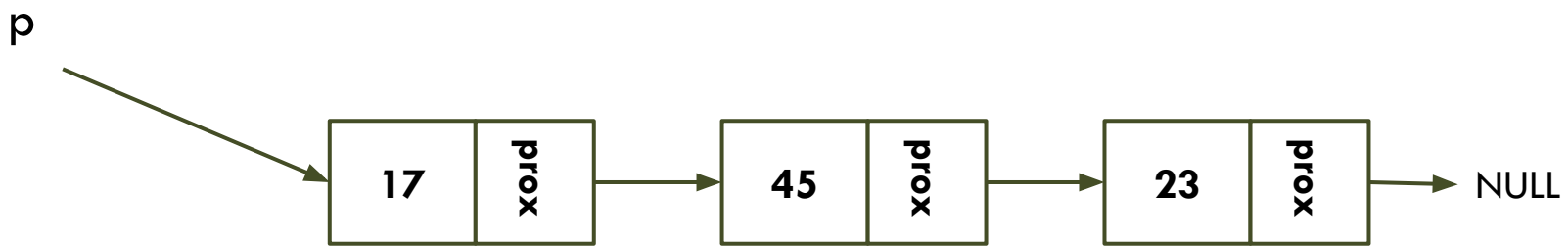


Lista encadeada simples

- **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:

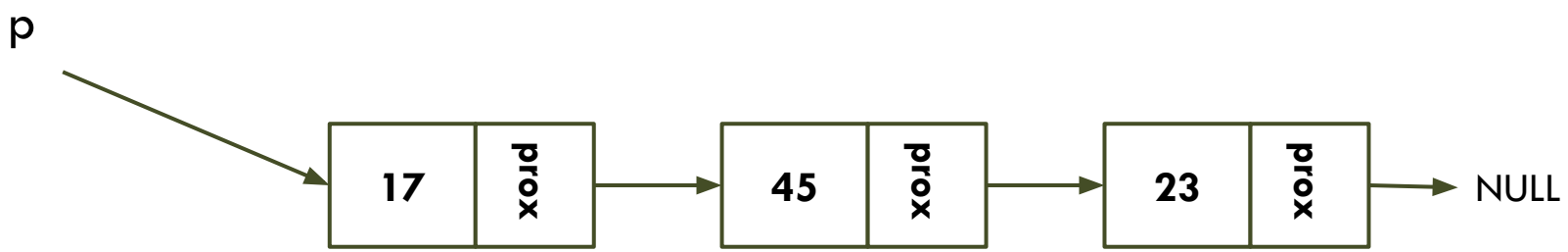


Lista encadeada simples

- **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

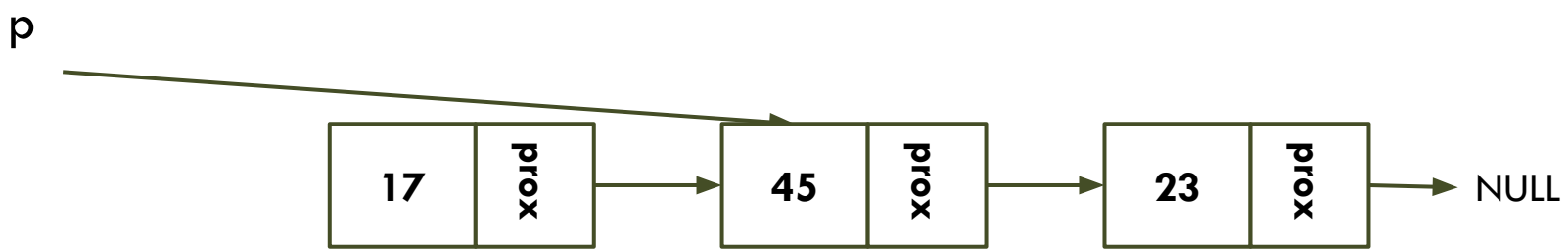


Lista encadeada simples

- **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

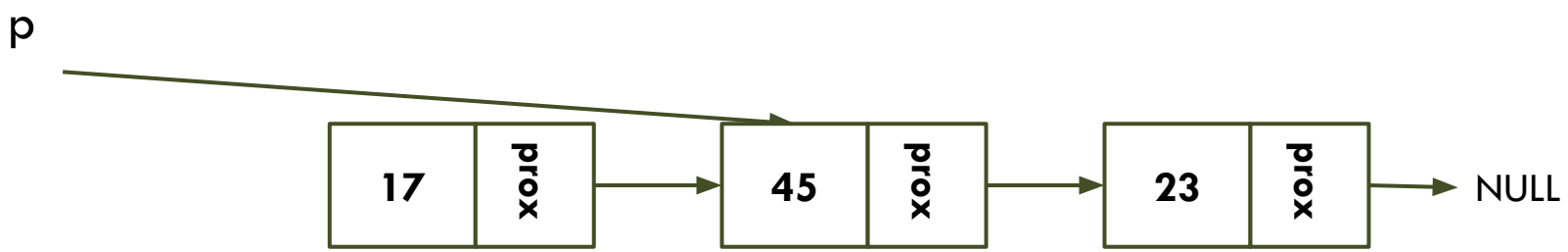


Lista encadeada simples

- **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

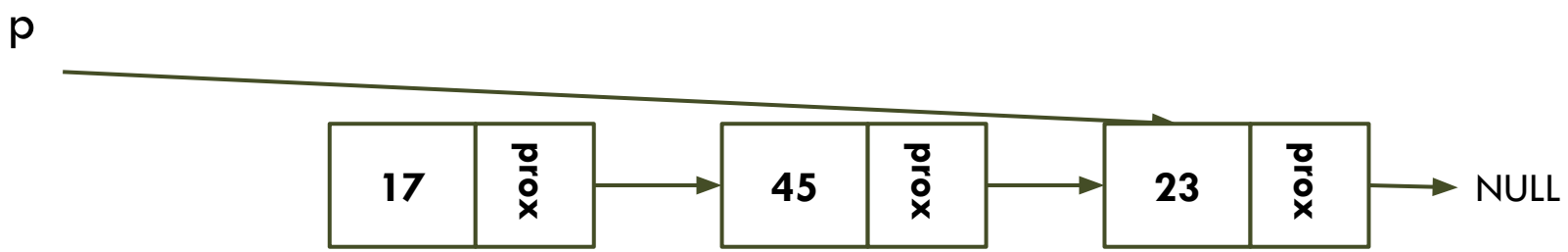


Lista encadeada simples

- **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

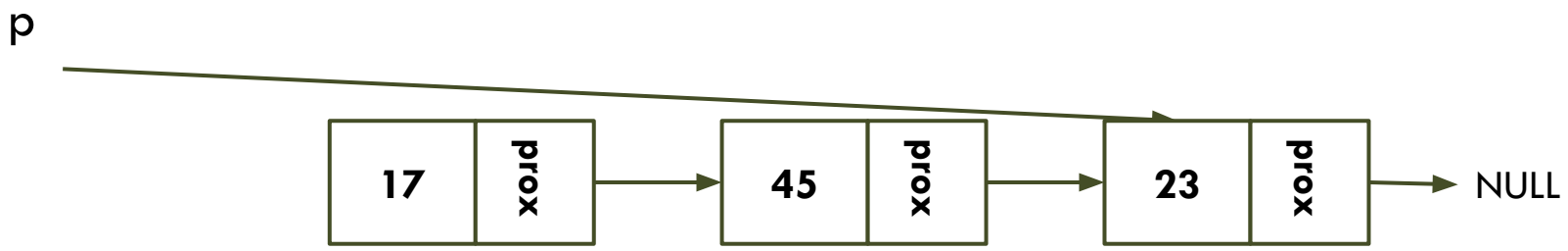


Lista encadeada simples

- **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

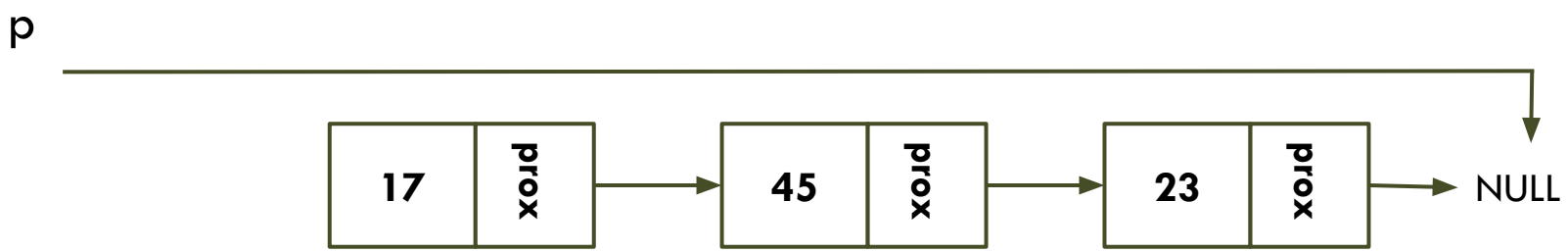


Lista encadeada simples

- **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);
    }
}
```

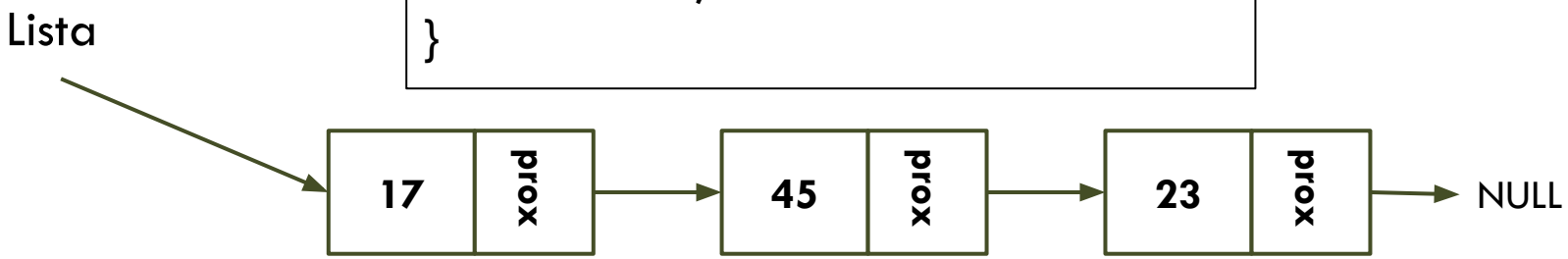
Saída: 17 45 23



Lista encadeada simples

- **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;
}
```

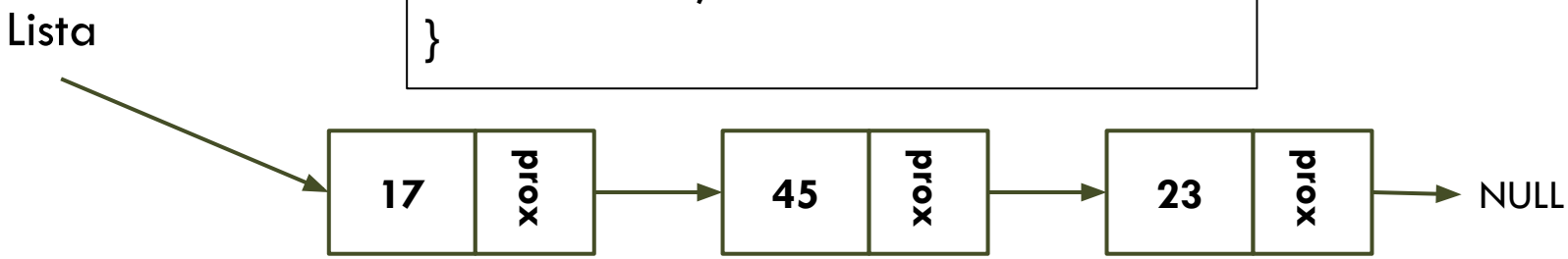


Lista encadeada simples

- **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

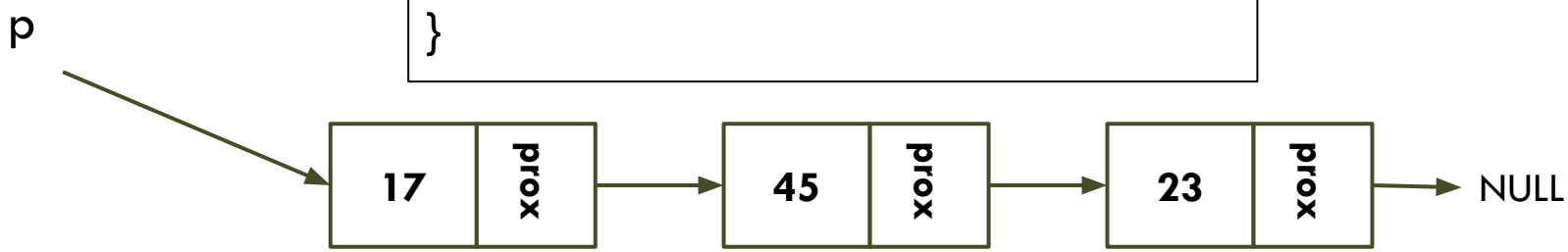


Lista encadeada simples

- **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*

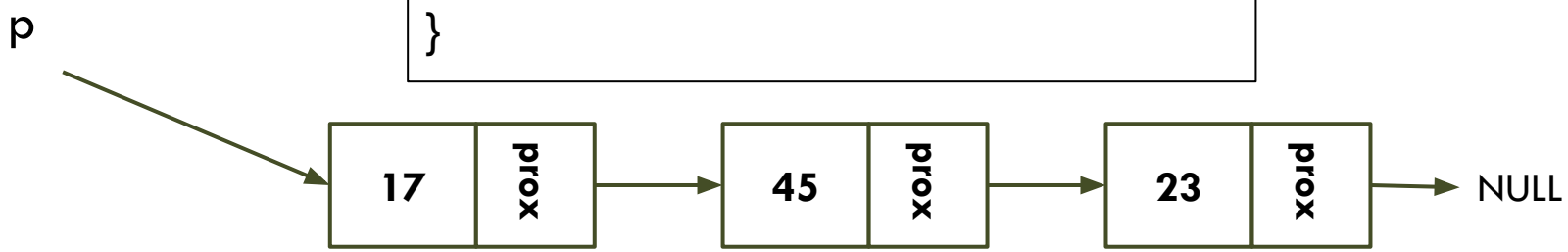


Lista encadeada simples

- **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

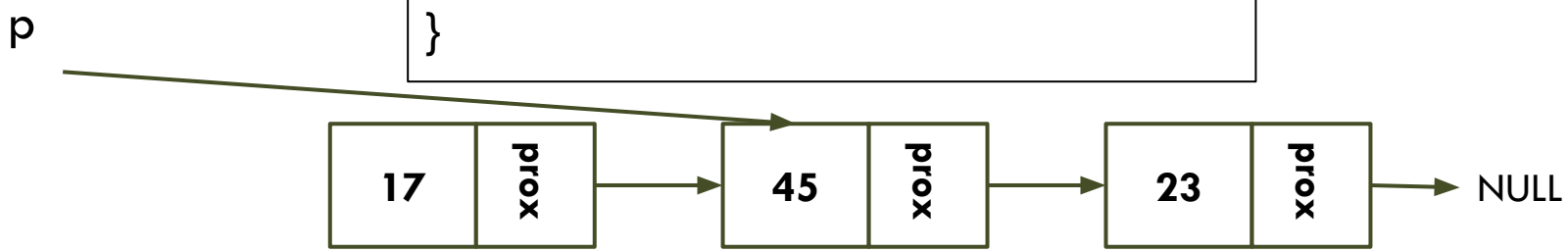


Lista encadeada simples

- **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*

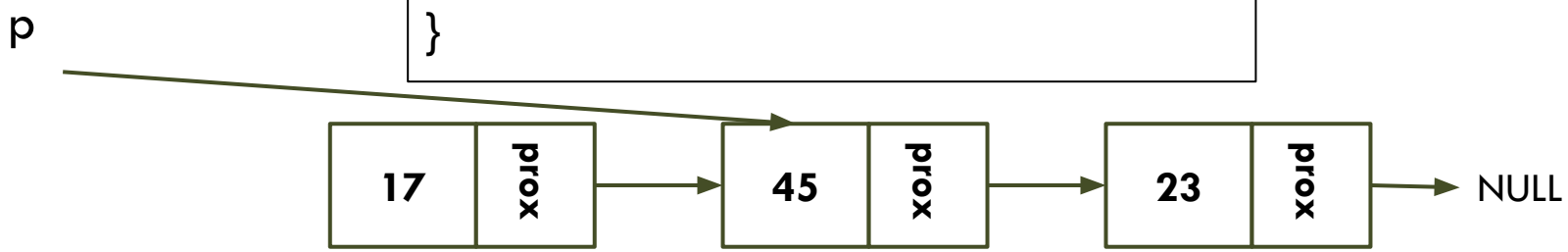


Lista encadeada simples

- **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



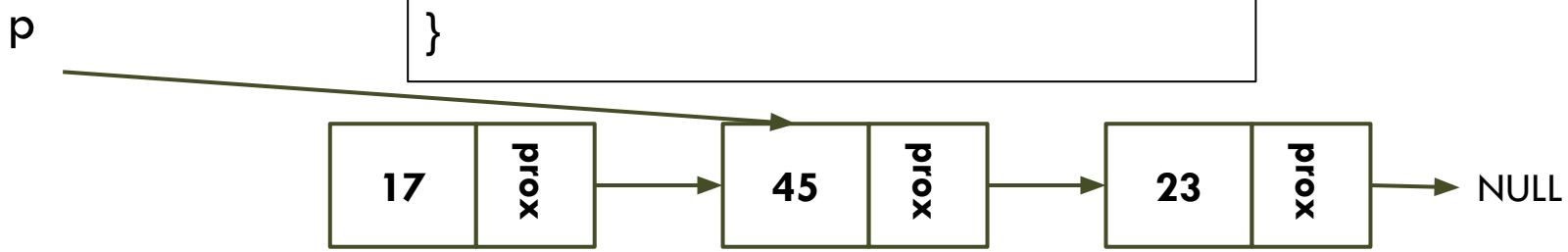
Lista encadeada simples

- **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;
}
```

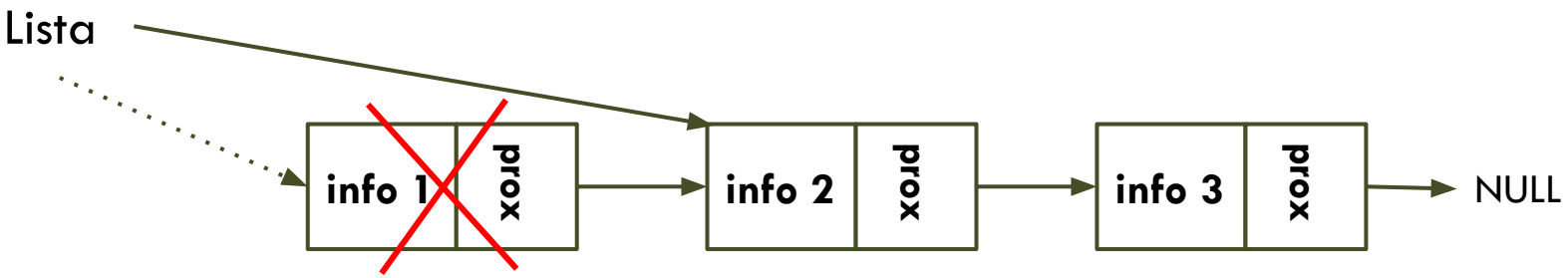
Buscando: 45

Retorna p

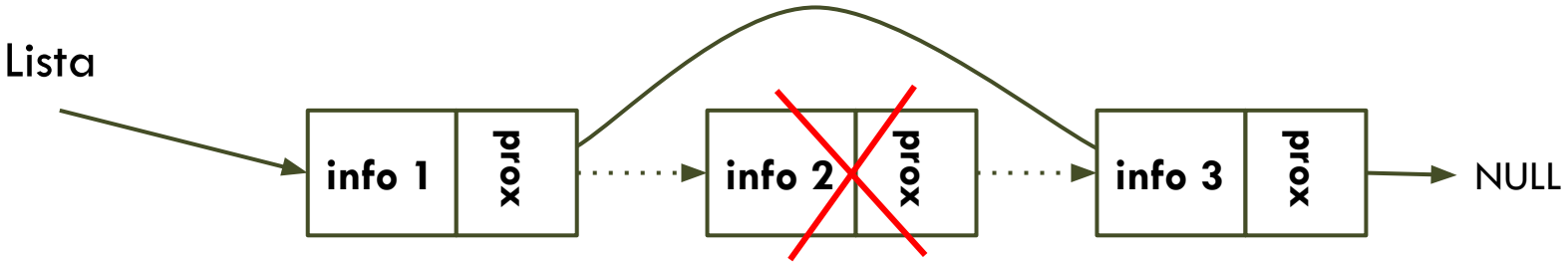


Lista encadeada simples

- **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

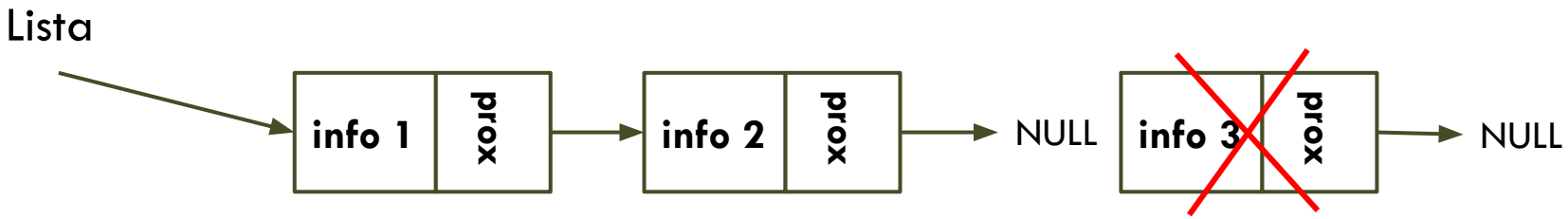


  - Caso seja o elemento do meio, remove o elemento da lista e o ponteiro do elemento anterior aponta para o próximo



Lista encadeada simples

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



Lista encadeada simples

- **Função retira**

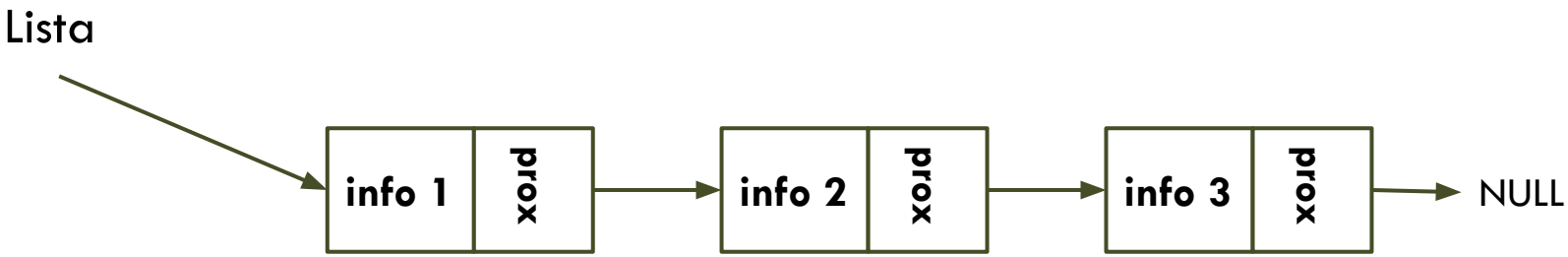
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;

    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

- **Função libera**
  - Liberado todos os elementos (nós) alocados

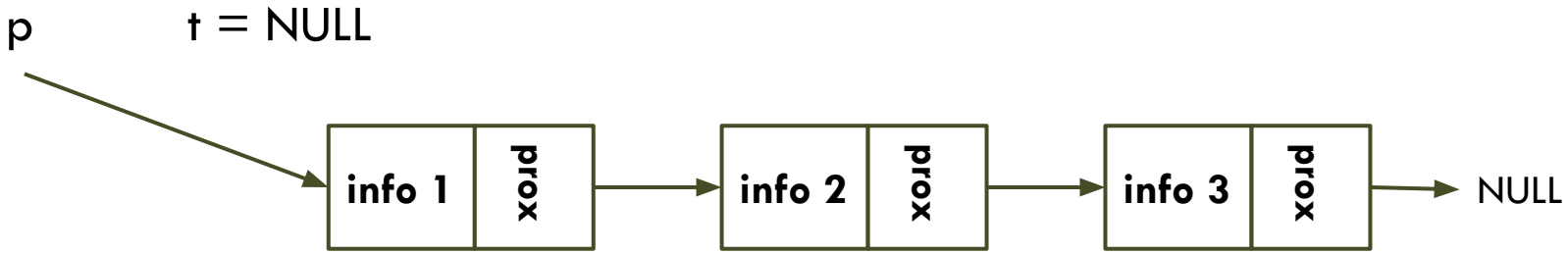
```
Lista* libera (Lista* l){
        Lista* p = l;
        Lista* t = NULL;
        while (p != NULL) {
                t = p->prox;
                free(p);
                p = t;
        }
    return NULL;
}
```



Lista encadeada simples

- **Função libera**
  - Liberado todos os elementos (nós) alocados

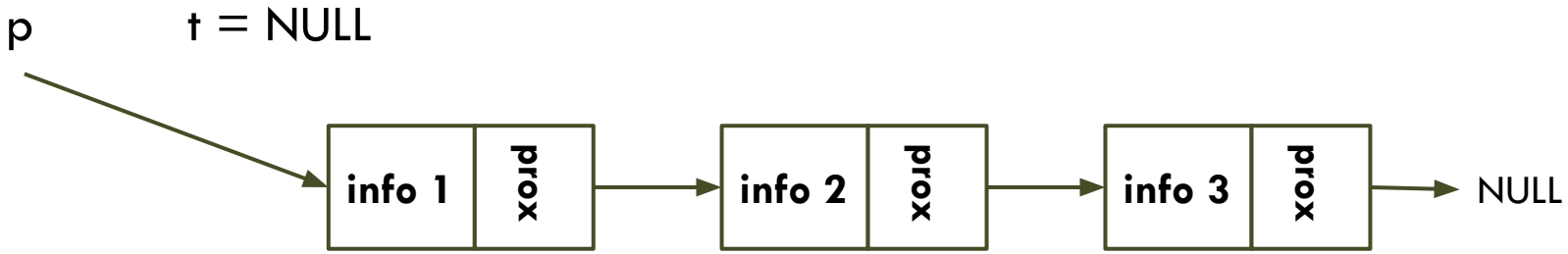
```
Lista* libera (Lista* l){
      Lista* p = l;
      Lista* t = NULL;
       while (p != NULL) {
             t = p->prox;
             free(p);
             p = t;
       }
   return NULL;
}
```



Lista encadeada simples

- **Função libera**
  - Liberado todos os elementos (nós) alocados

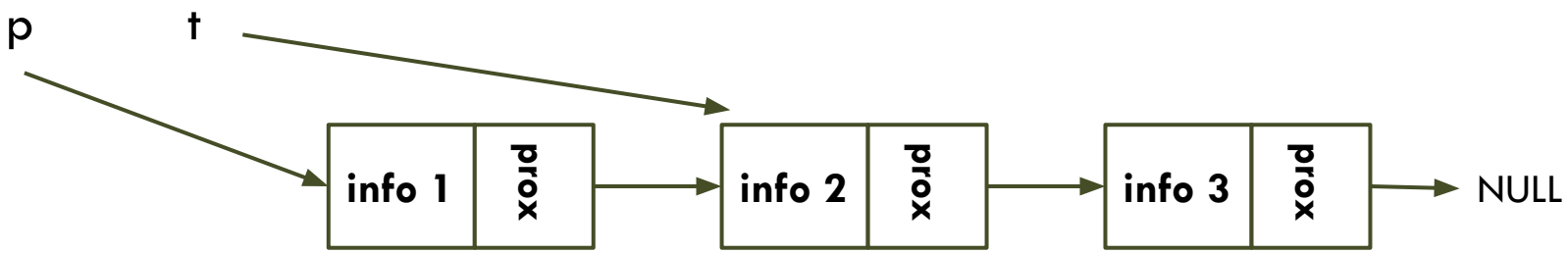
```
Lista* libera (Lista* l){
        Lista* p = l;
        Lista* t = NULL;
        while (p != NULL) {
                t = p->prox;
                free(p);
                p = t;
        }
    return NULL;
}
```



Lista encadeada simples

- **Função libera**
  - Liberado todos os elementos (nós) alocados

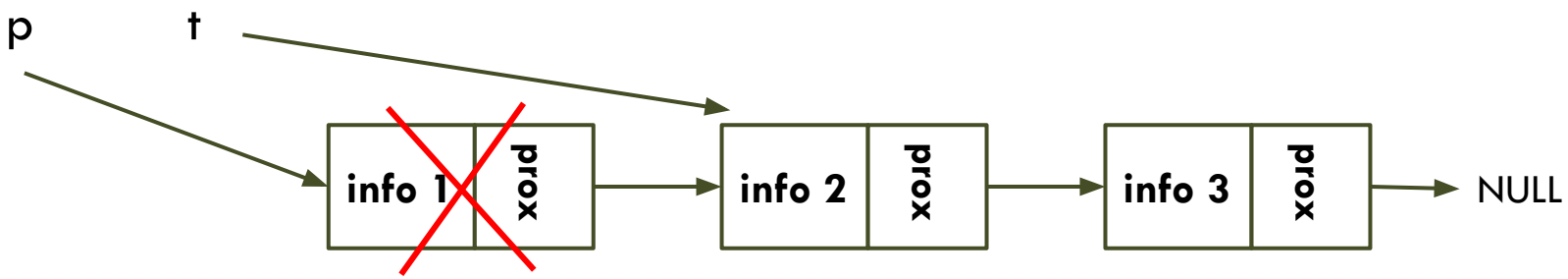
```
Lista* libera (Lista* l){
        Lista* p = l;
        Lista* t = NULL;
        while (p != NULL) {
                t = p->prox;
                free(p);
                p = t;
        }
    return NULL;
}
```



Lista encadeada simples

- **Função libera**
  - Liberado todos os elementos (nós) alocados

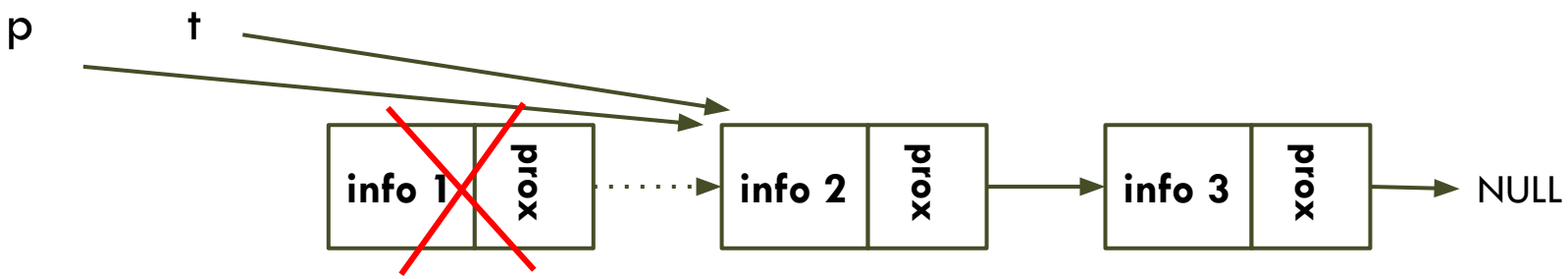
```
Lista* libera (Lista* l){
        Lista* p = l;
        Lista* t = NULL;
        while (p != NULL) {
                t = p->prox;
                free(p);
                p = t;
        }
    return NULL;
}
```



Lista encadeada simples

- **Função libera**
  - Liberado todos os elementos (nós) alocados

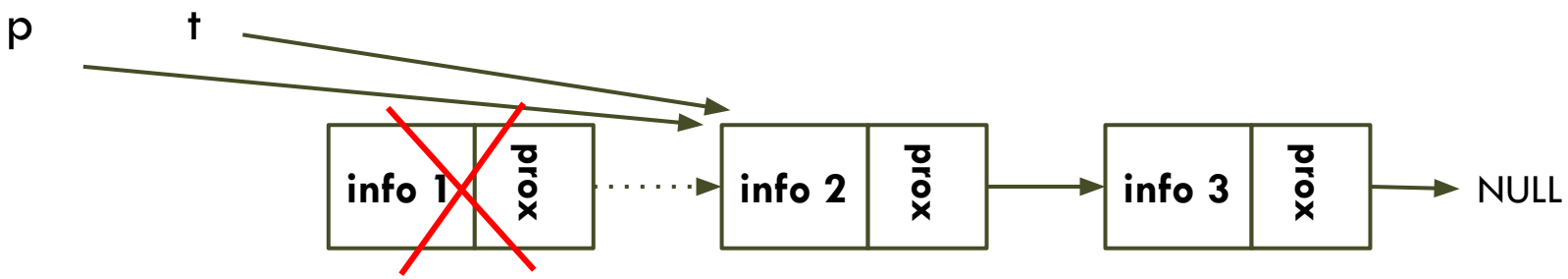
```
Lista* libera (Lista* l){
        Lista* p = l;
        Lista* t = NULL;
        while (p != NULL) {
                t = p->prox;
                free(p);
                p = t;
        }
    return NULL;
}
```



Lista encadeada simples

- **Função libera**
  - Liberado todos os elementos (nós) alocados

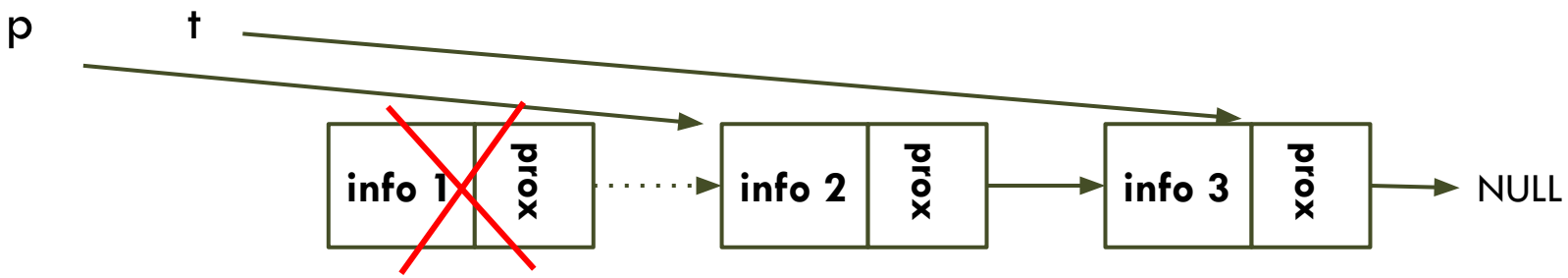
```
Lista* libera (Lista* l){
        Lista* p = l;
        Lista* t = NULL;
        while (p != NULL) {
                t = p->prox;
                free(p);
                p = t;
        }
    return NULL;
}
```



Lista encadeada simples

- **Função libera**
  - Liberado todos os elementos (nós) alocados

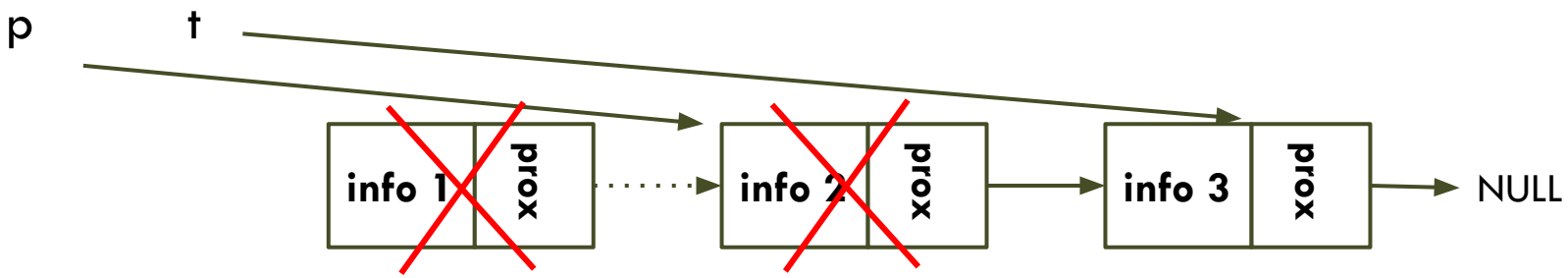
```
Lista* libera (Lista* l){
        Lista* p = l;
        Lista* t = NULL;
        while (p != NULL) {
                t = p->prox;
                free(p);
                p = t;
        }
    return NULL;
}
```



Lista encadeada simples

- **Função libera**
  - Liberado todos os elementos (nós) alocados

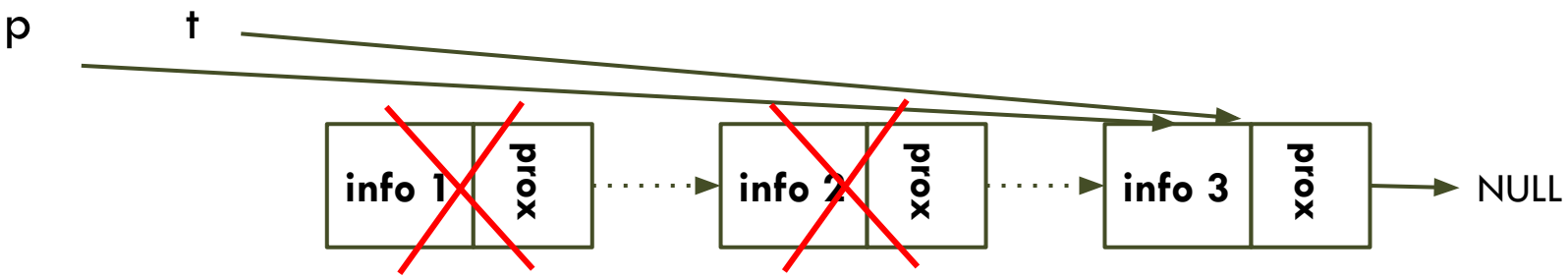
```
Lista* libera (Lista* l){
        Lista* p = l;
        Lista* t = NULL;
        while (p != NULL) {
                t = p->prox;
                free(p);
                p = t;
        }
    return NULL;
}
```



Lista encadeada simples

- **Função libera**
  - Liberado todos os elementos (nós) alocados

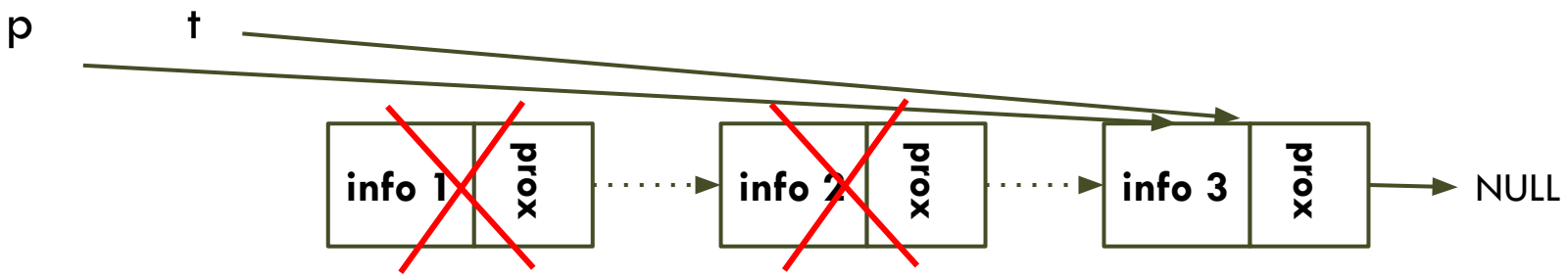
```
Lista* libera (Lista* l){
        Lista* p = l;
        Lista* t = NULL;
        while (p != NULL) {
                t = p->prox;
                free(p);
                p = t;
        }
    return NULL;
}
```



Lista encadeada simples

- **Função libera**
  - Liberado todos os elementos (nós) alocados

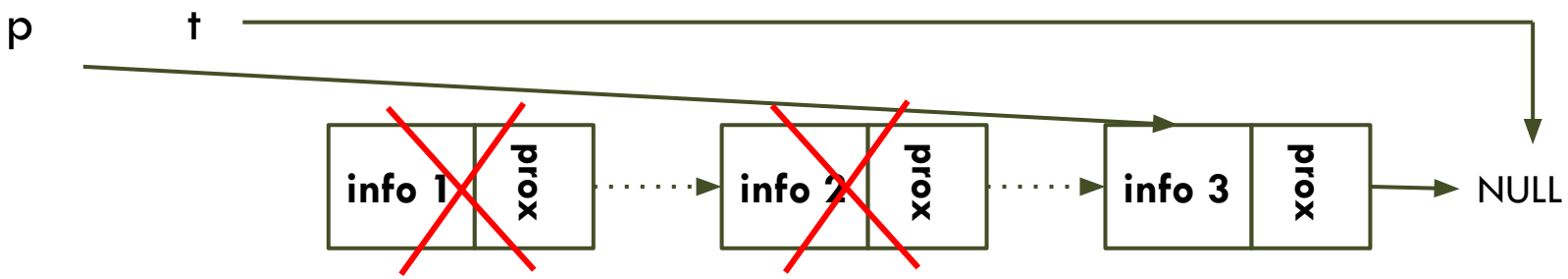
```
Lista* libera (Lista* l){
        Lista* p = l;
        Lista* t = NULL;
        while (p != NULL) {
                t = p->prox;
                free(p);
                p = t;
        }
    return NULL;
}
```



Lista encadeada simples

- **Função libera**
  - Liberado todos os elementos (nós) alocados

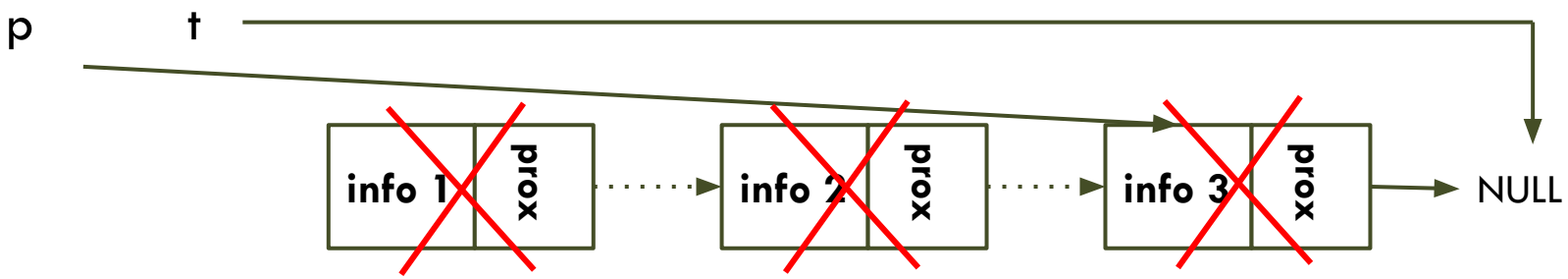
```
Lista* libera (Lista* l){
        Lista* p = l;
        Lista* t = NULL;
        while (p != NULL) {
                t = p->prox;
                free(p);
                p = t;
        }
    return NULL;
}
```



Lista encadeada simples

- **Função libera**
  - Liberado todos os elementos (nós) alocados

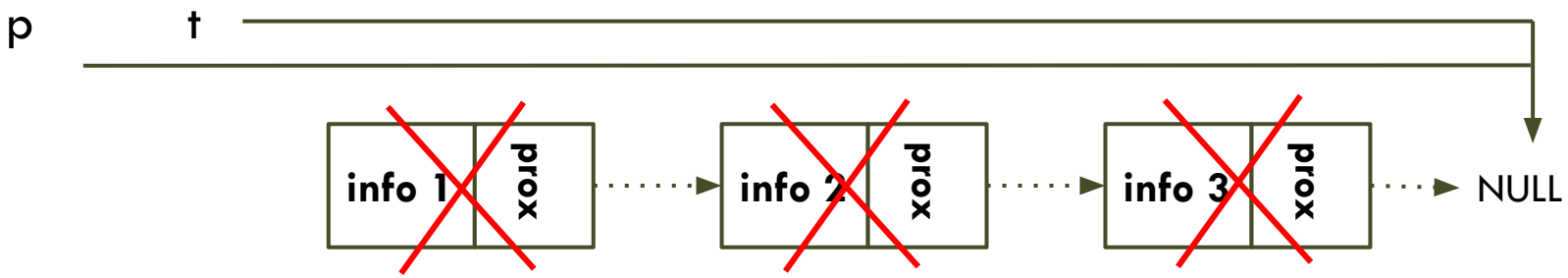
```
Lista* libera (Lista* l){
        Lista* p = l;
        Lista* t = NULL;
        while (p != NULL) {
                t = p->prox;
                free(p);
                p = t;
        }
    return NULL;
}
```



Lista encadeada simples

- **Função libera**
  - Liberado todos os elementos (nós) alocados

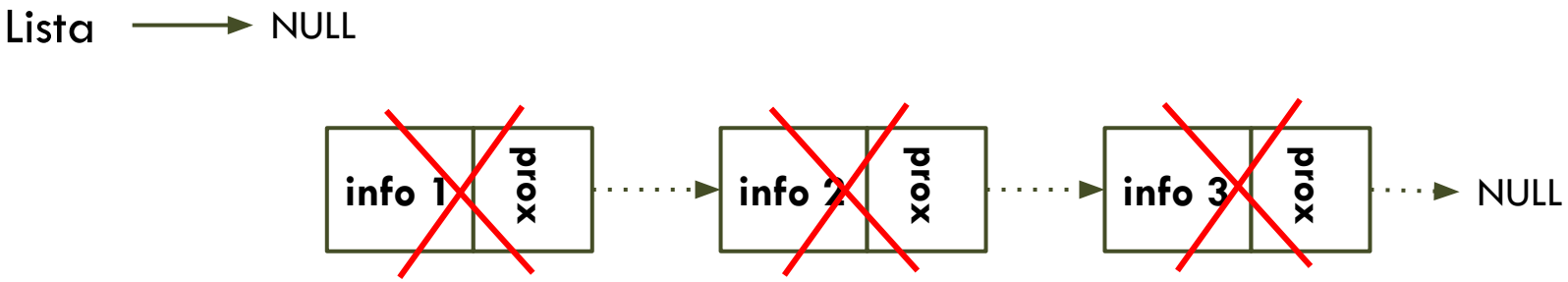
```
Lista* libera (Lista* l){
        Lista* p = l;
        Lista* t = NULL;
        while (p != NULL) {
                t = p->prox;
                free(p);
                p = t;
        }
    return NULL;
}
```



Lista encadeada simples

- **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

- **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

Passo 1

lista = lixo

Passo 2

lista = NULL

main()

```
Lista* lista;
lista = inicializa();
```

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

Lista encadeada simples

Estudo de caso - insere

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

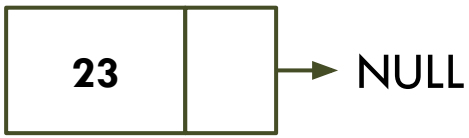
Estudo de caso - insere

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

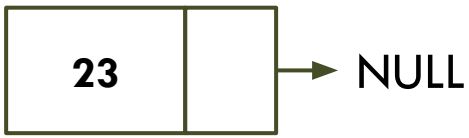
Estudo de caso - insere

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

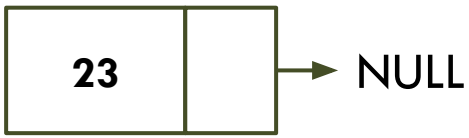
Estudo de caso - insere

lista = NULL

main()

lista = insere(lista, 23);

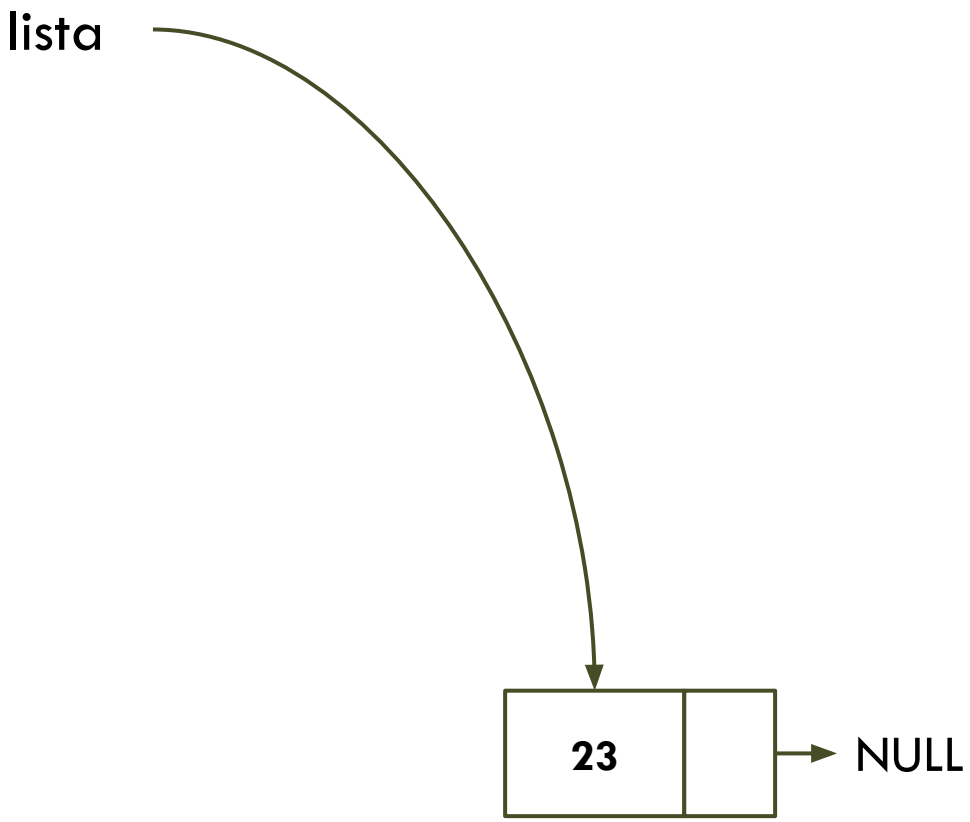
```
Lista* insere (Lista* l, int valor){
        Lista* novo = alocaNo(valor);
        novo->prox = l;
        return novo;
}
```




```
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



main()

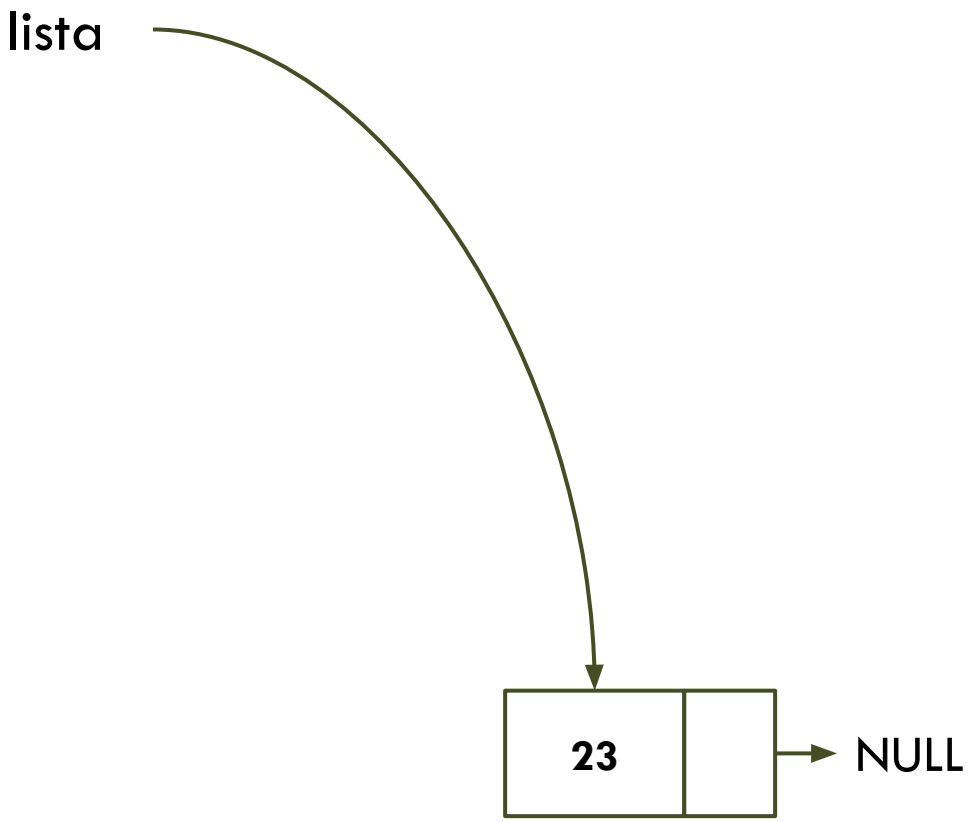
```
lista = insere(lista, 23);
```

```
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

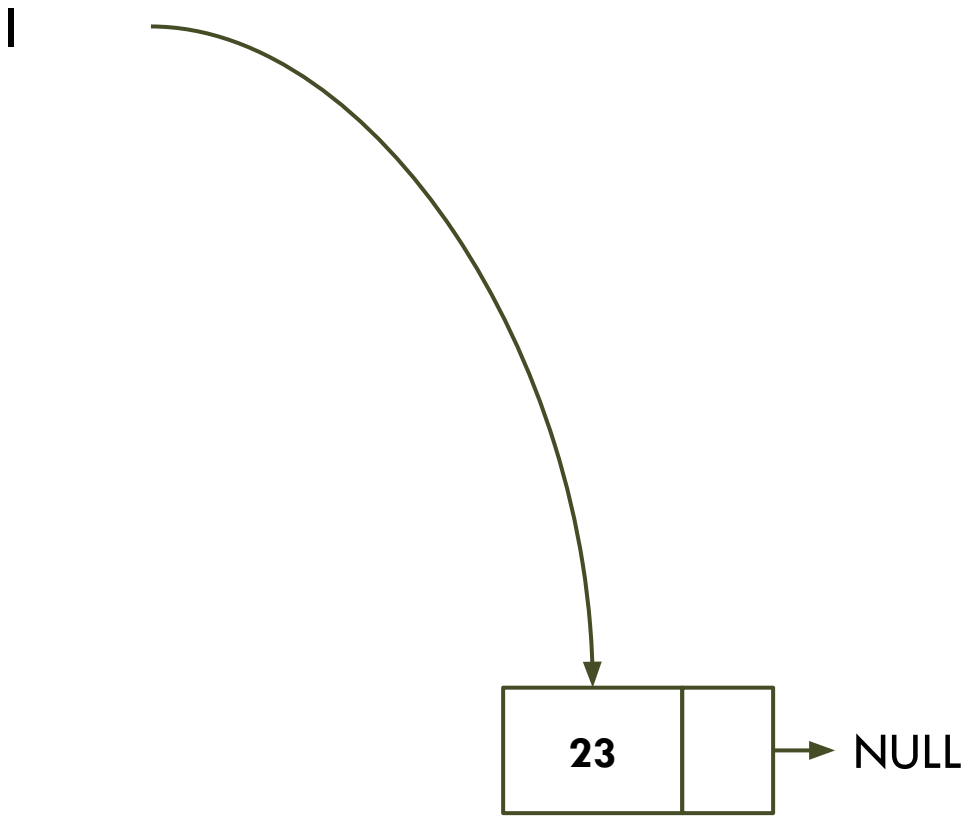


main()

lista = insere(lista, 45);

Lista encadeada simples

Estudo de caso - insere



main()

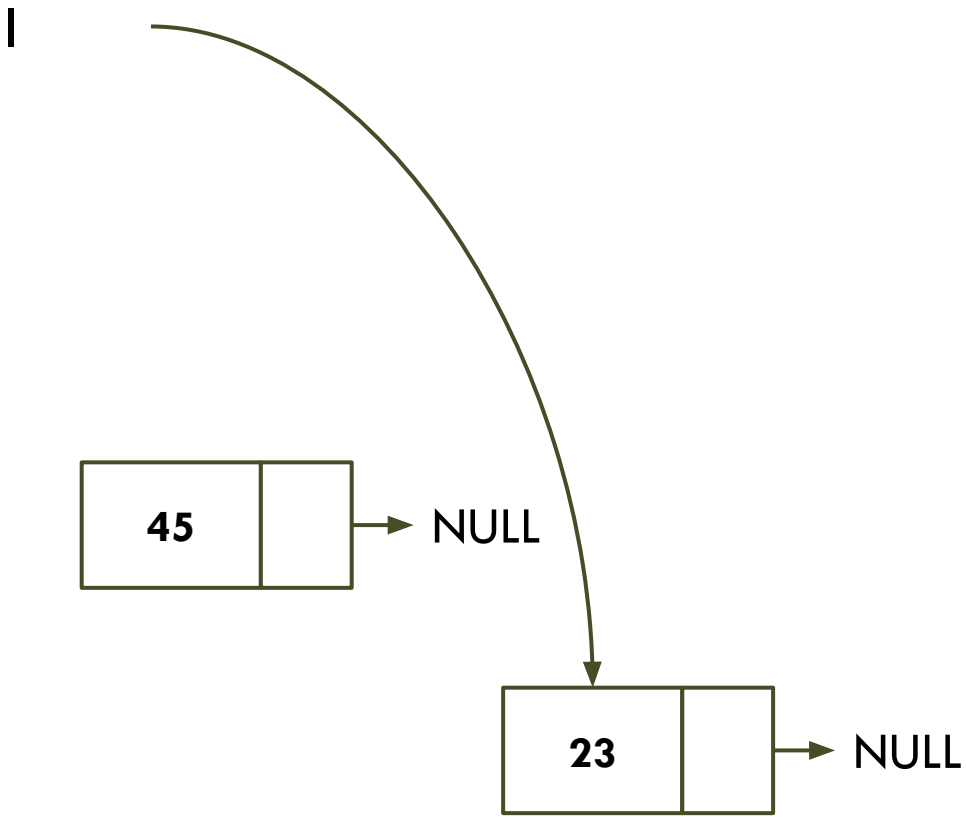
```
lista = insere(lista, 45);
```

```
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



main()

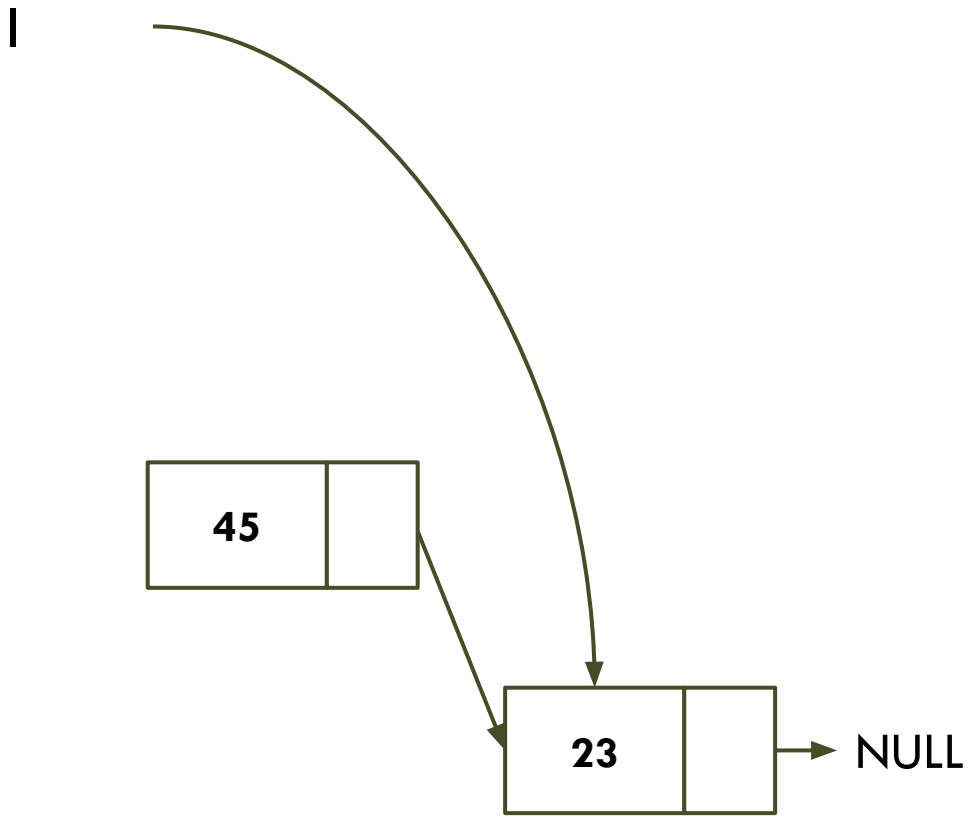
```
lista = insere(lista, 45);
```

```
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



main()

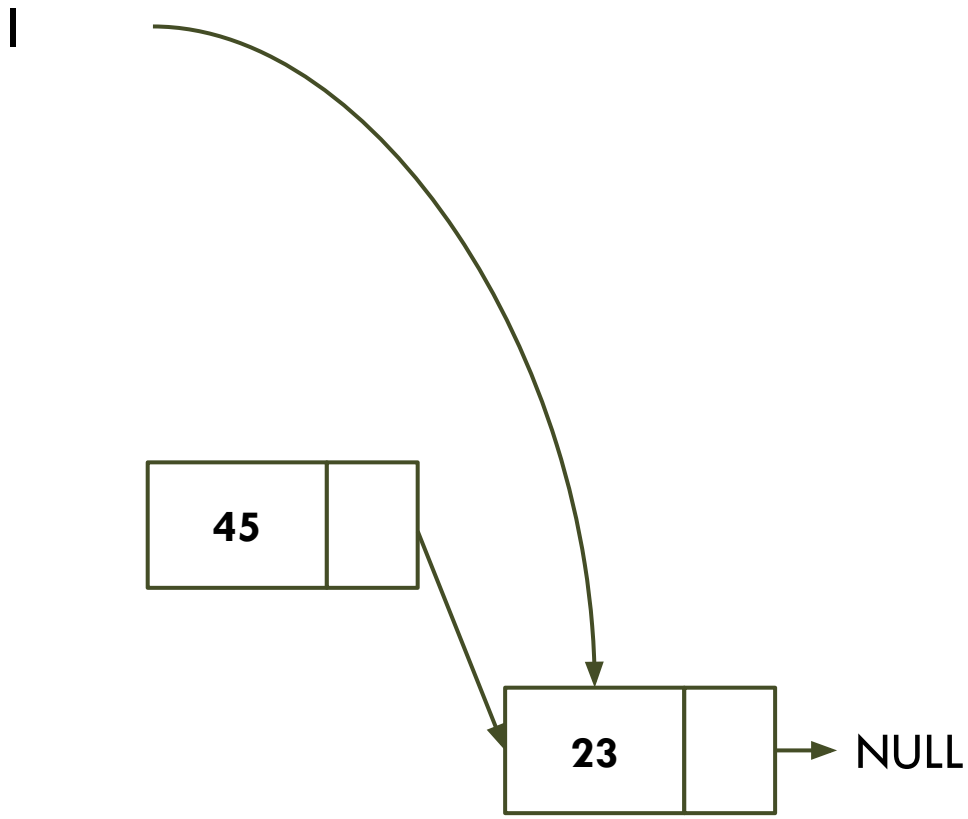
```
lista = insere(lista, 45);
```

```
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



main()

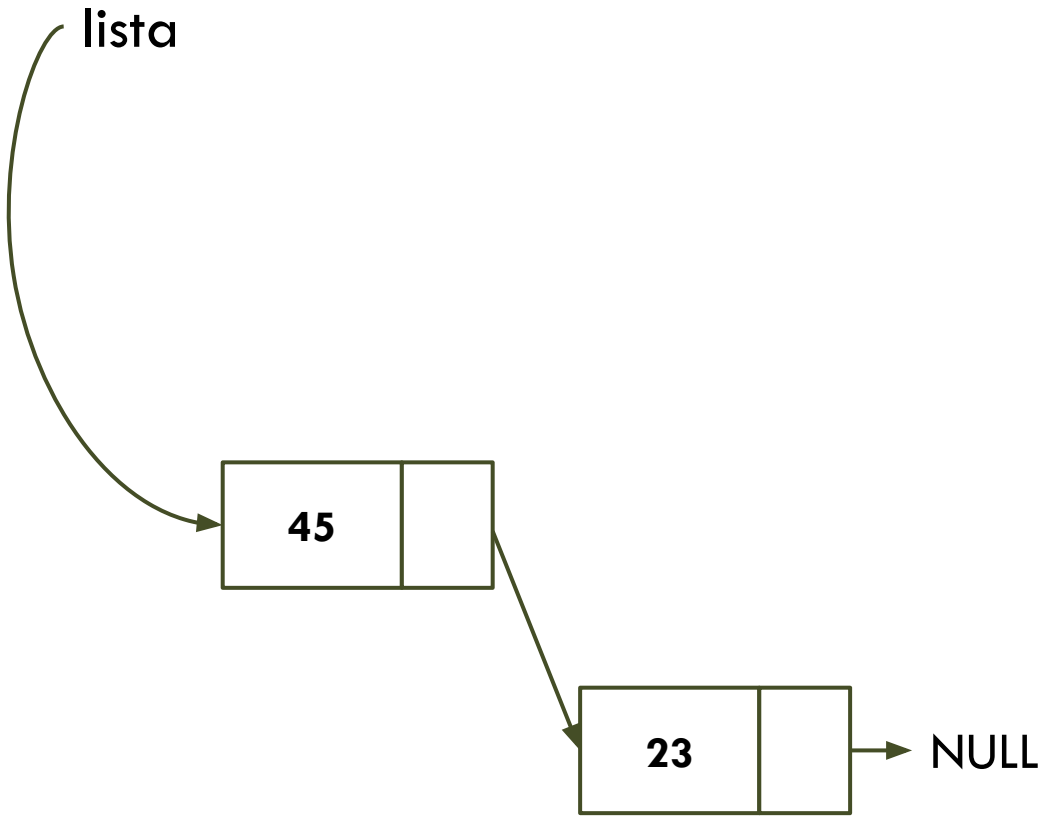
```
lista = insere(lista, 45);
```

```
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



main()

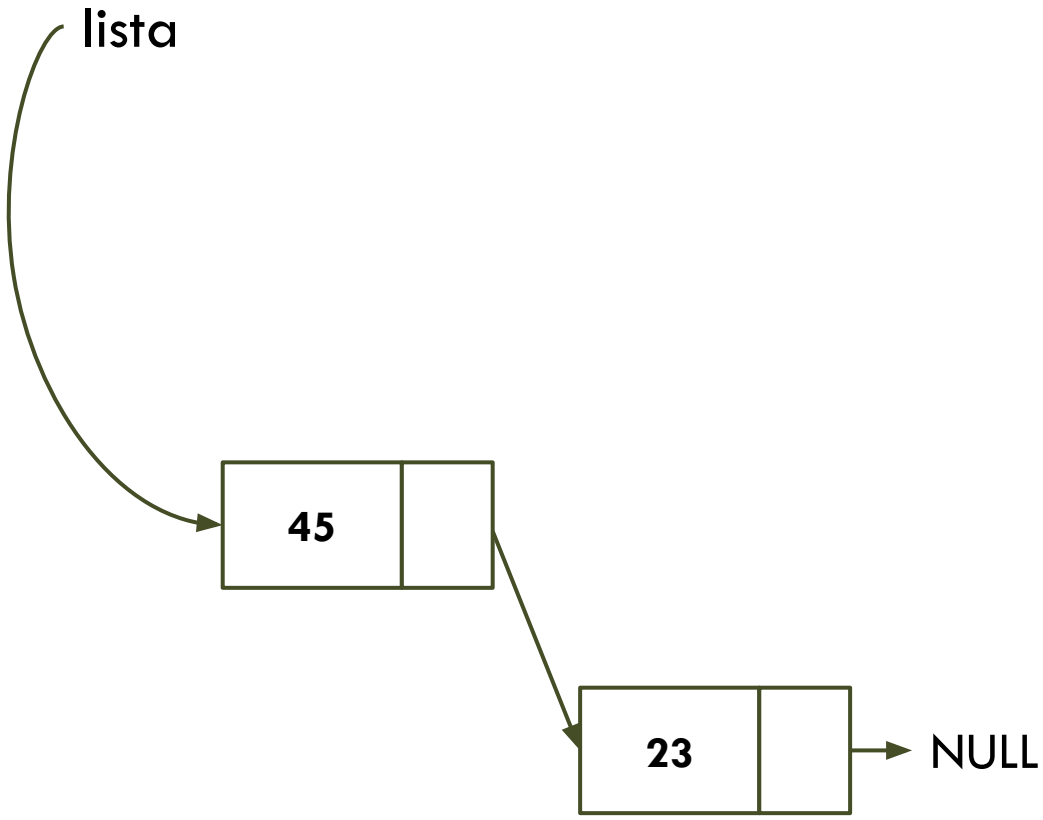
```
lista = insere(lista, 45);
```

```
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

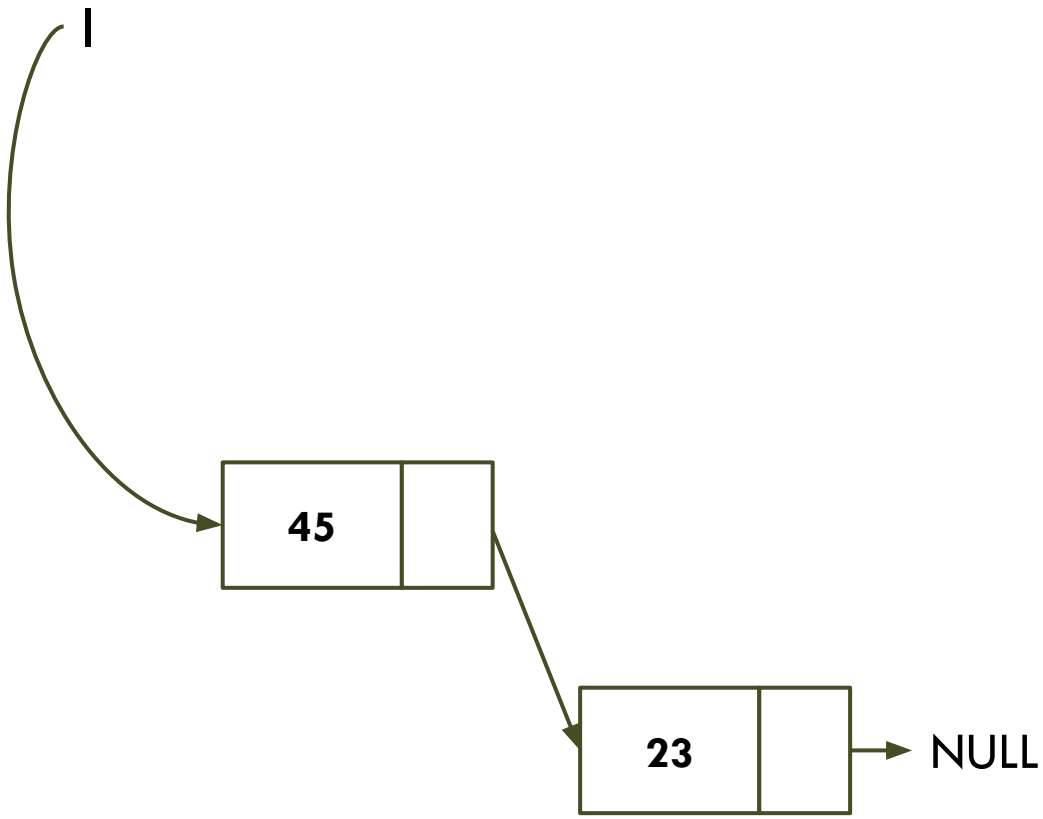


main()

lista = insere(lista, 17);

Lista encadeada simples

Estudo de caso - insere



main()

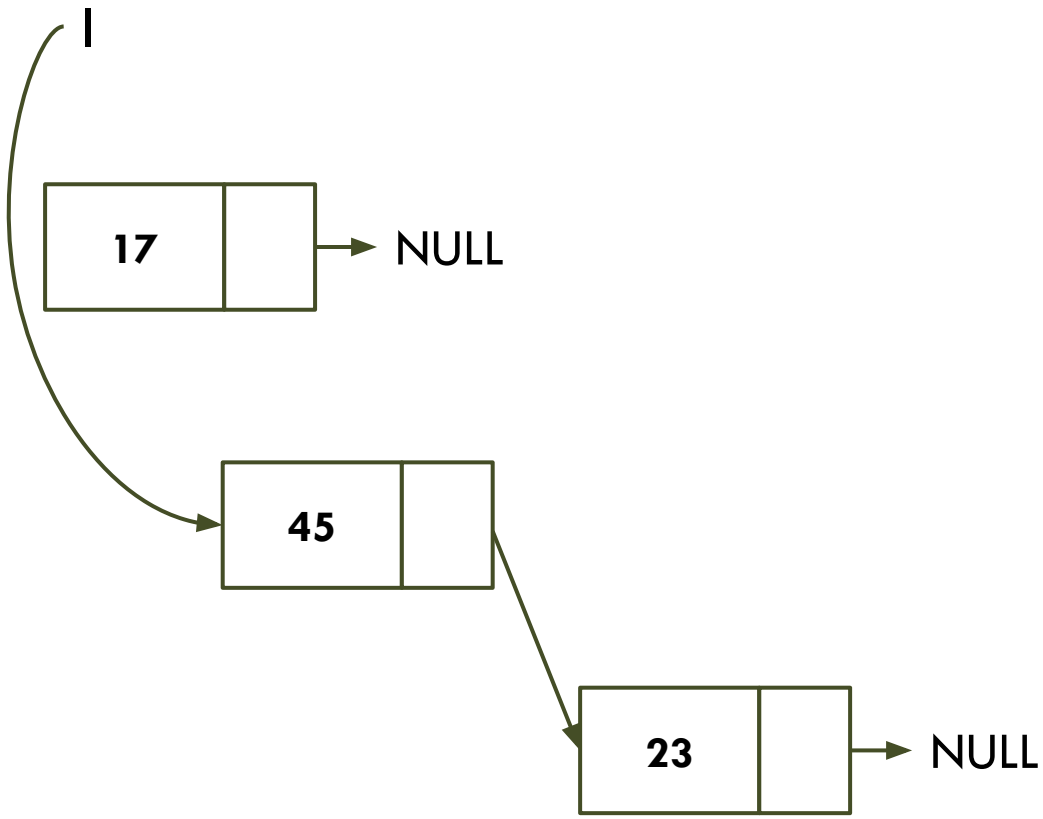
```
lista = insere(lista, 17);
```

```
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



main()

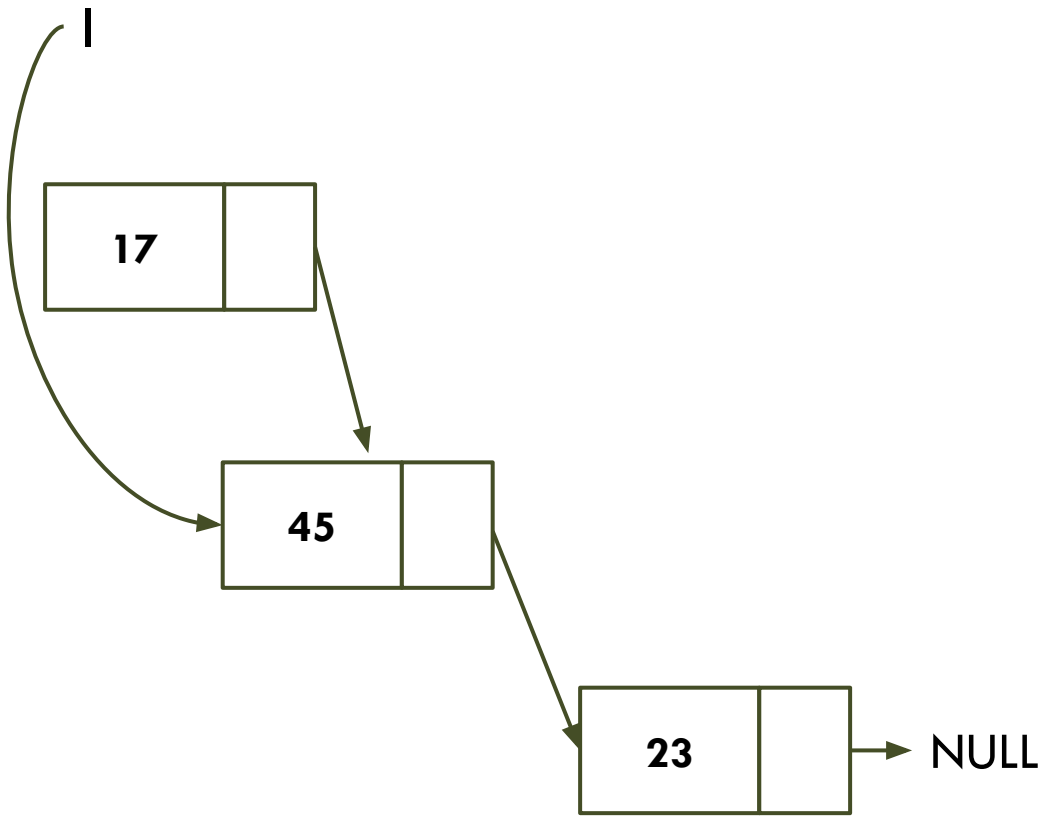
```
lista = insere(lista, 17);
```

```
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



main()

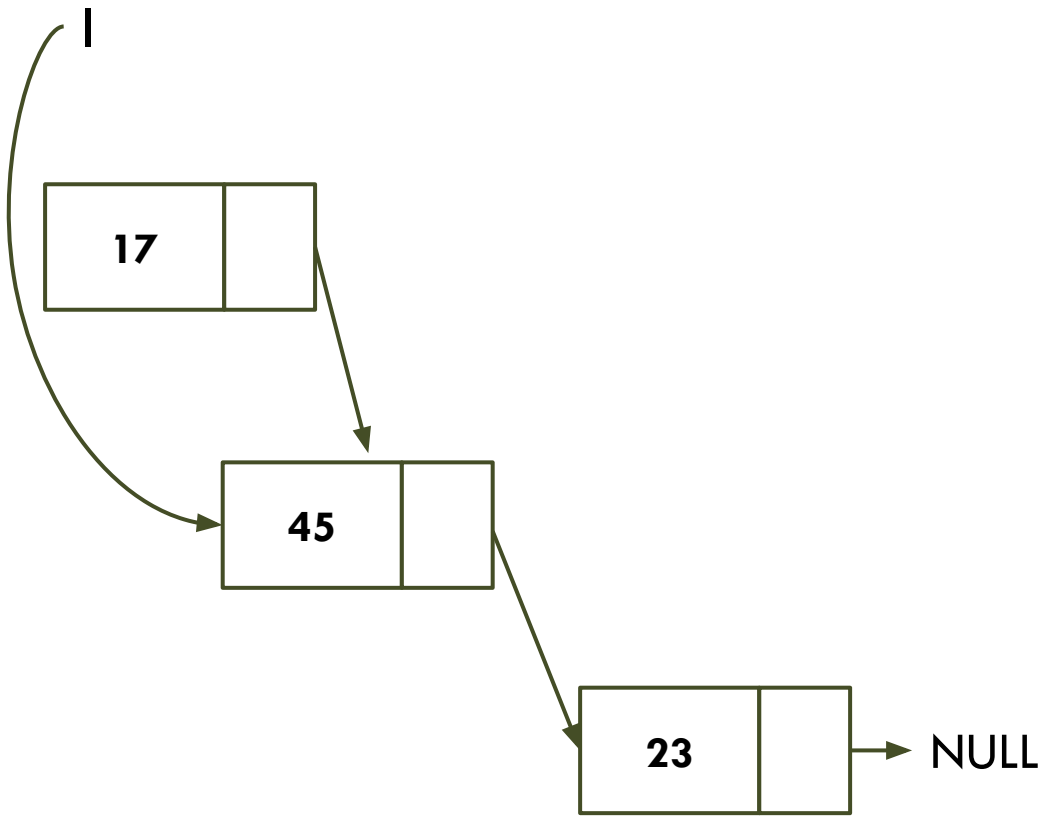
```
lista = insere(lista, 17);
```

```
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



main()

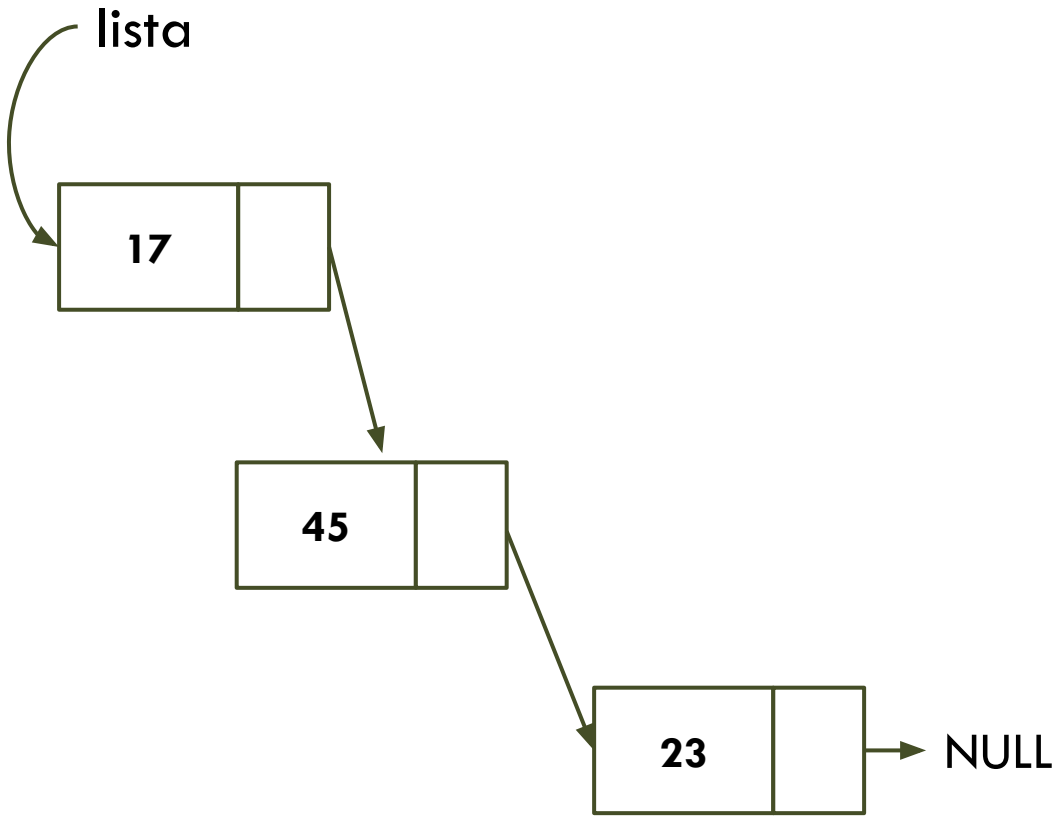
```
lista = insere(lista, 17);
```

```
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



main()

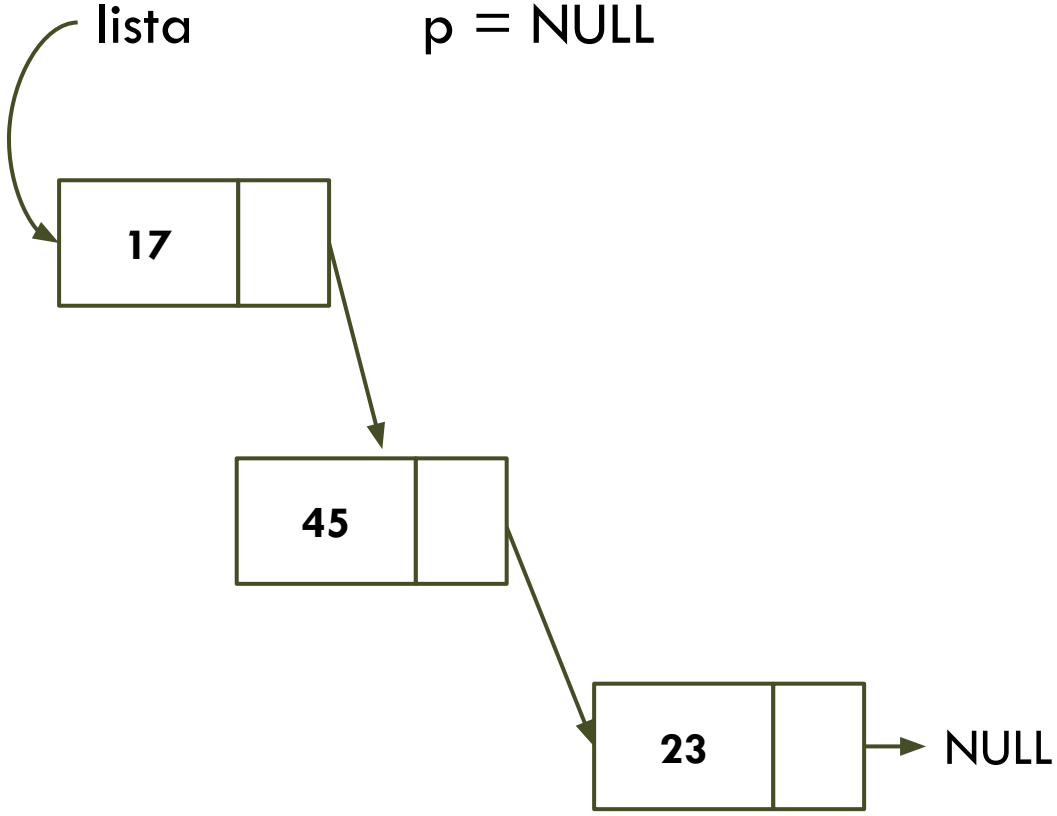
```
lista = insere(lista, 17);
```

```
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



main()

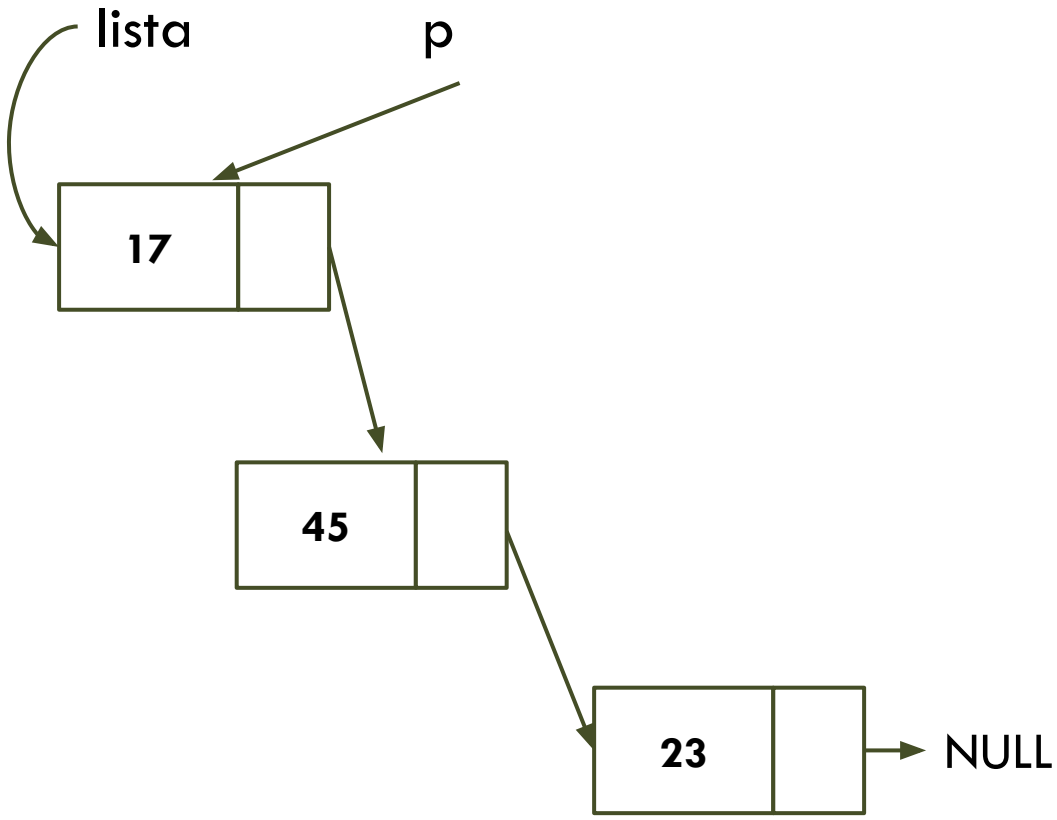
imprime(lista);

```
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



main()

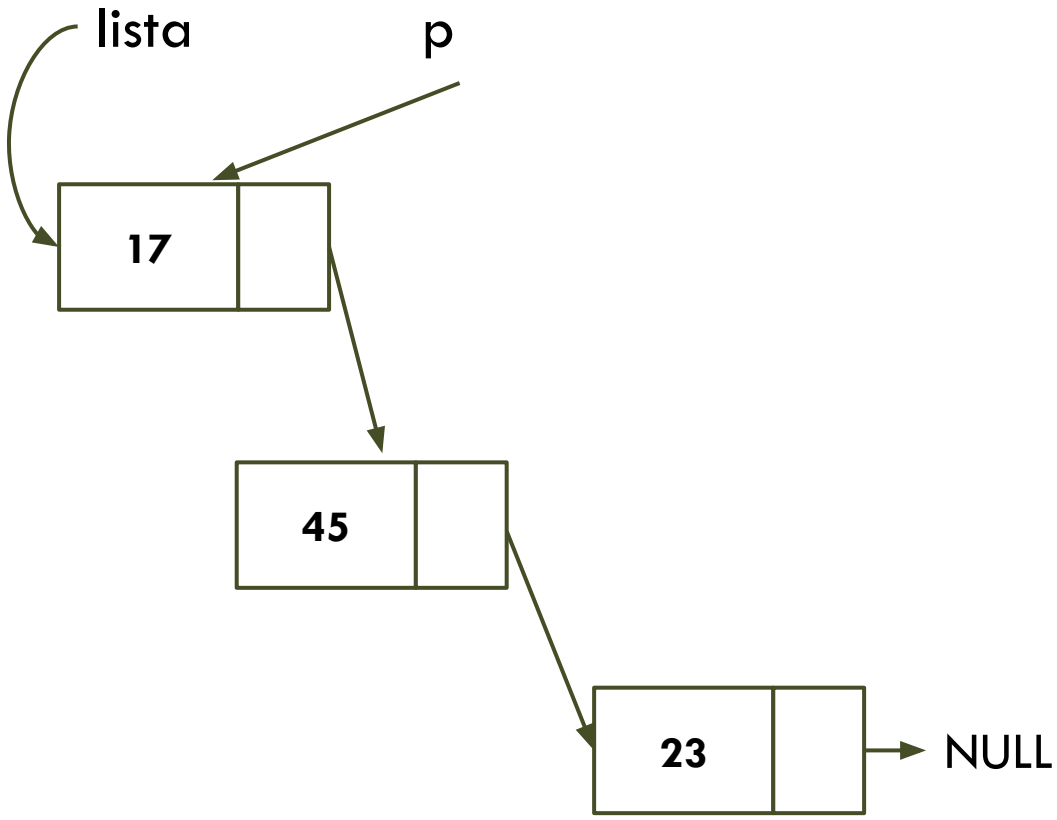
imprime(lista);

```
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



main()

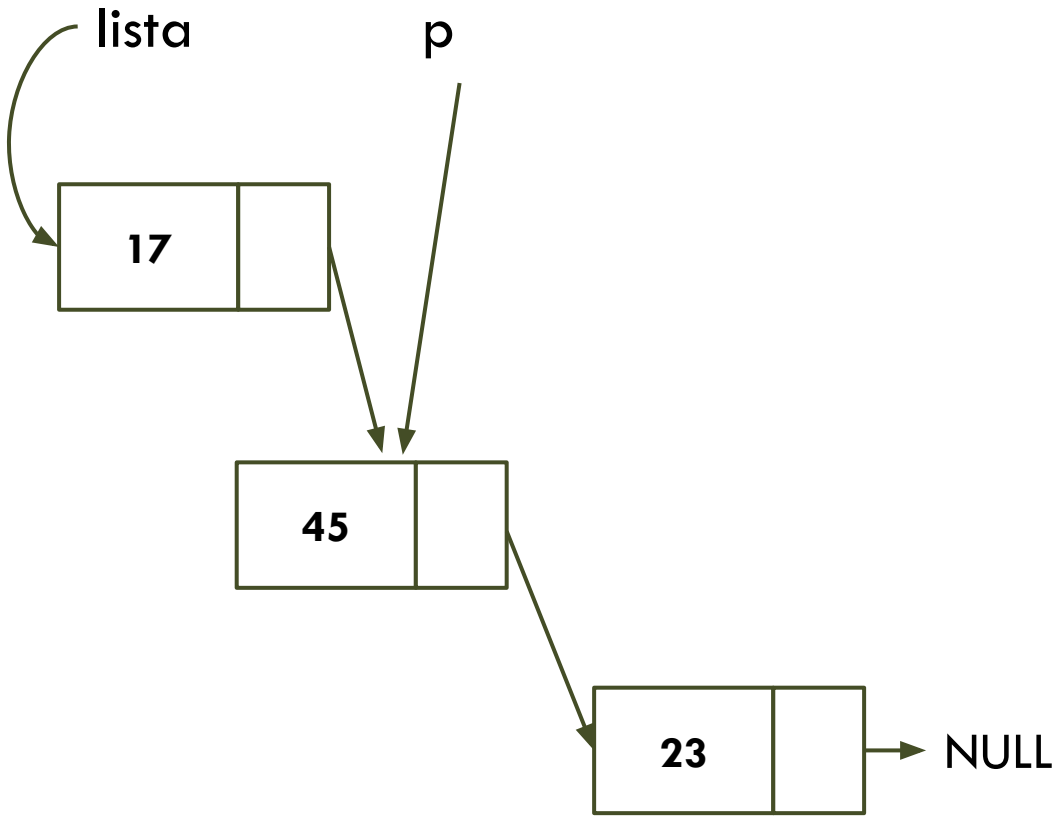
```
imprime(lista);
```

```
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



main()

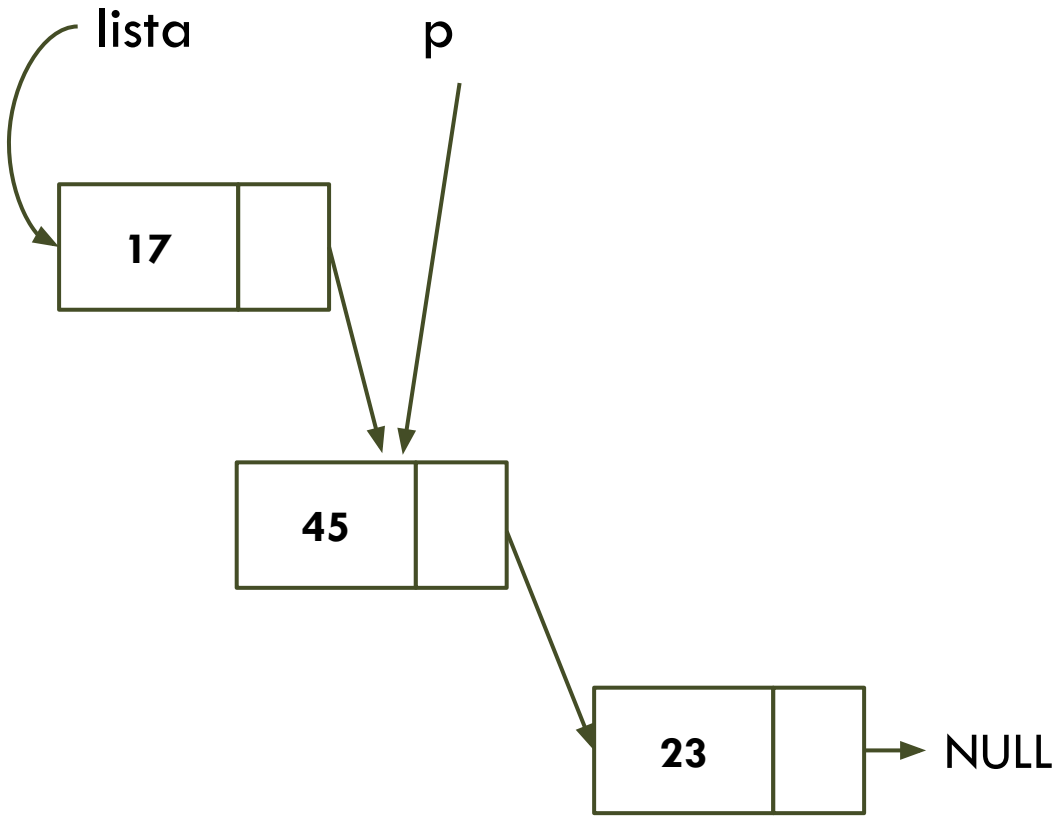
```
imprime(lista);
```

```
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



main()

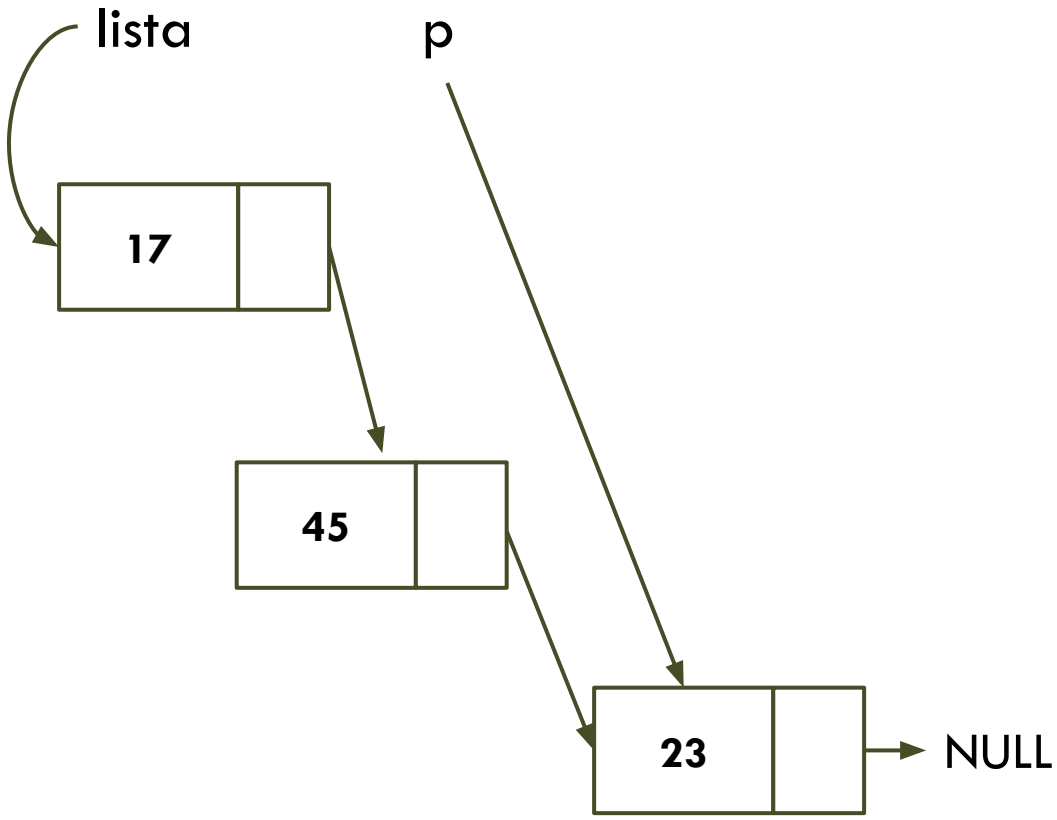
imprime(lista);

```
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



main()

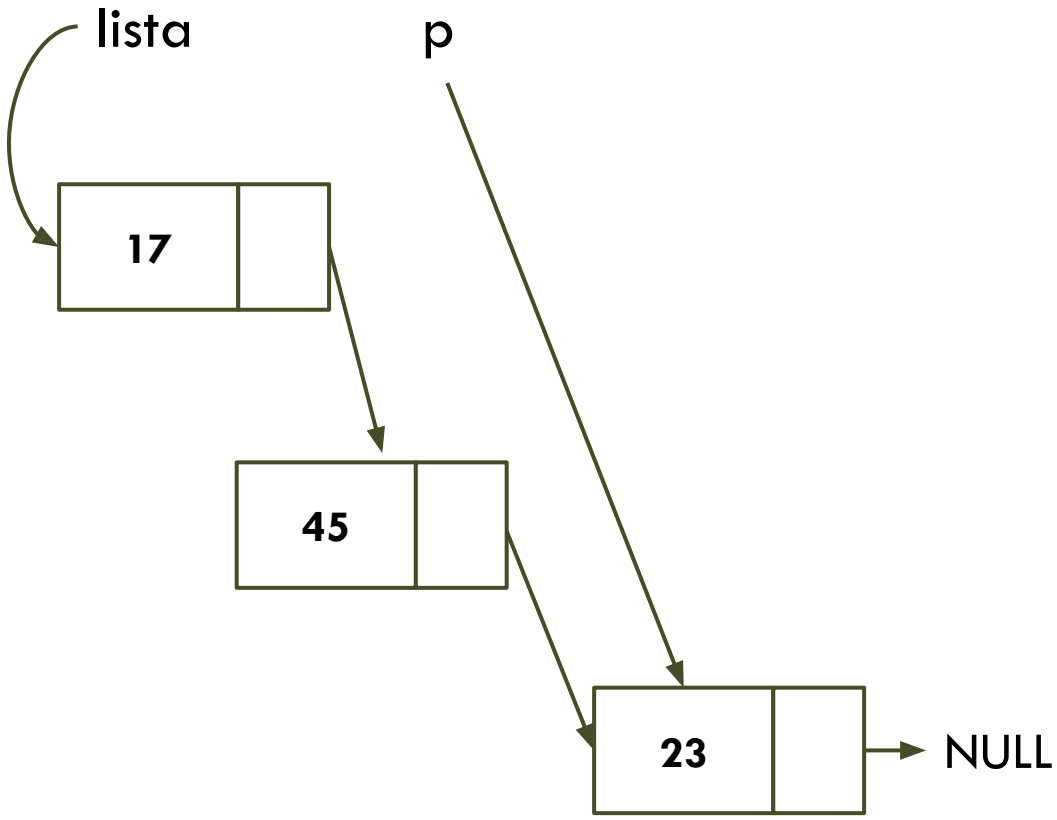
```
imprime(lista);
```

```
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



main()

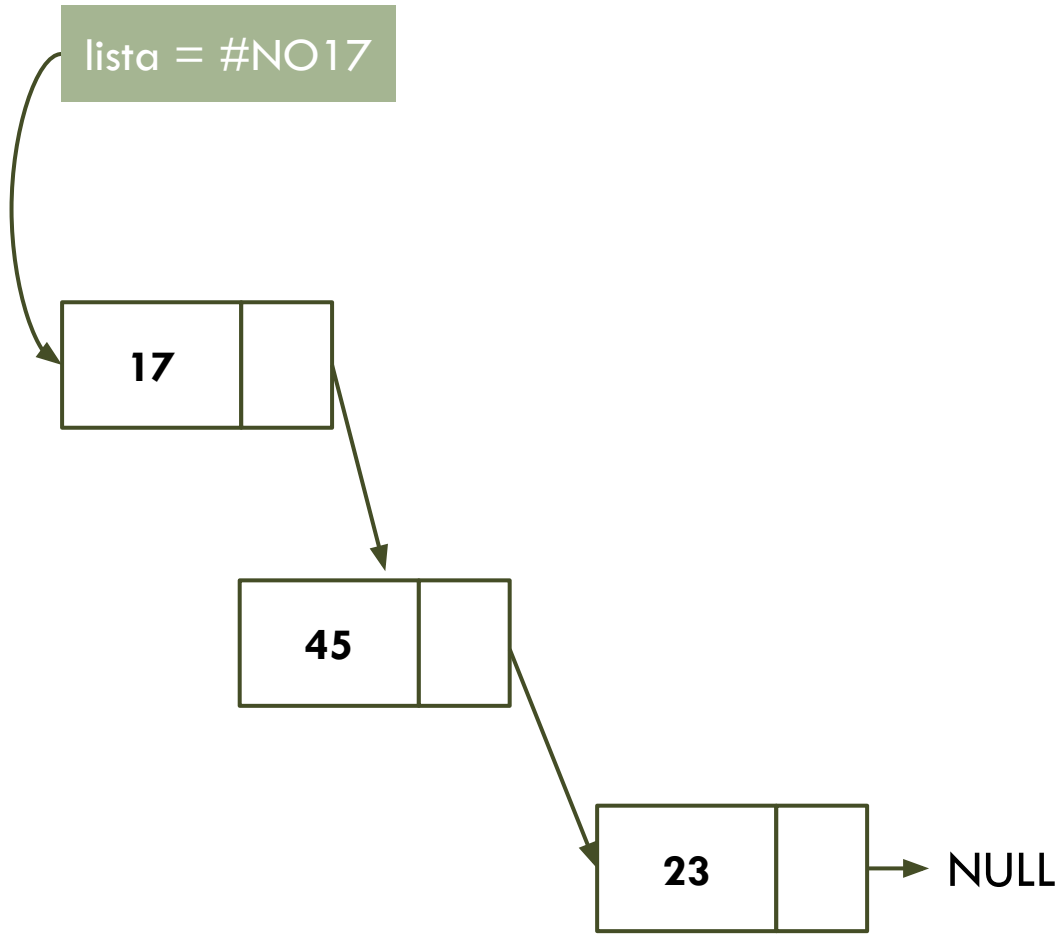
```
imprime(lista);
```

```
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**

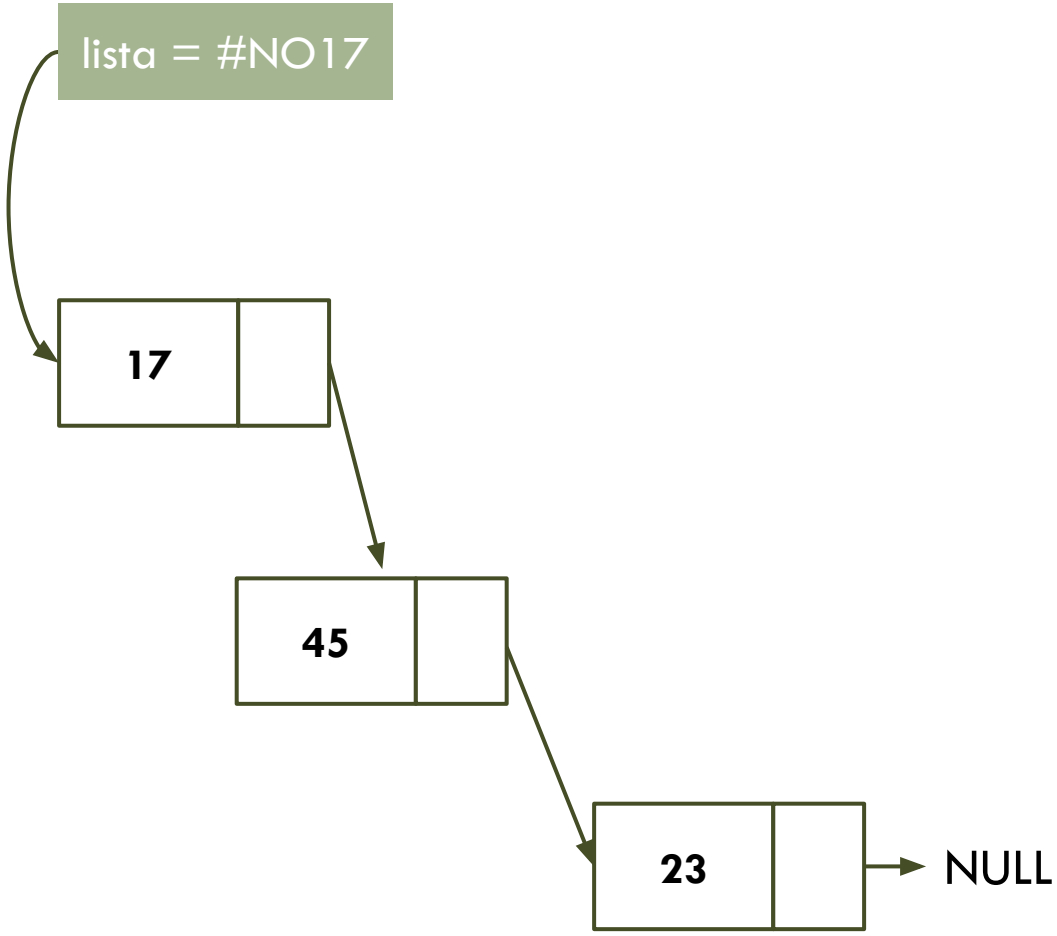


main()

lista = retira(lista, 17);

Lista encadeada simples

Estudo de caso - remove elemento do **início**



main()

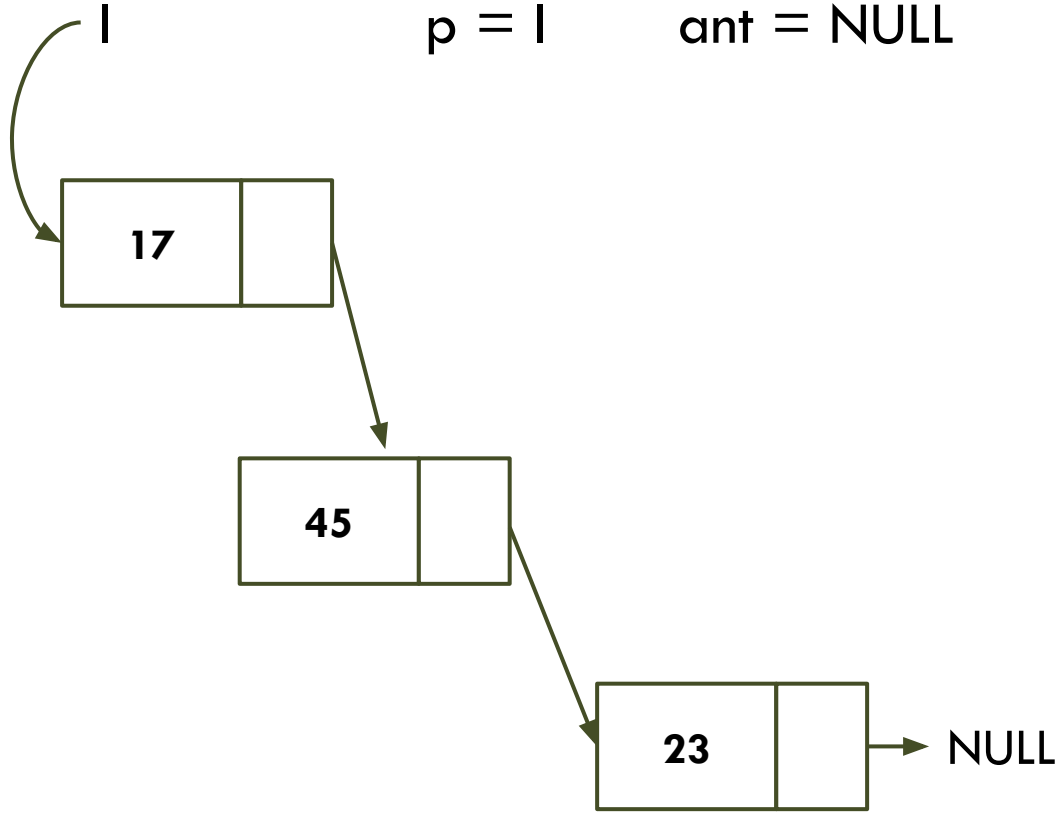
lista = retira(lista, 17);

```
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**

lista = #NO17



main()

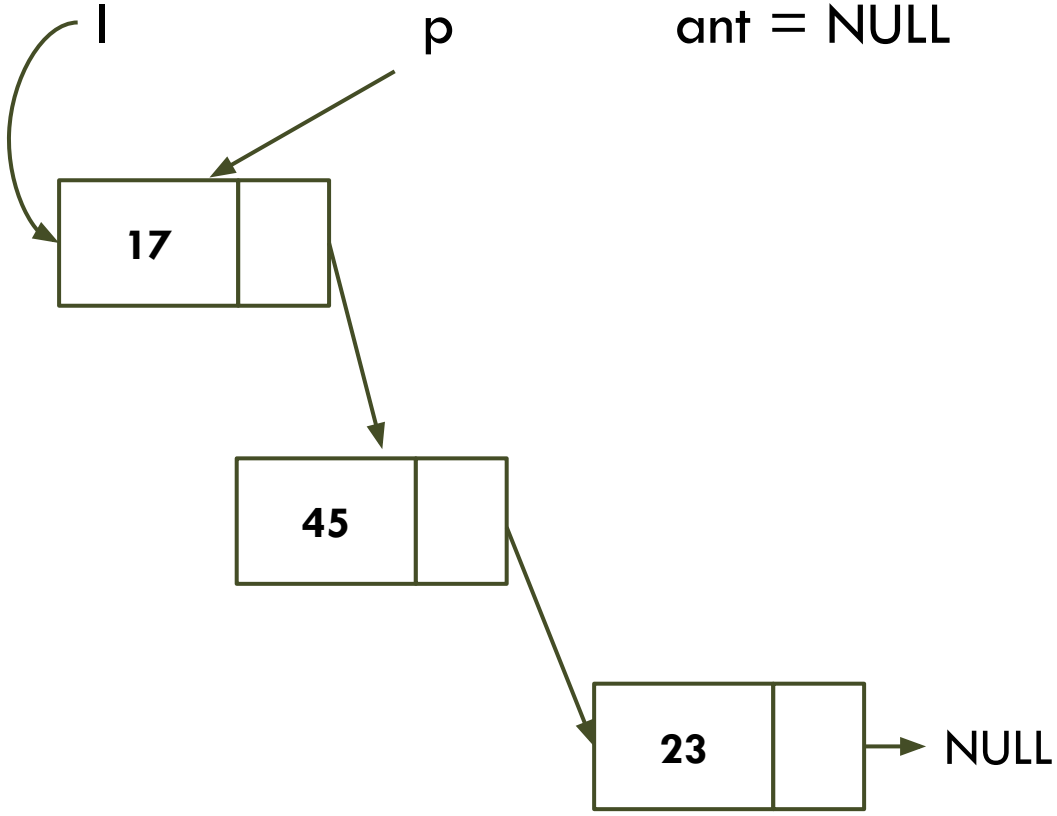
lista = retira(lista, 17);

```
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**

lista = #NO17



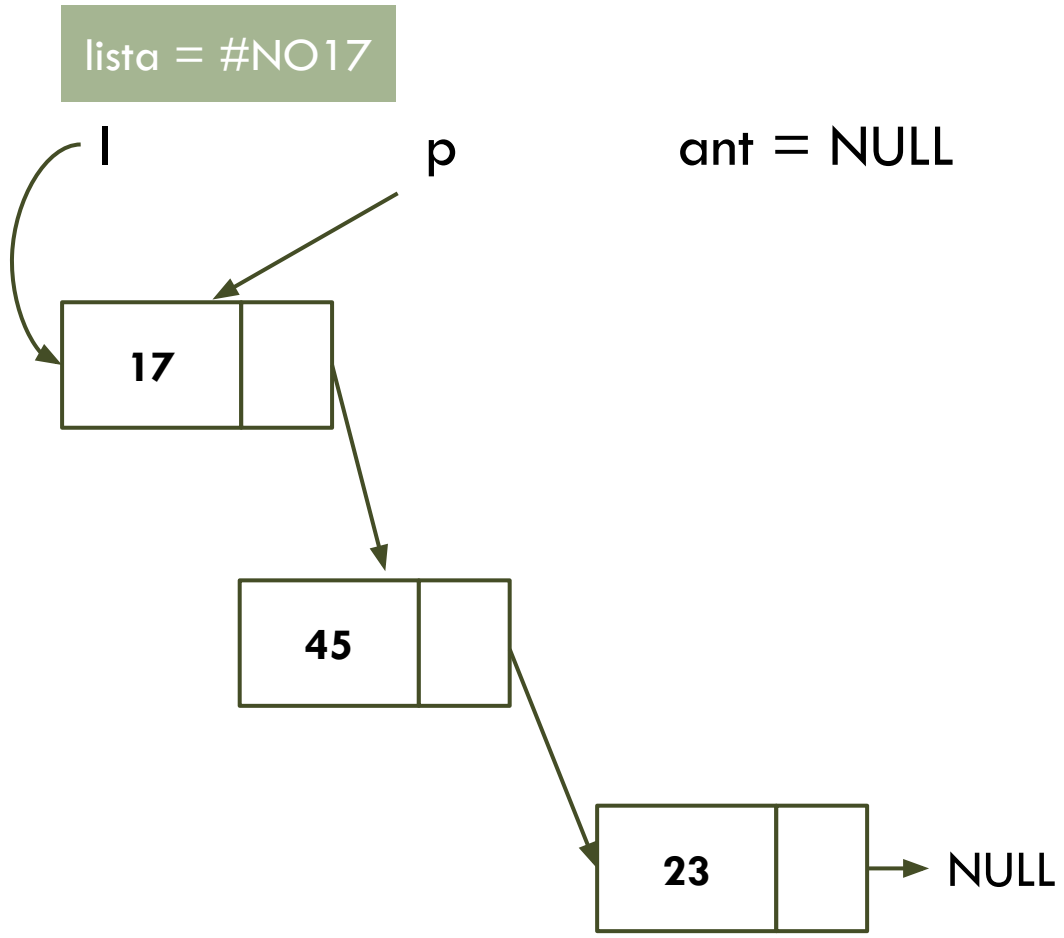
main()

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**



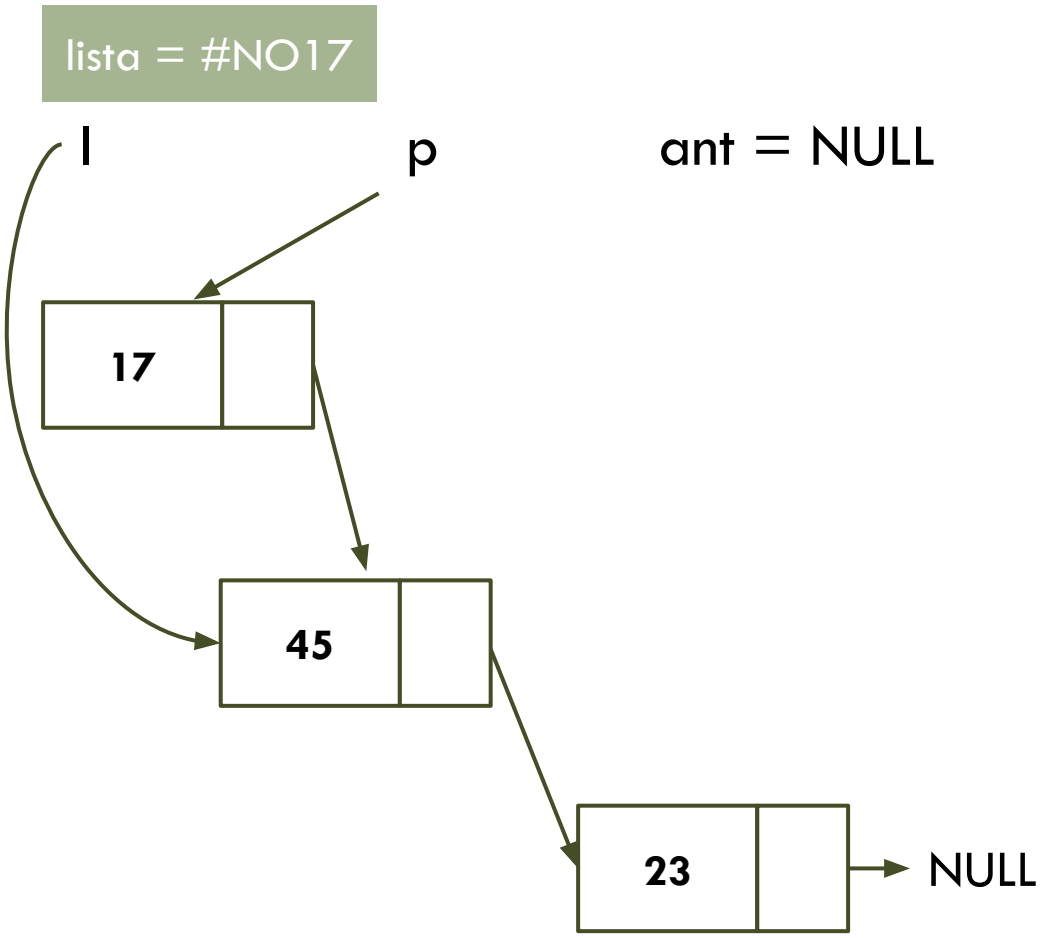
main()

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**



main()

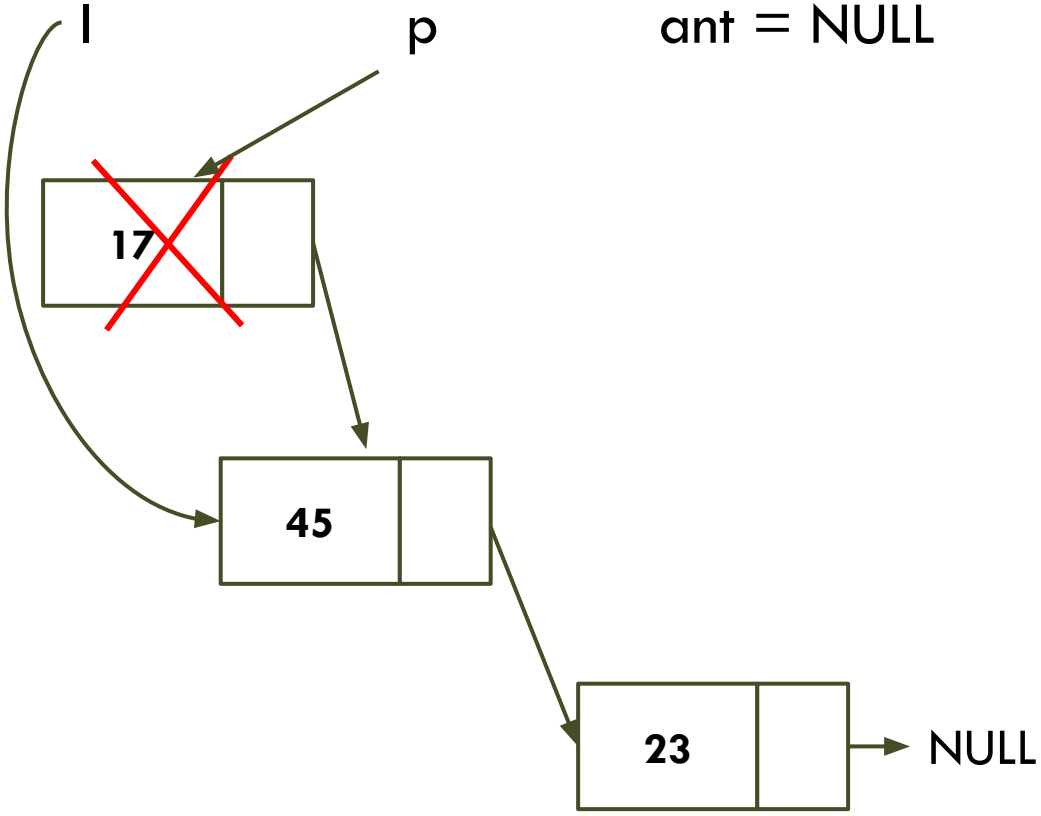
lista = retira(lista, 17);

```
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**

lista = #NO17



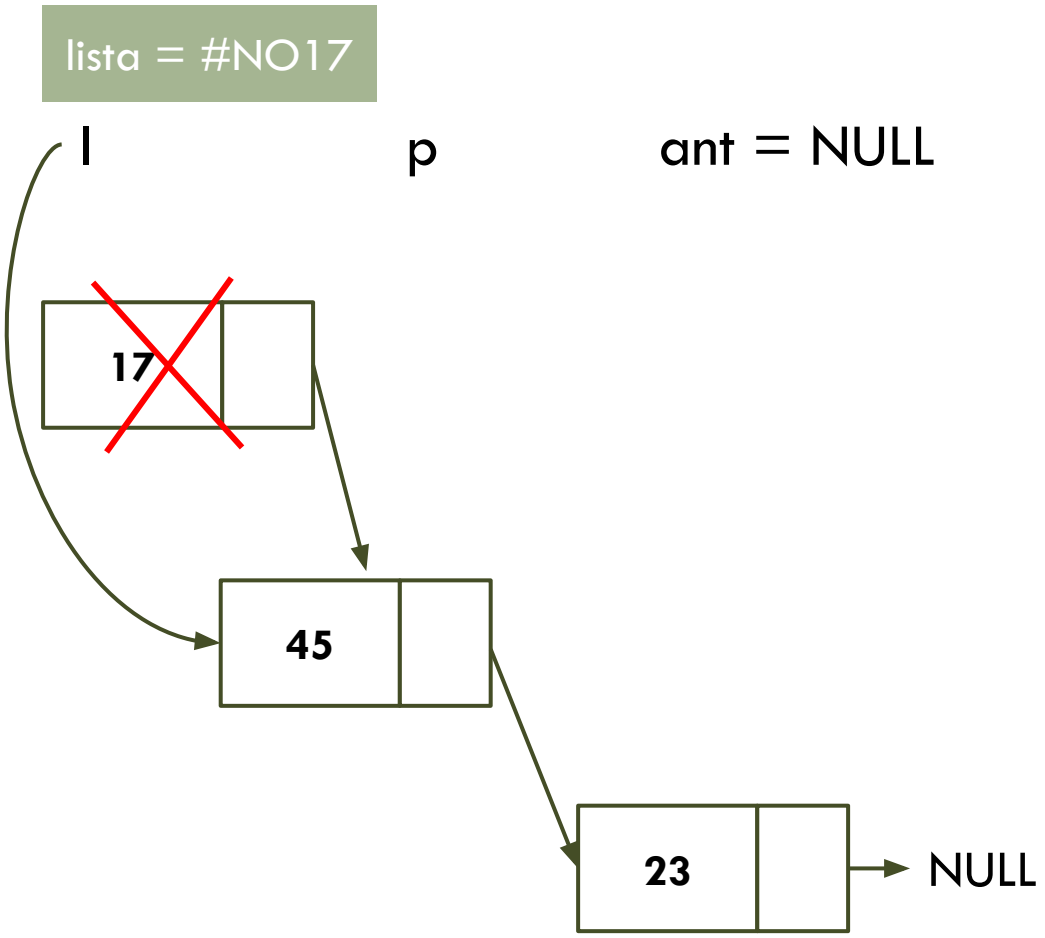
main()

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**



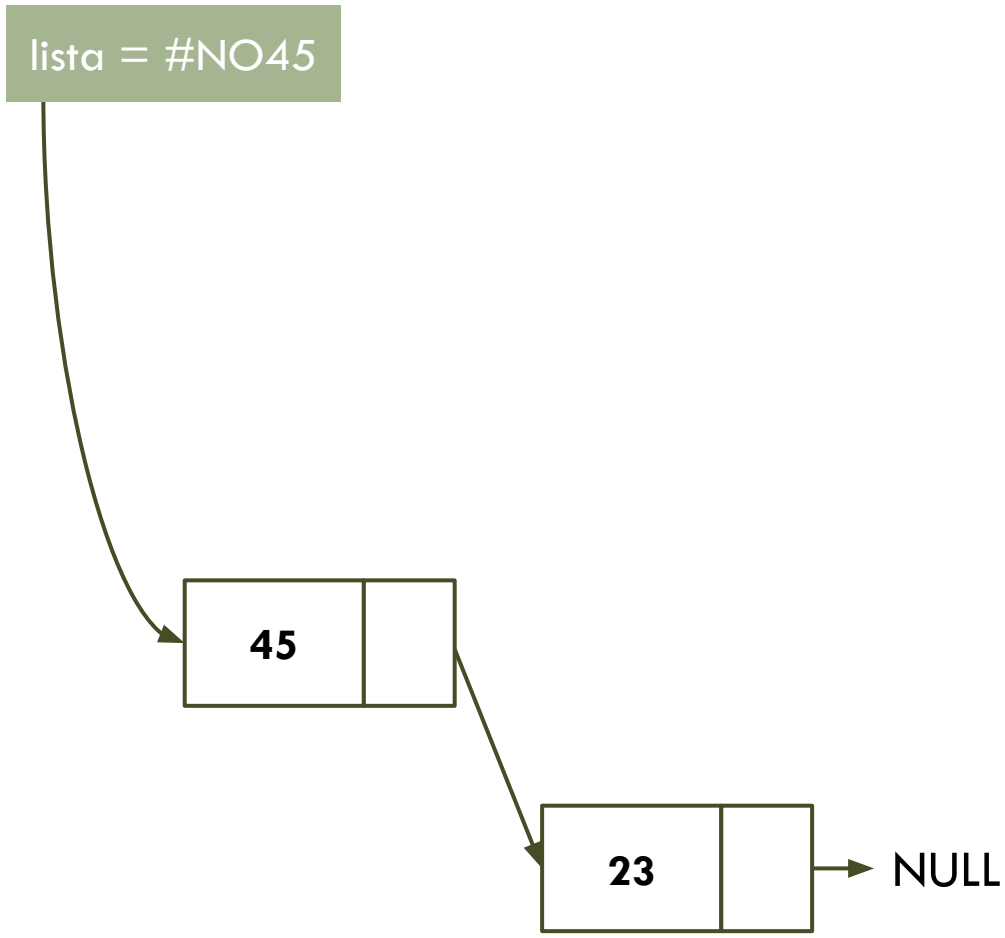
main()

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**

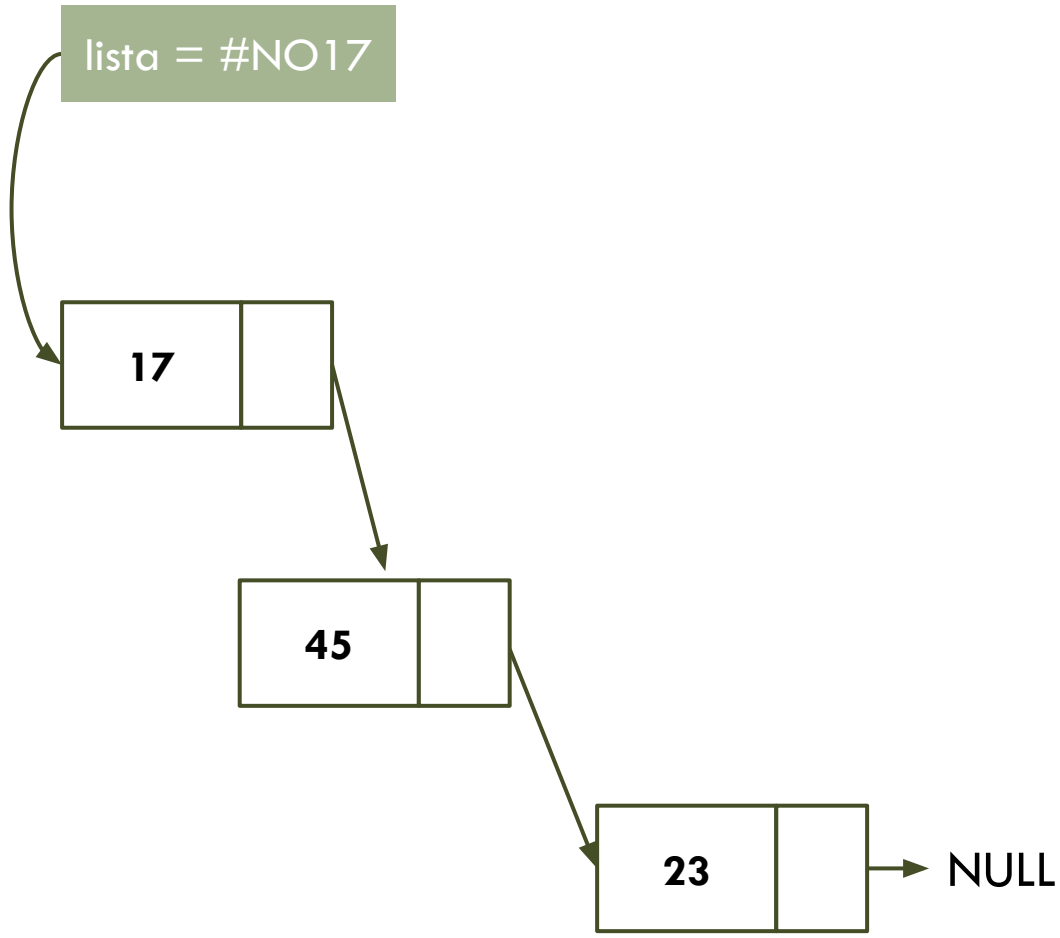


main()

lista = retira(lista, 17);

Lista encadeada simples

Estudo de caso - remove elemento do **meio**

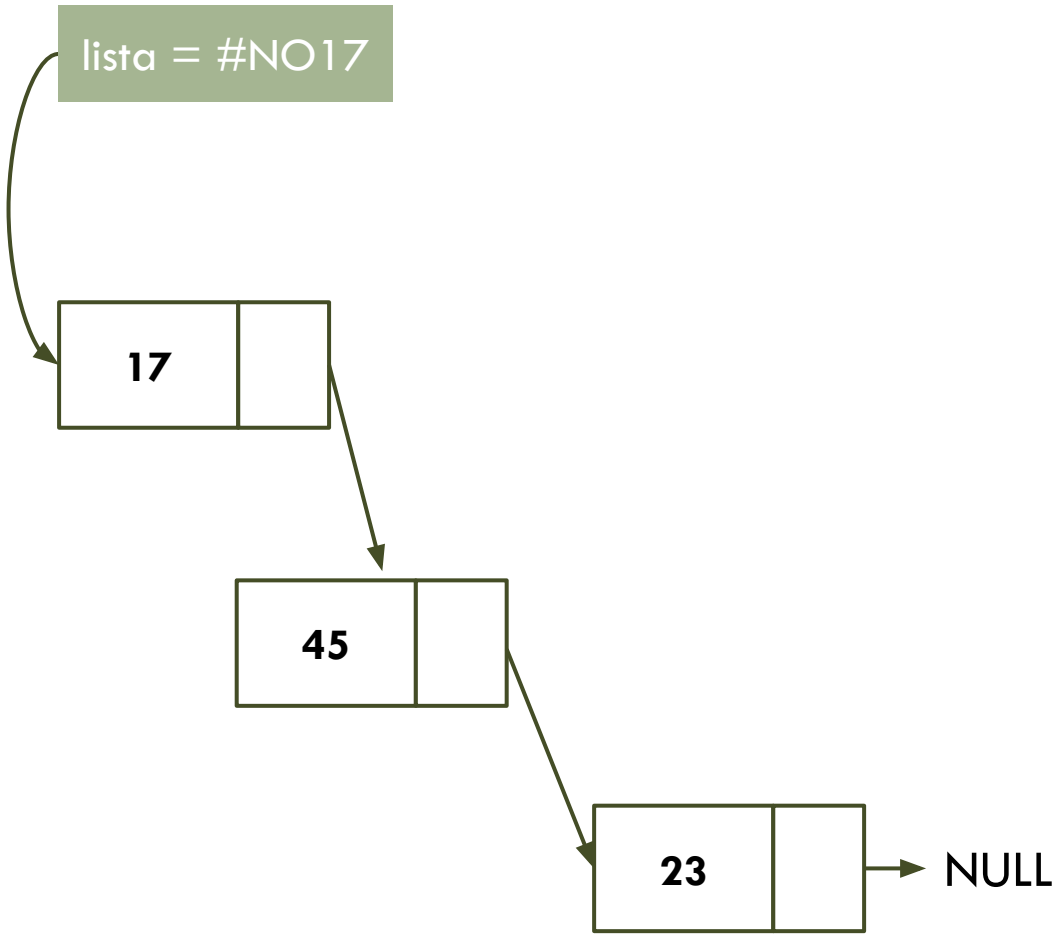


main()

lista = retira(lista, 45);

Lista encadeada simples

Estudo de caso - remove elemento do **meio**



main()

lista = retira(lista, 45);

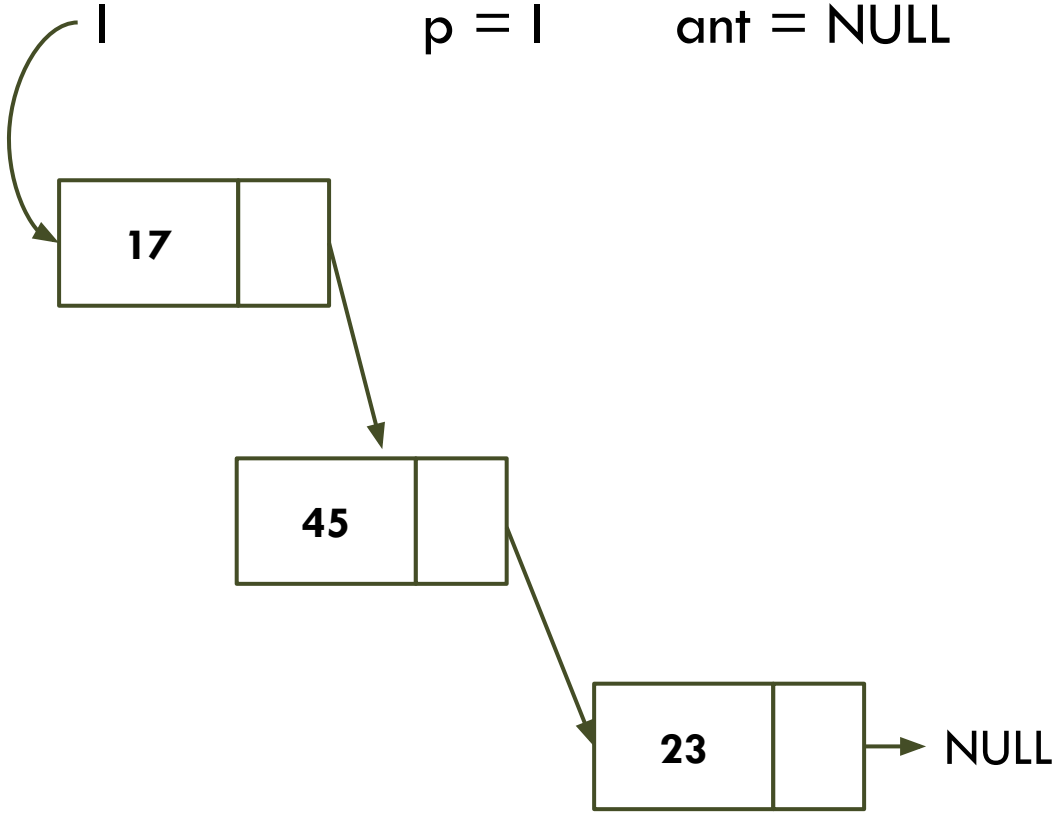
```
Lista* retira (Lista* l, int v){
    Lista* ant = NULL;
    Lista* p = l;
    for (p=l; p!=NULL; p = p->prox) {
        if (p->info == v){
            break;
        }
        ant = p;
    }
    if (p == NULL)
        return l;
    if (ant == NULL){
        l = p->prox;
    }else {
        ant->prox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

Lista encadeada simples

Estudo de caso - remove elemento do **meio**

lista = #NO17

l p = l ant = NULL



main()

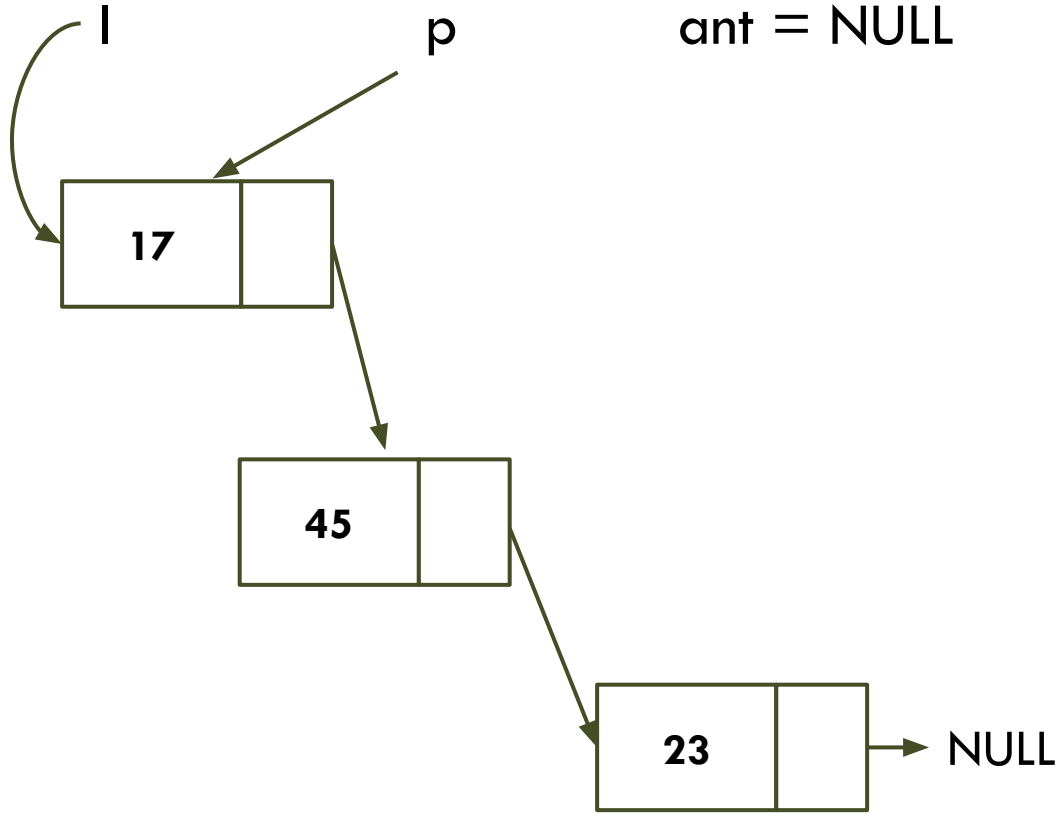
lista = retira(lista, 45);

```
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**

lista = #NO17



main()

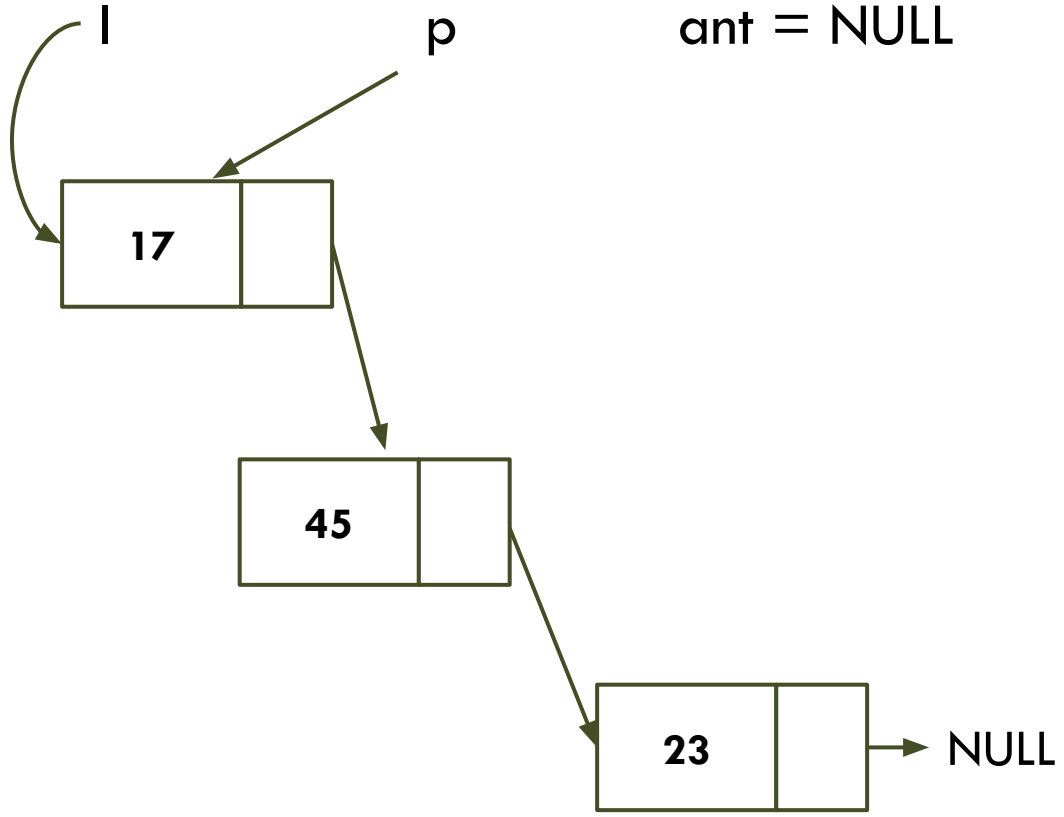
lista = retira(lista, 45);

```
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**

lista = #NO17



main()

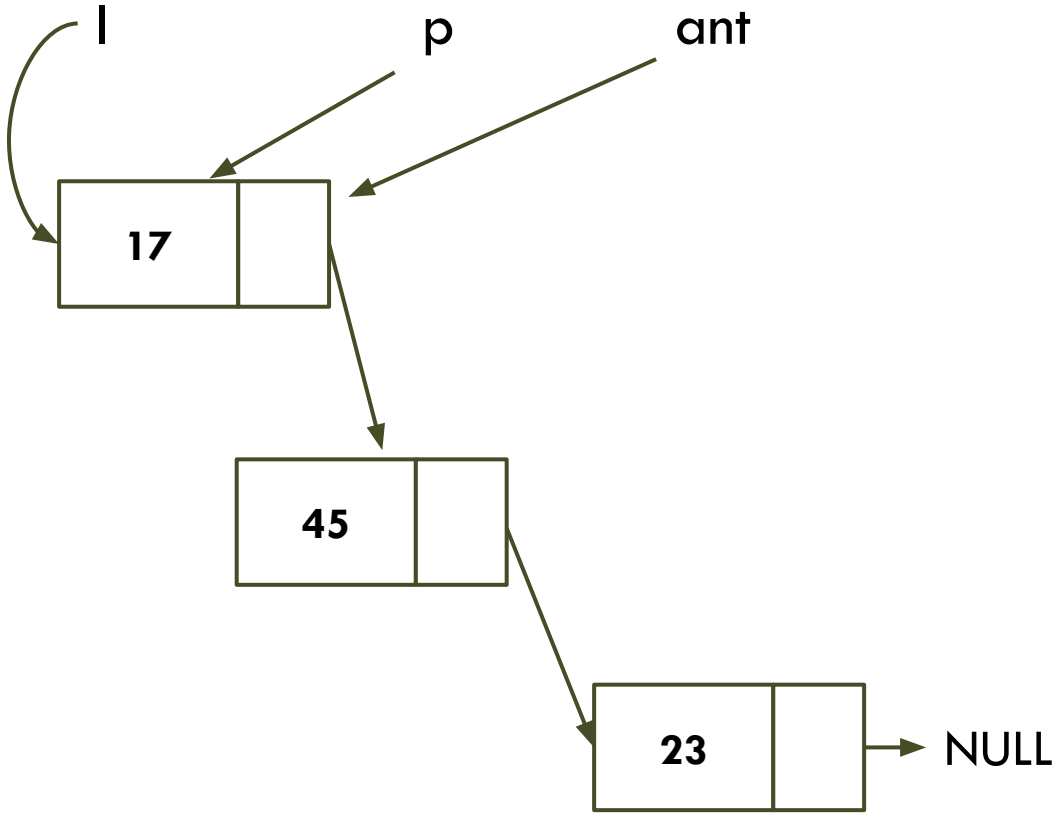
lista = retira(lista, 45);

```
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**

lista = #NO17



main()

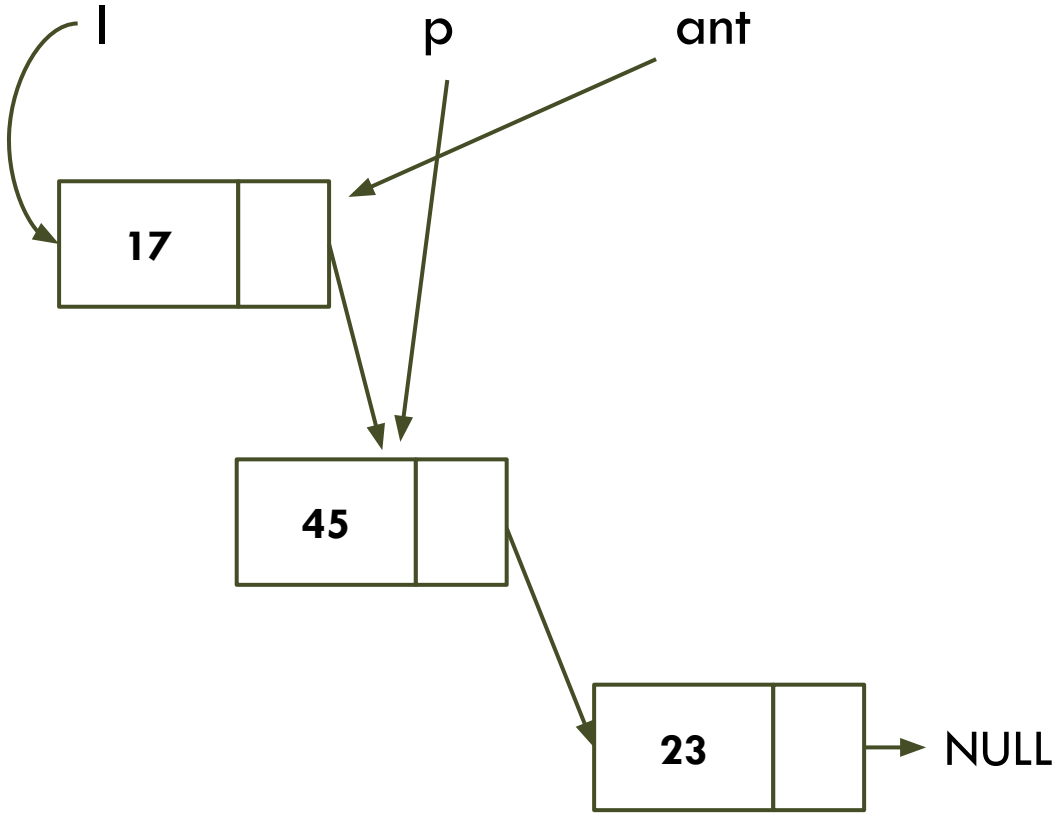
lista = retira(lista, 45);

```
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**

lista = #NO17



main()

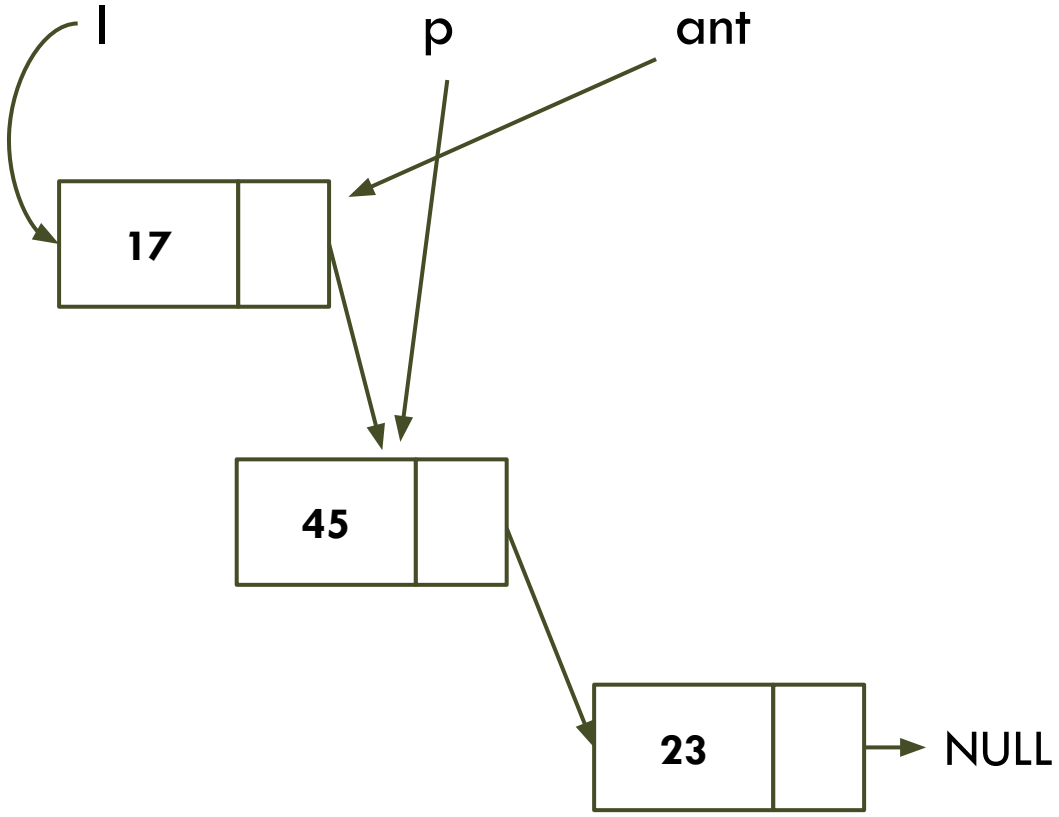
lista = retira(lista, 45);

```
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**

lista = #NO17



main()

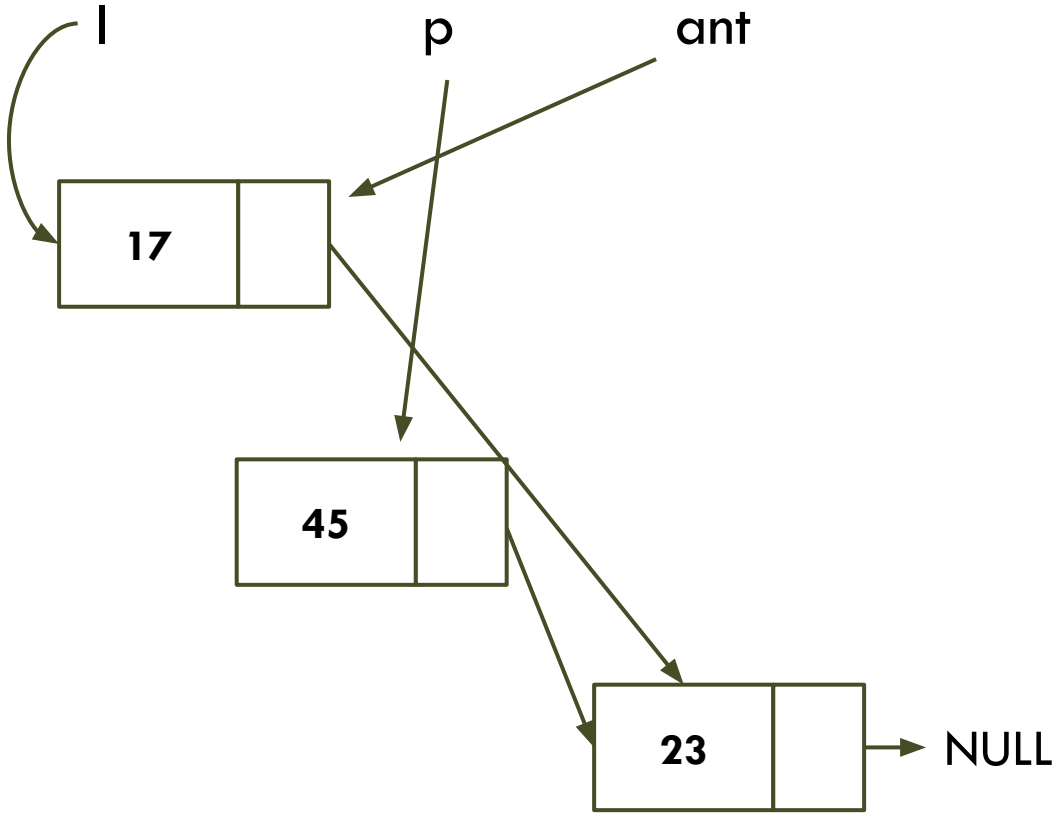
lista = retira(lista, 45);

```
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**

lista = #NO17



main()

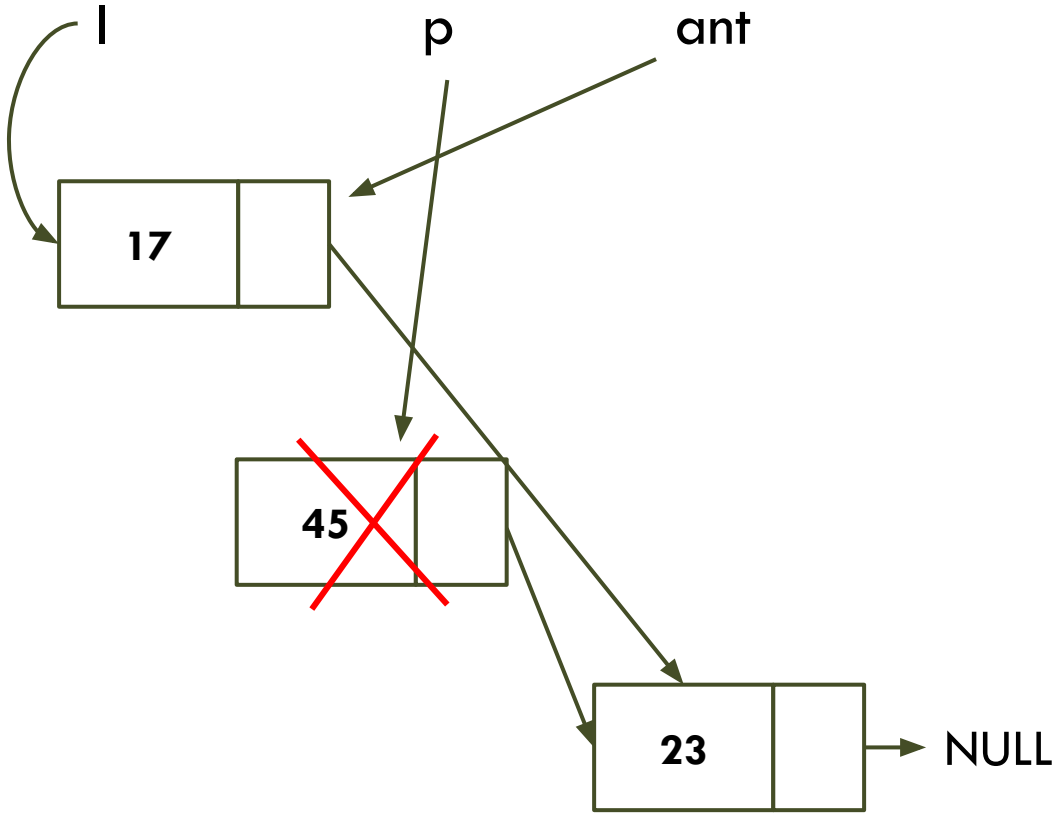
lista = retira(lista, 45);

```
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**

lista = #NO17



main()

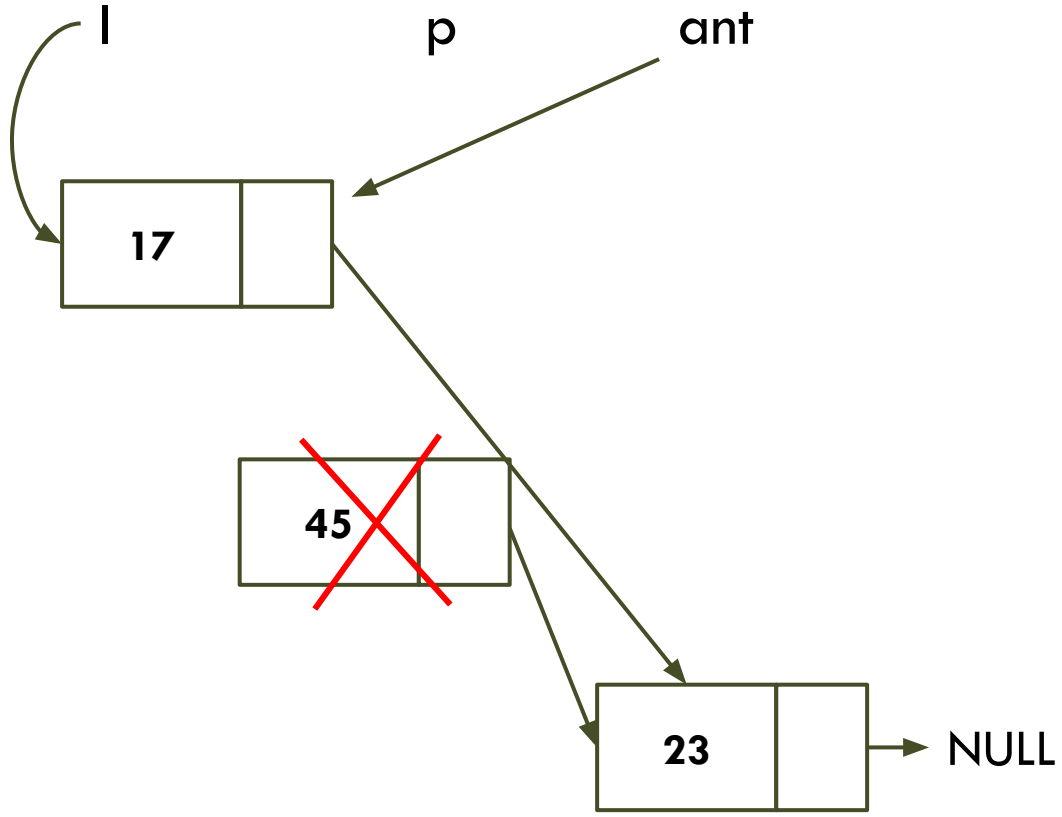
lista = retira(lista, 45);

```
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**

lista = #NO17



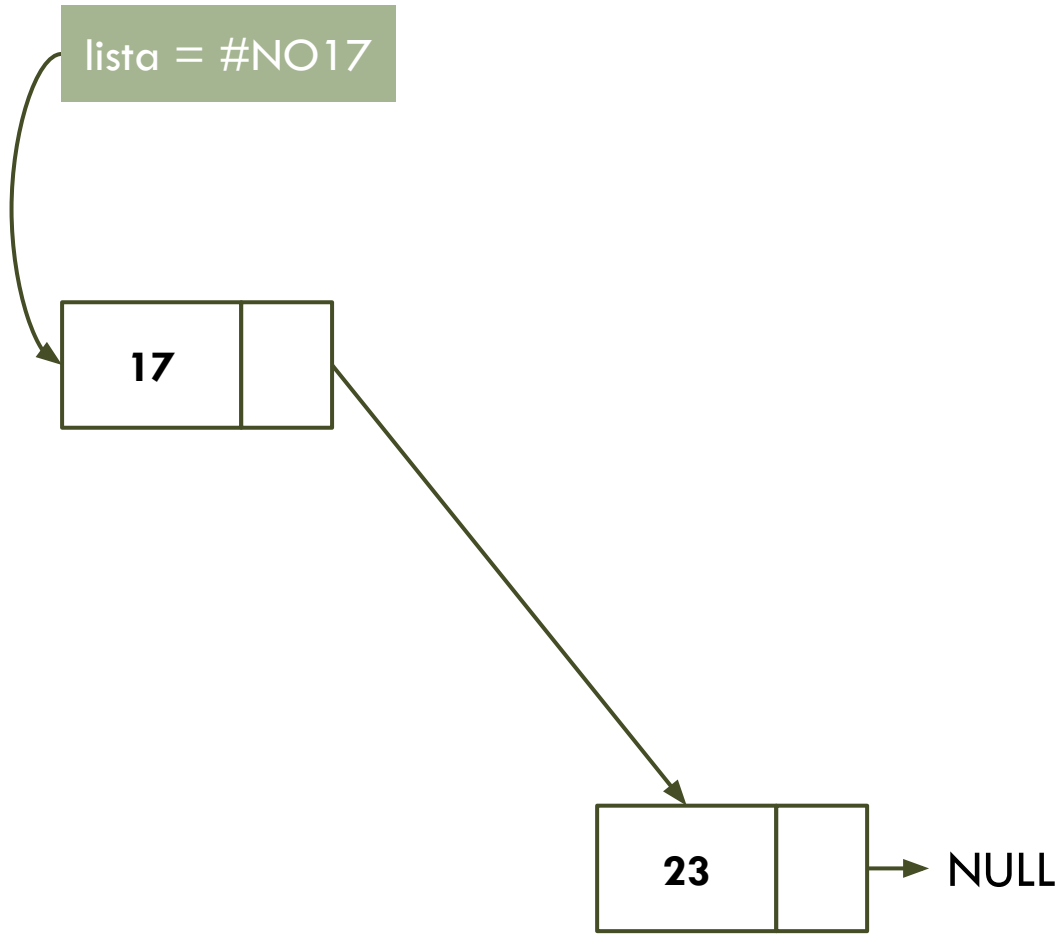
main()

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**

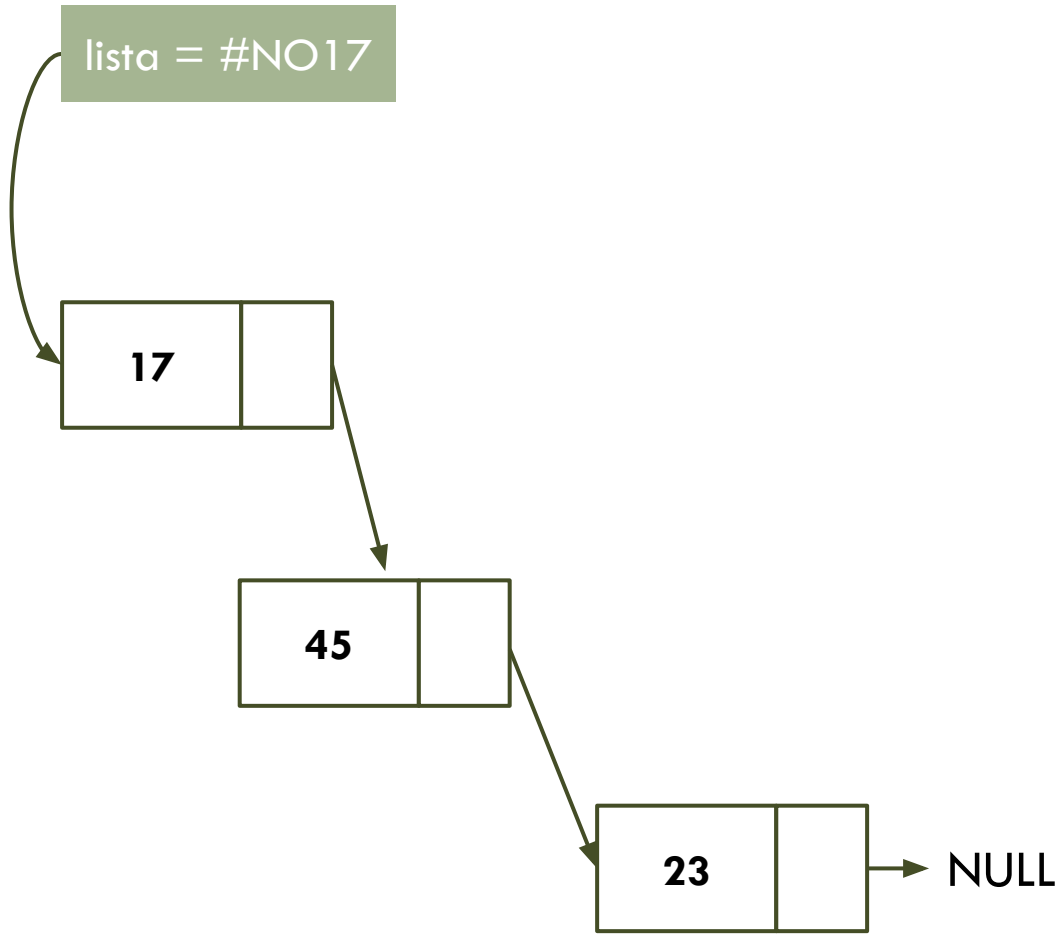


main()

lista = retira(lista, 45);

Lista encadeada simples

Estudo de caso - remove elemento do **fim**

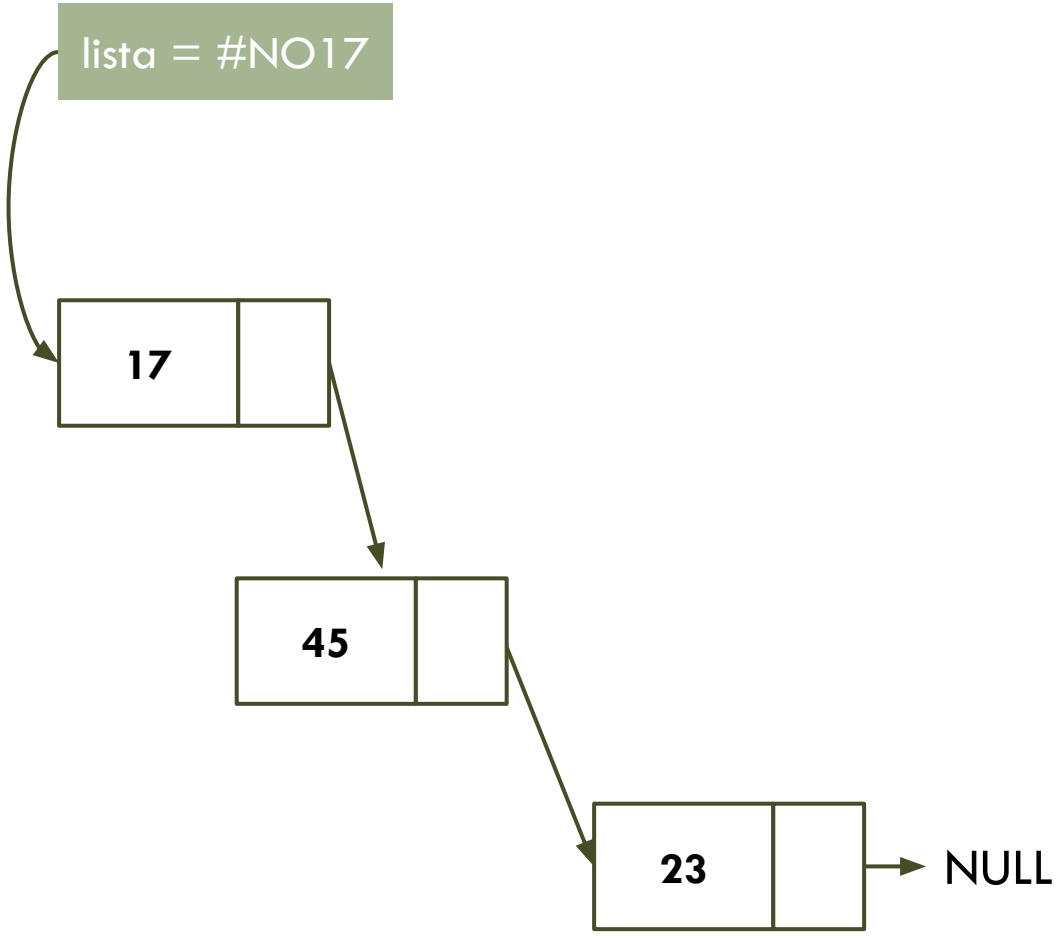


main()

lista = retira(lista, 23);

Lista encadeada simples

Estudo de caso - remove elemento do **fim**



main()

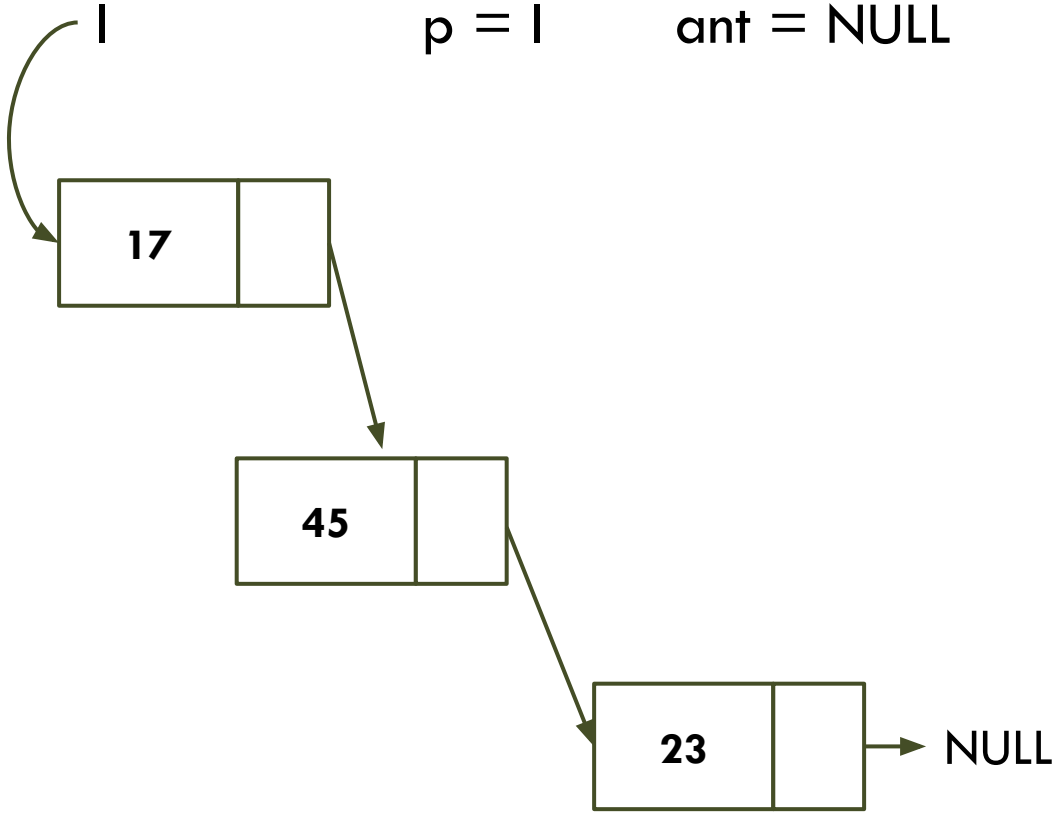
lista = retira(lista, 23);

```
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**

lista = #NO17



main()

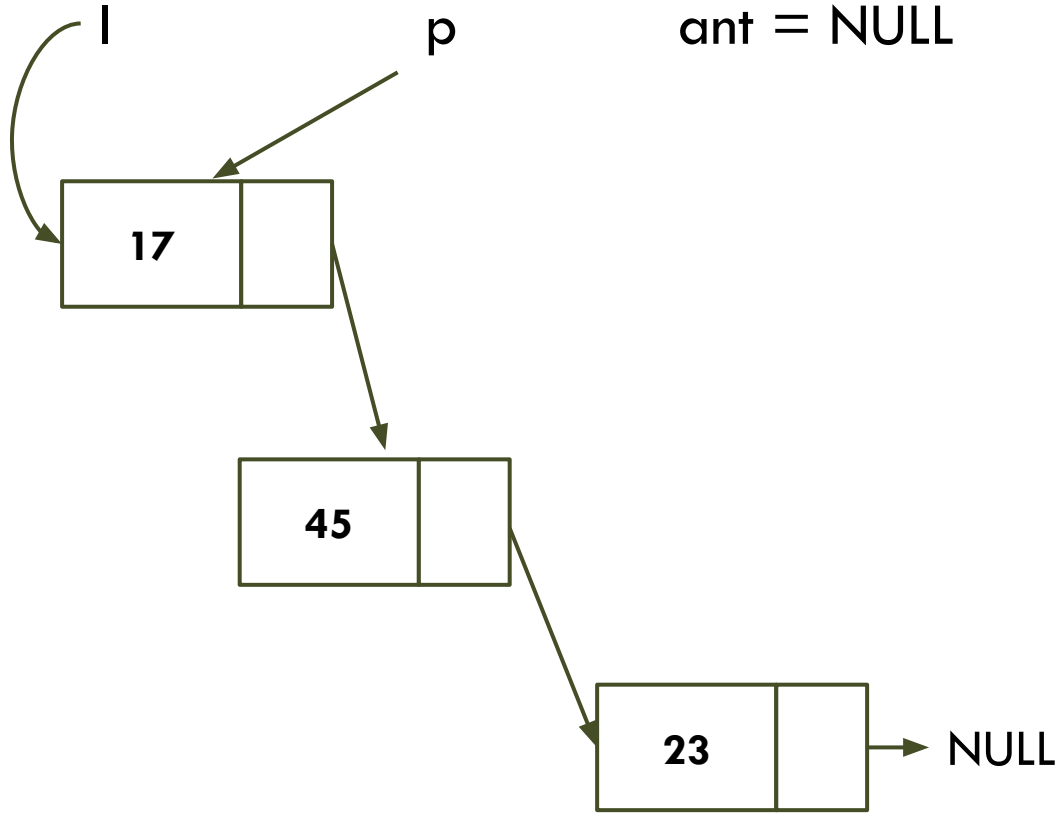
lista = retira(lista, 23);

```
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**

lista = #NO17



main()

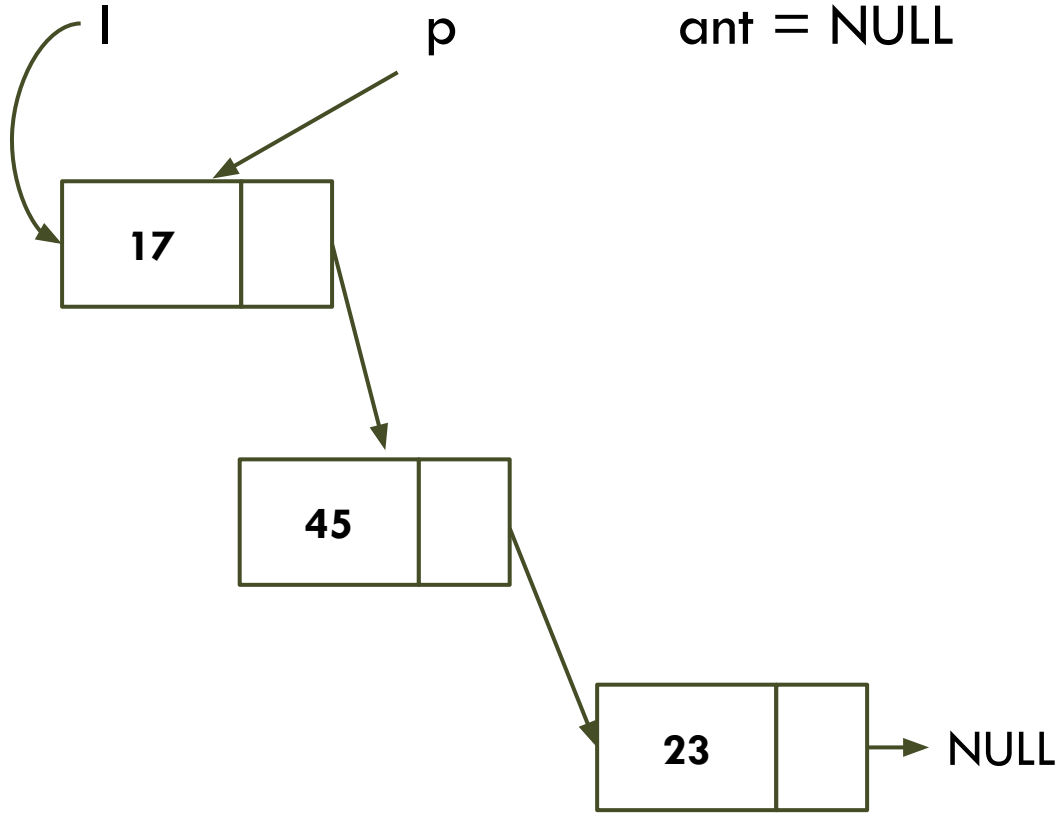
lista = retira(lista, 23);

```
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**

lista = #NO17



main()

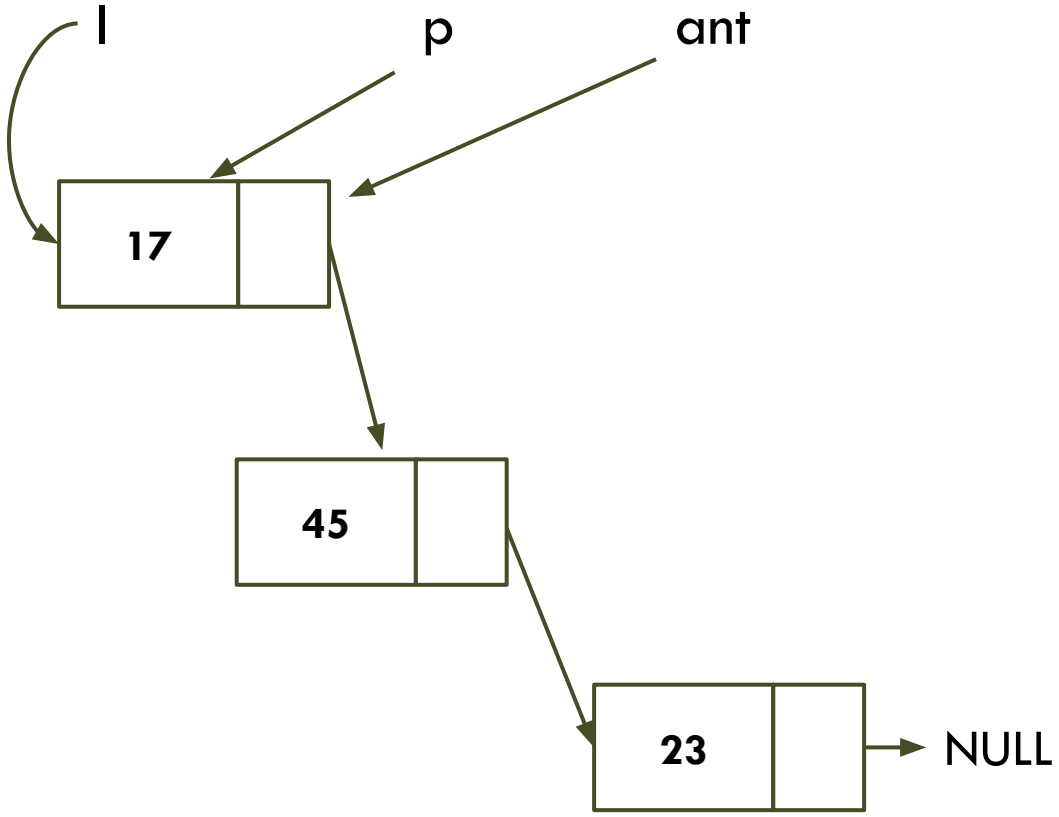
lista = retira(lista, 23);

```
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**

lista = #NO17



main()

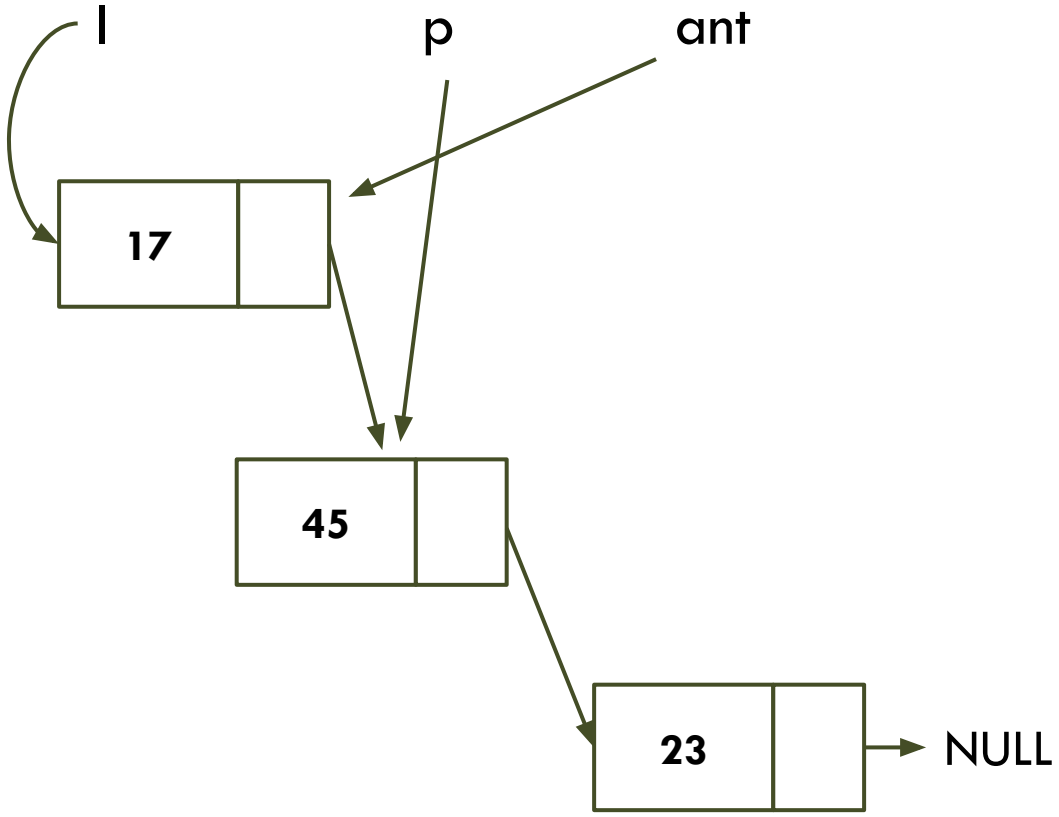
lista = retira(lista, 23);

```
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**

lista = #NO17



main()

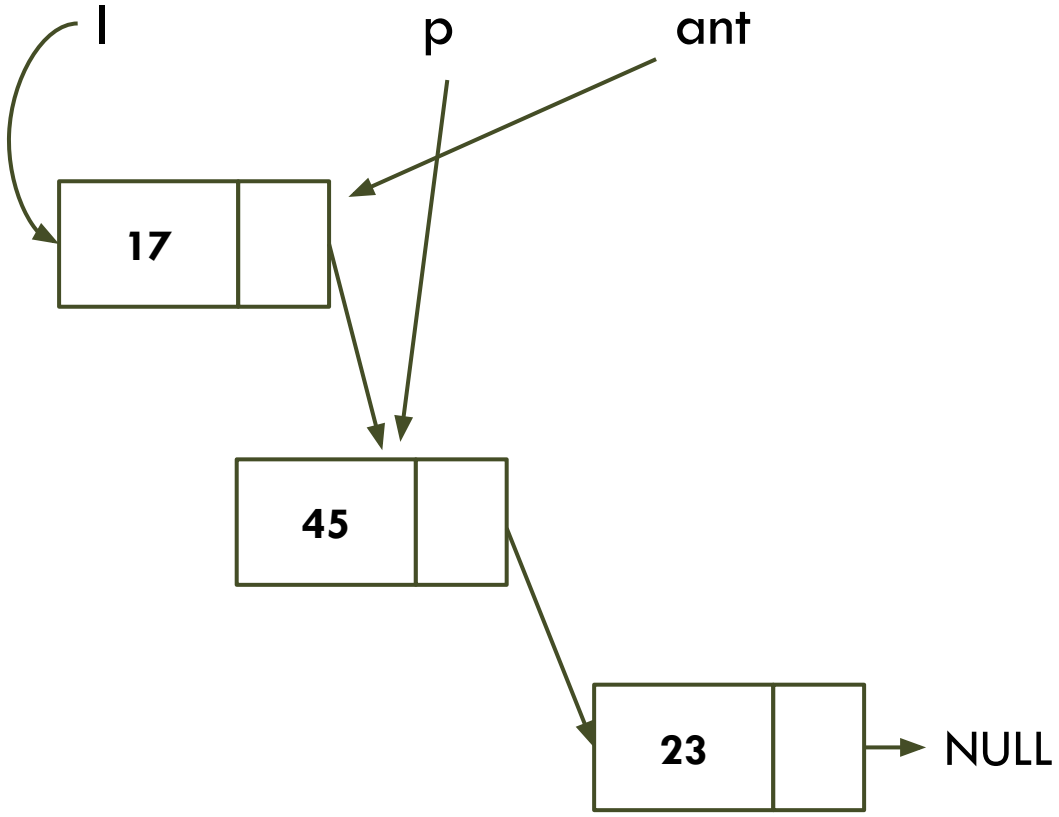
lista = retira(lista, 23);

```
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**

lista = #NO17



main()

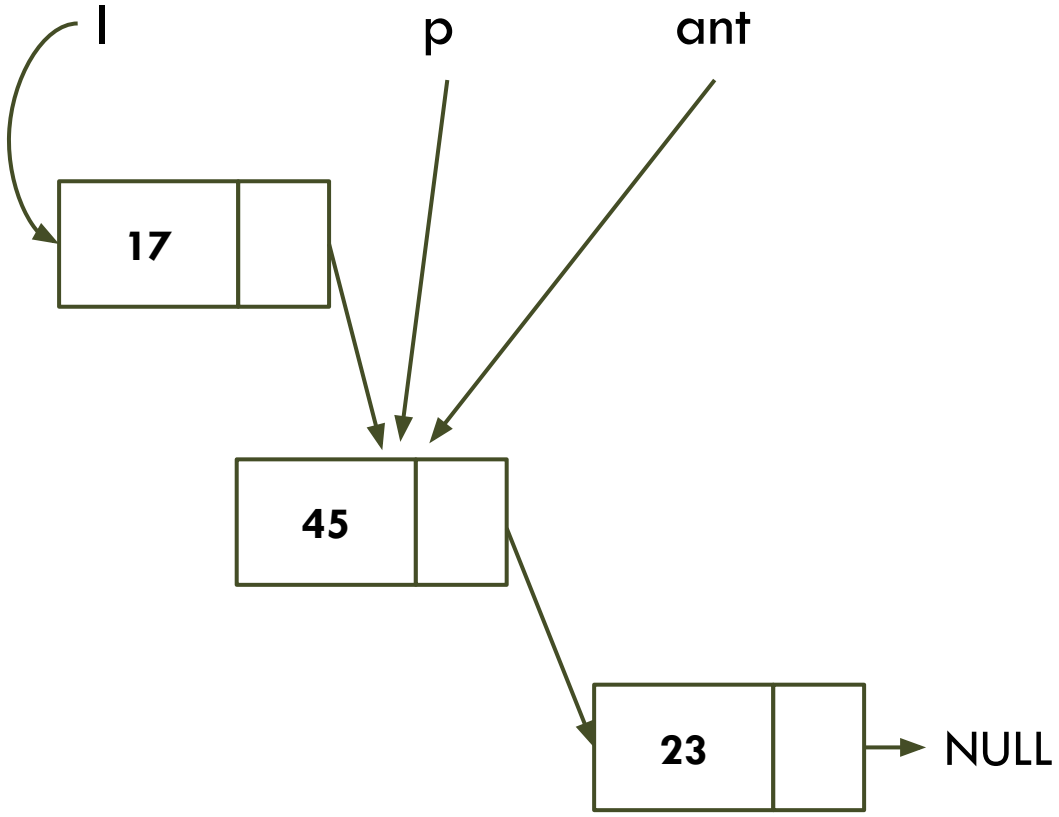
lista = retira(lista, 23);

```
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**

lista = #NO17



main()

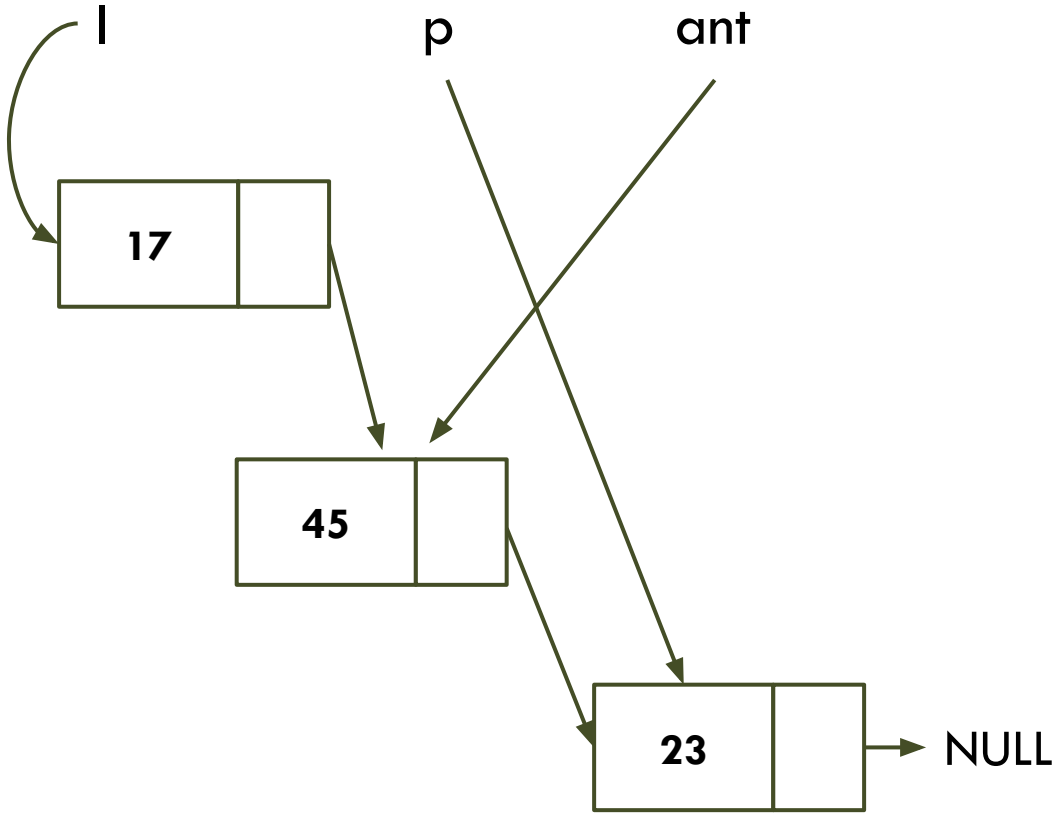
lista = retira(lista, 23);

```
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**

lista = #NO17



main()

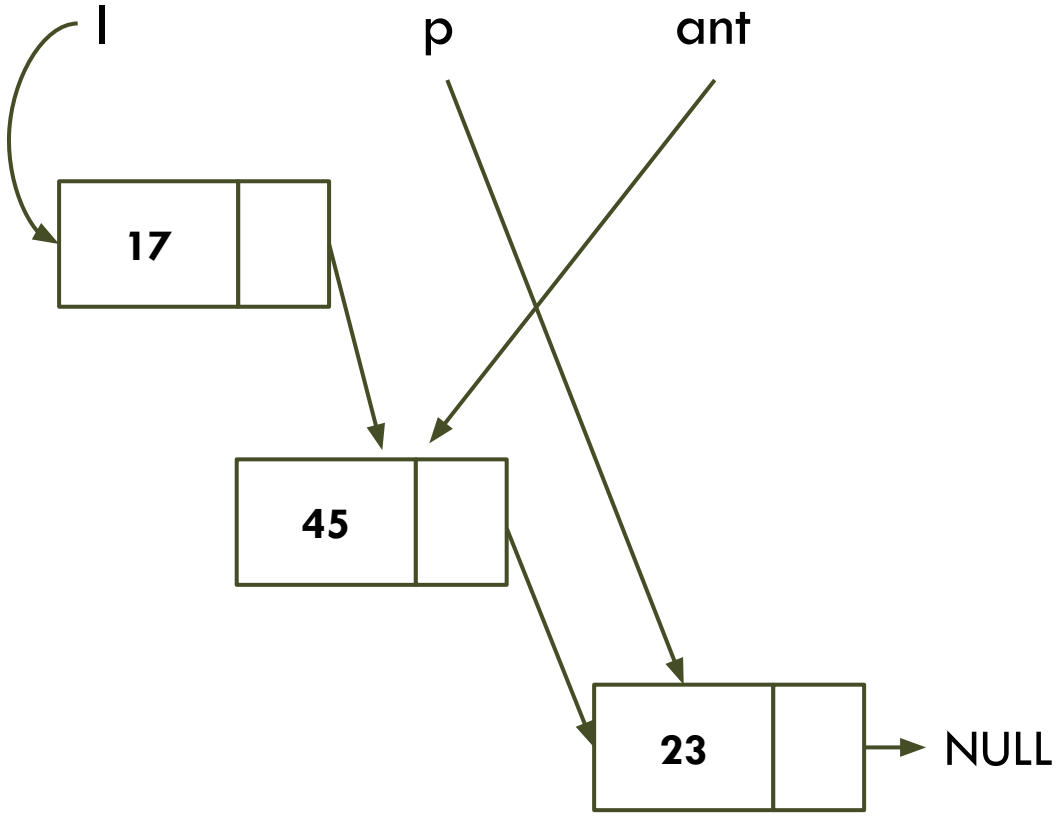
lista = retira(lista, 23);

```
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**

lista = #NO17



main()

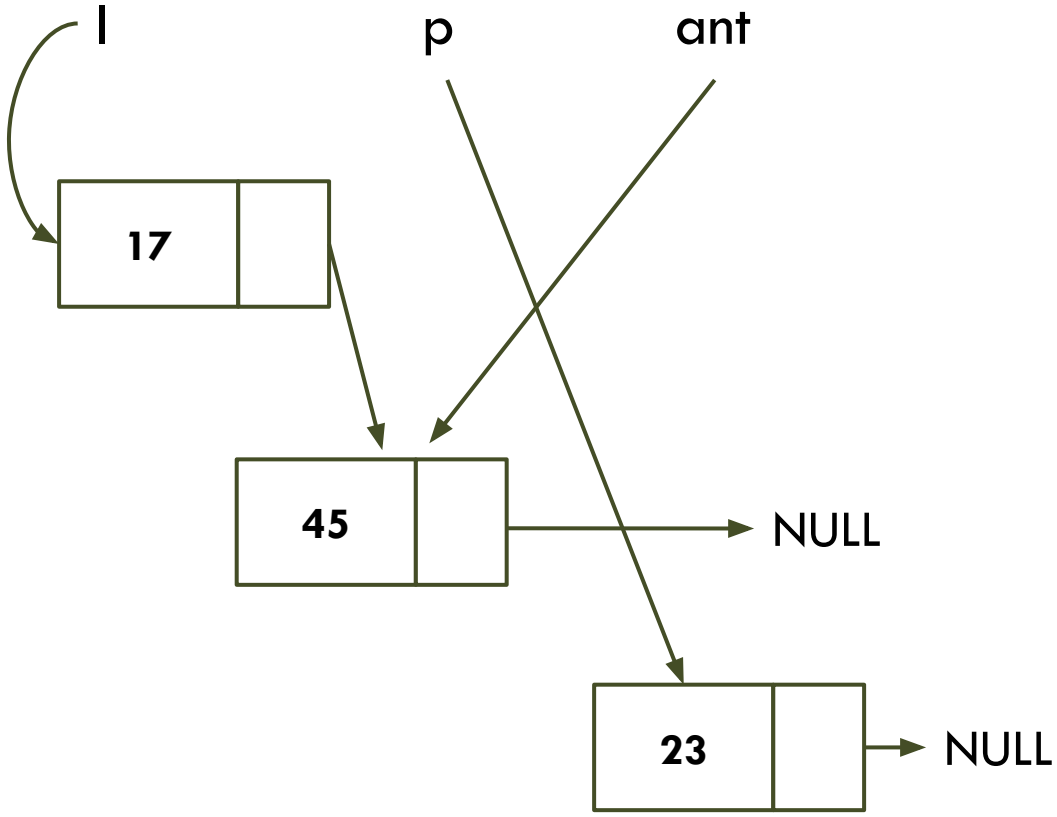
lista = retira(lista, 23);

```
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**

lista = #NO17



main()

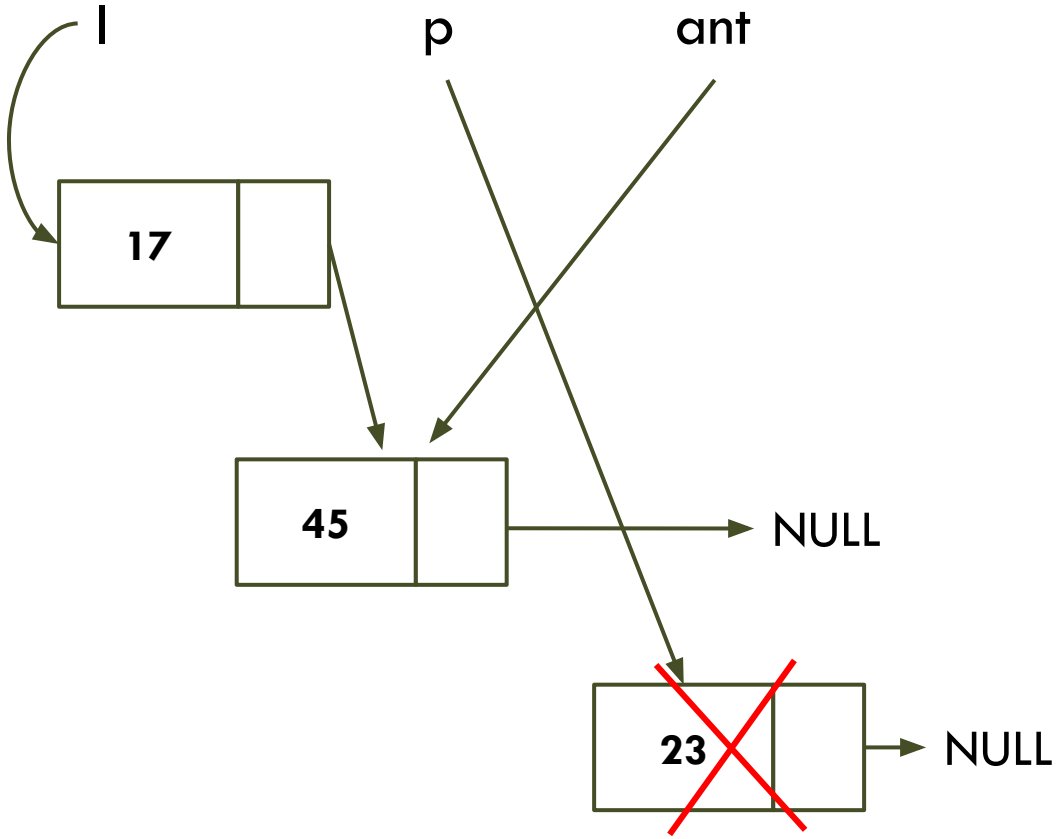
lista = retira(lista, 23);

```
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**

lista = #NO17



main()

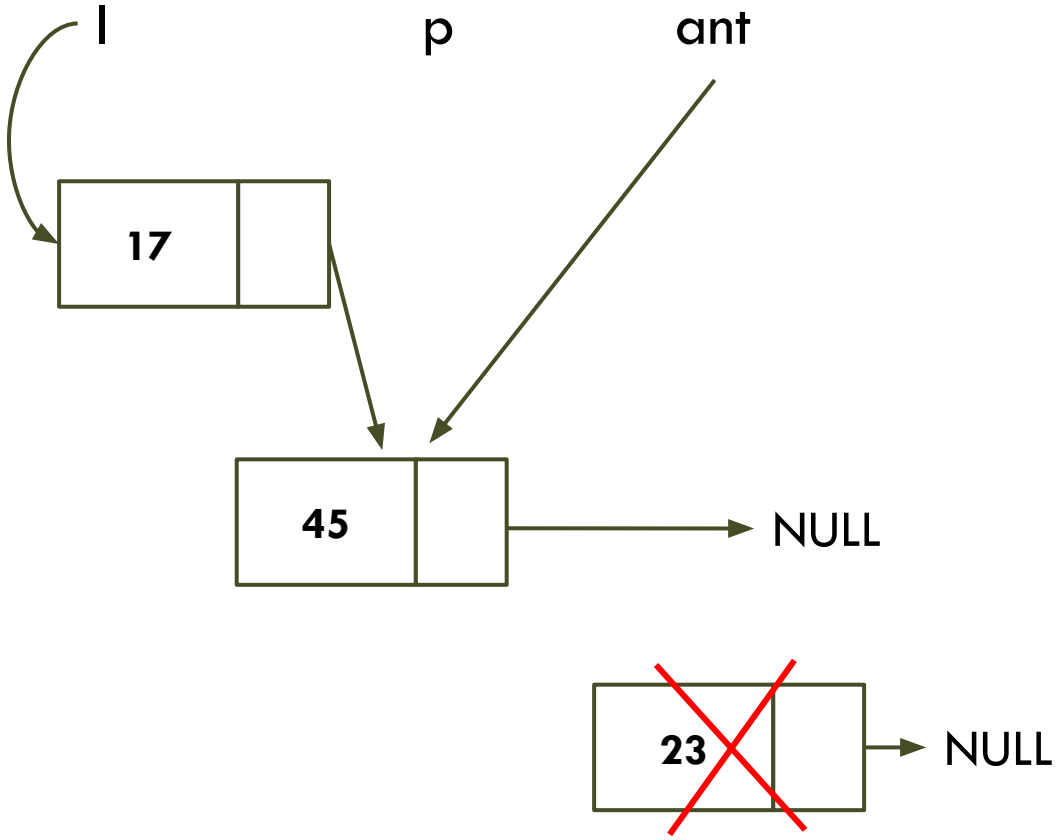
lista = retira(lista, 23);

```
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**

lista = #NO17



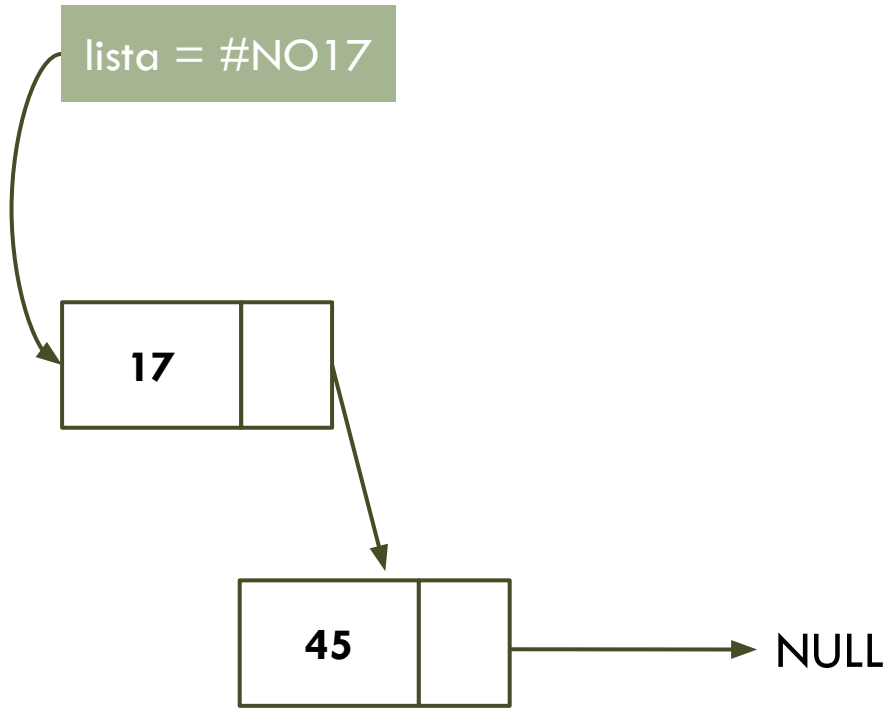
main()

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**



main()

lista = retira(lista, 23);

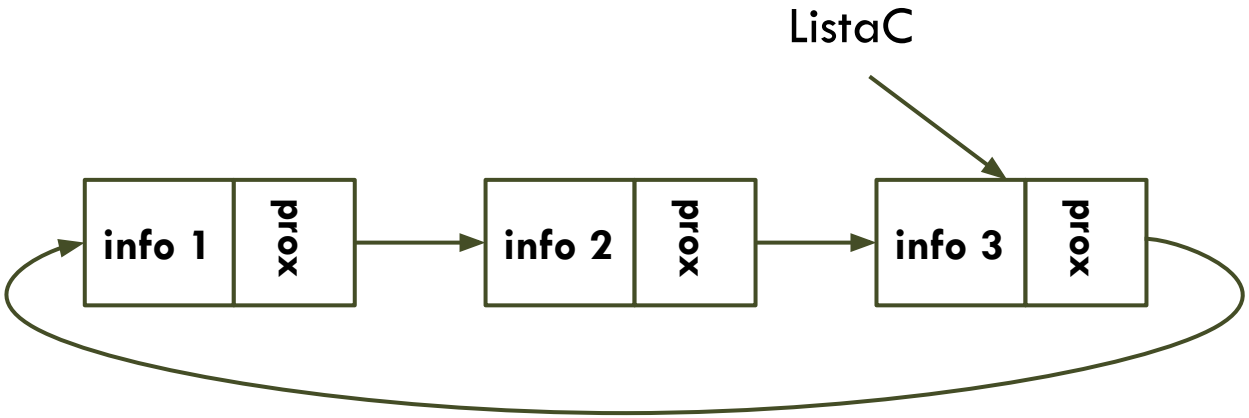
Lista encadeada circular

Lista encadeada circular

- 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

- 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

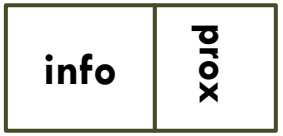


Lista encadeada circular

- **Exemplo**
  - Lista encadeada circular armazenando valores inteiros

```
typedef struct listaC ListaC;

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



Lista encadeada circular

- **Função inicializa**
  - Cria uma lista vazia, representada pelo ponteiro NULL

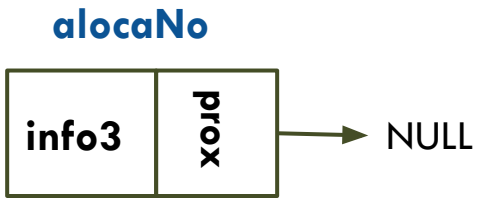
```
ListaC* inicializa (void){
        return NULL;
}
```

Lista encadeada circular

- **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;
}
```

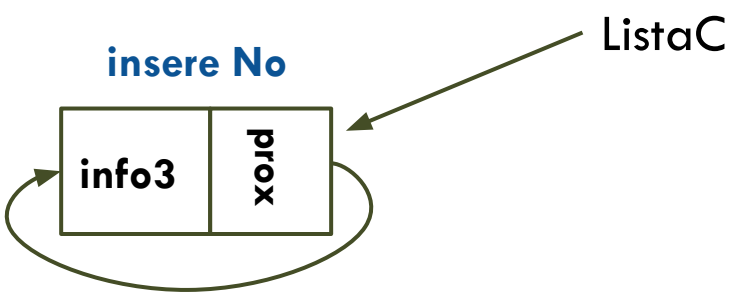


Lista encadeada circular

- **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;
}
```

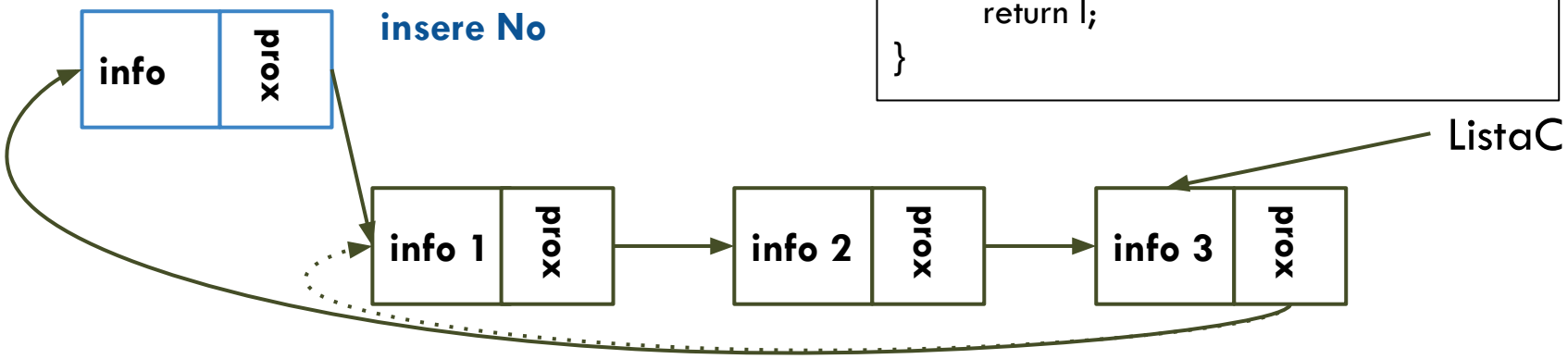


Lista encadeada circular

- **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;
}
```

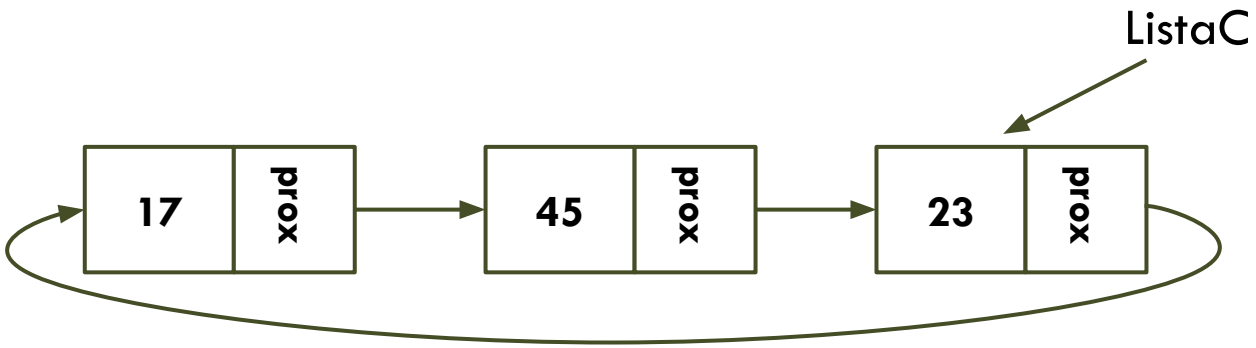
```
ListaC* insereCircular (ListaC* l, int valor){
        ListaC* novo = alocaNo(valor);
        if (l == NULL){
                novo->prox = novo;
                l = novo;
        }else{
                novo->prox = l->prox;
                l->prox = novo;
        }
        return l;
}
```



Lista encadeada circular

- **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);
    }
}
```

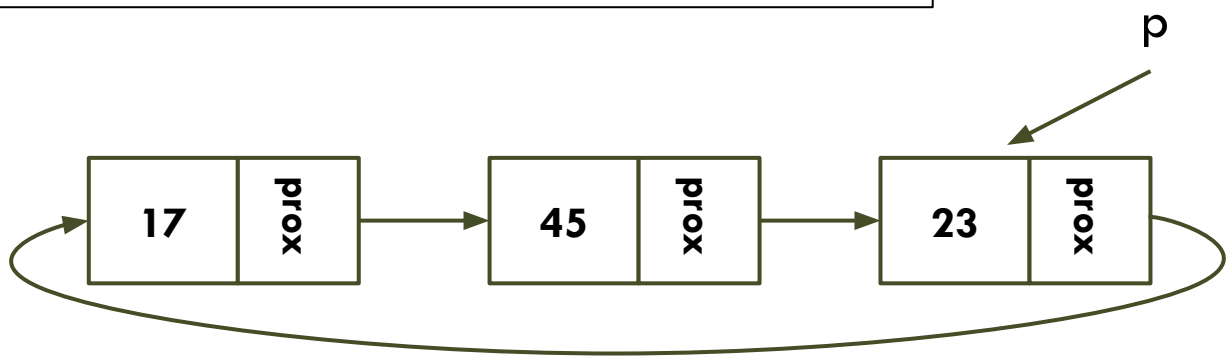


Lista encadeada circular

- **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:

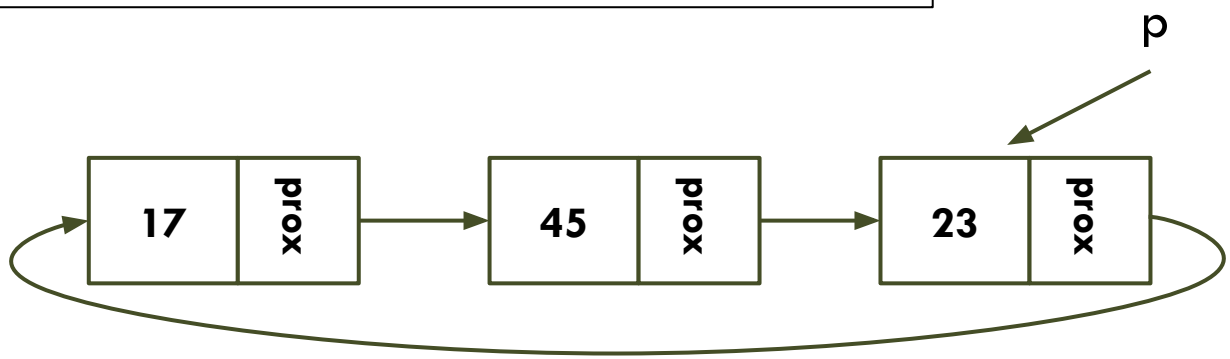


Lista encadeada circular

- **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:

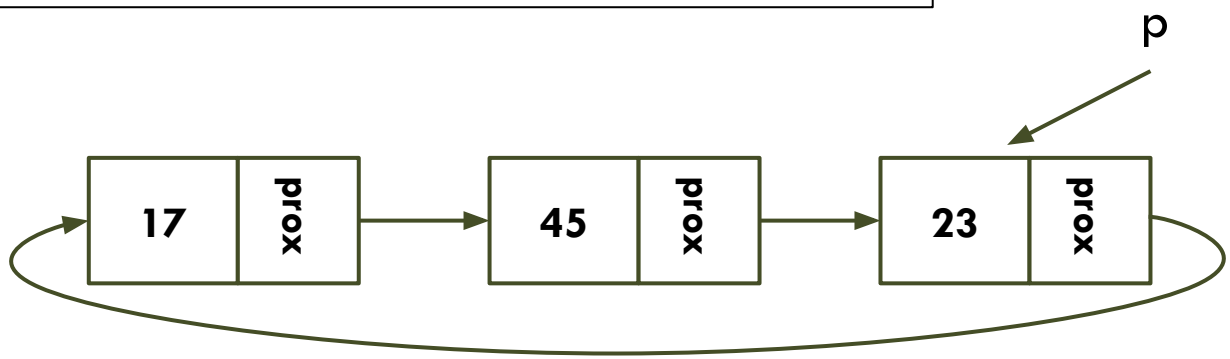


Lista encadeada circular

- **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:

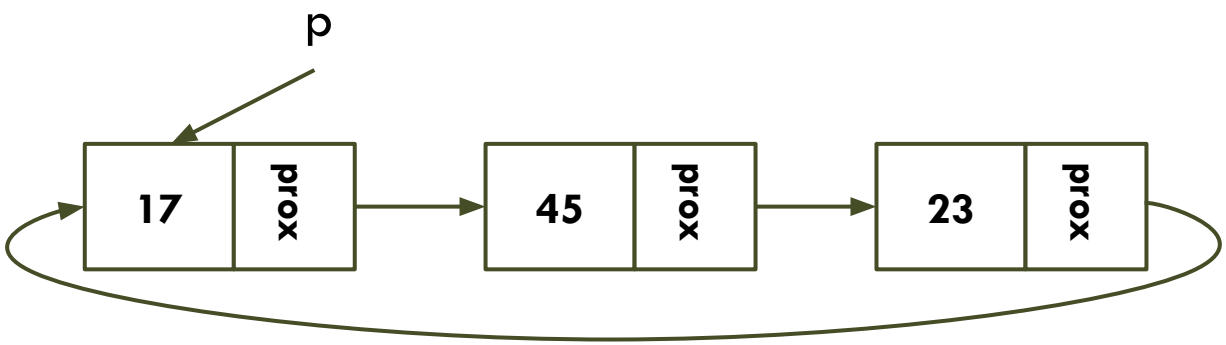


Lista encadeada circular

- **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:

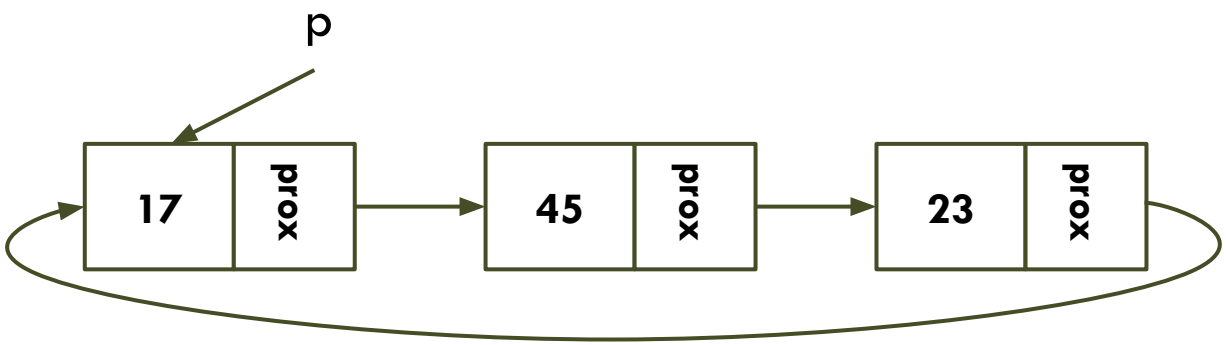


Lista encadeada circular

- **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

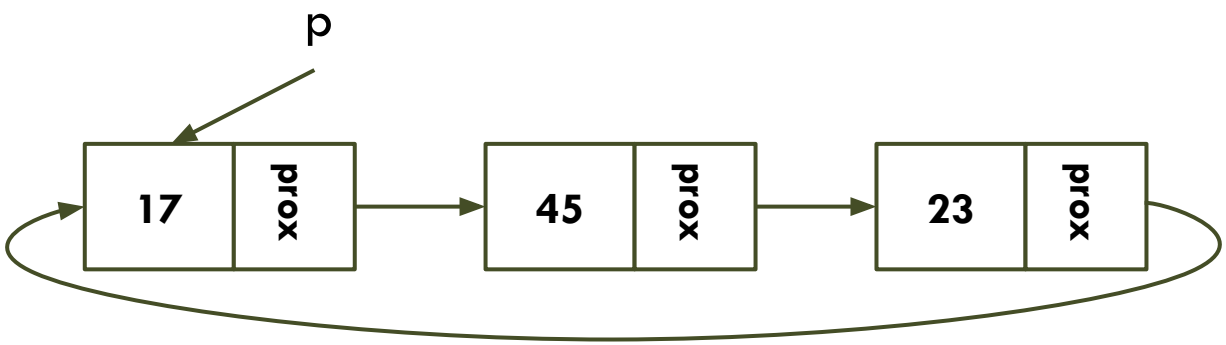


Lista encadeada circular

- **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

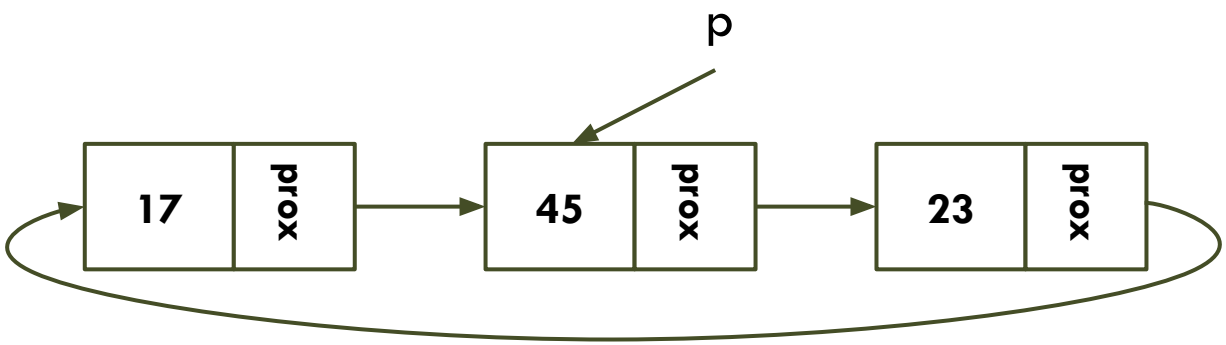


Lista encadeada circular

- **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

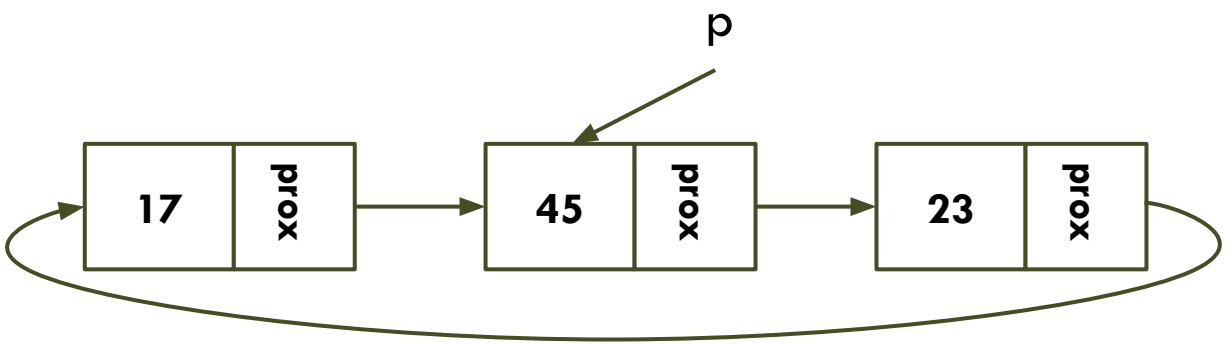


Lista encadeada circular

- **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

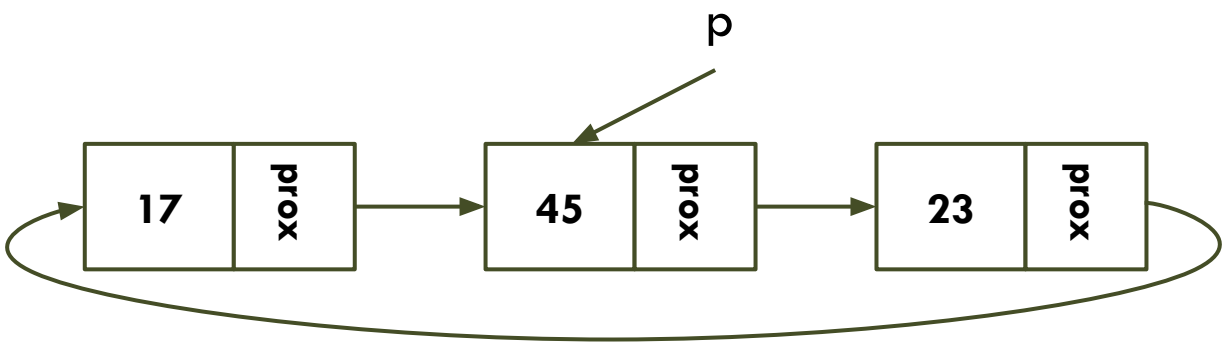


Lista encadeada circular

- **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

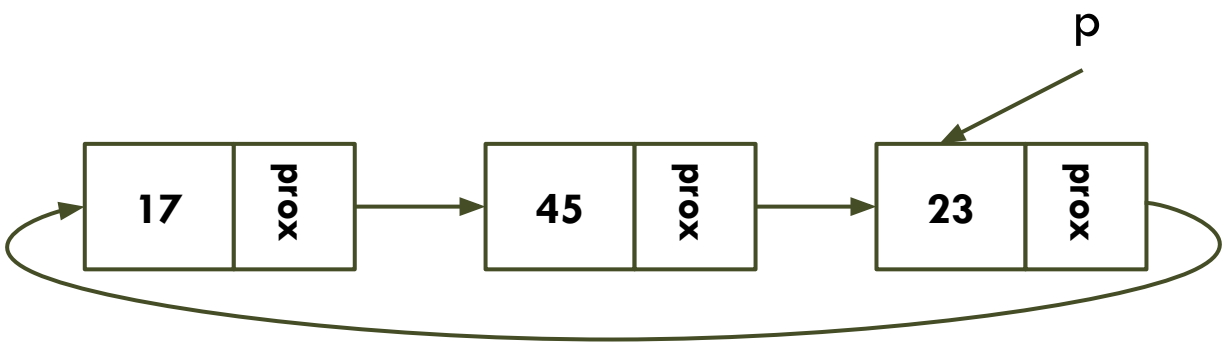


Lista encadeada circular

- **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

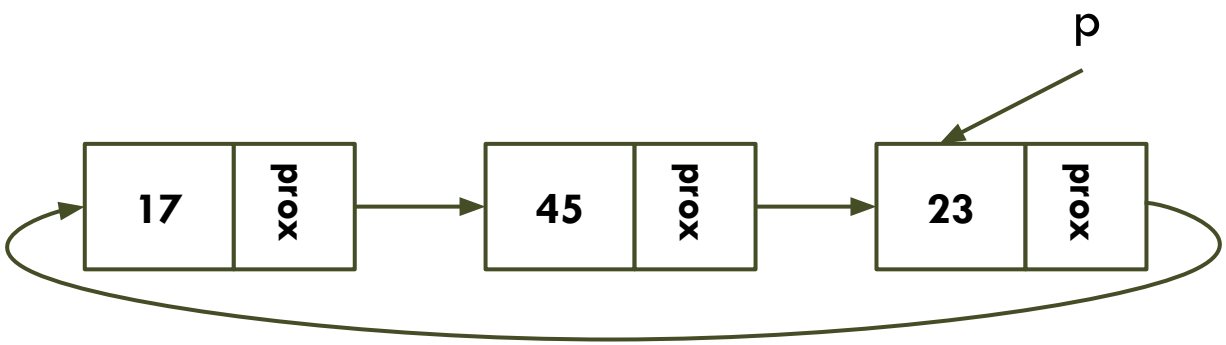


Lista encadeada circular

- **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

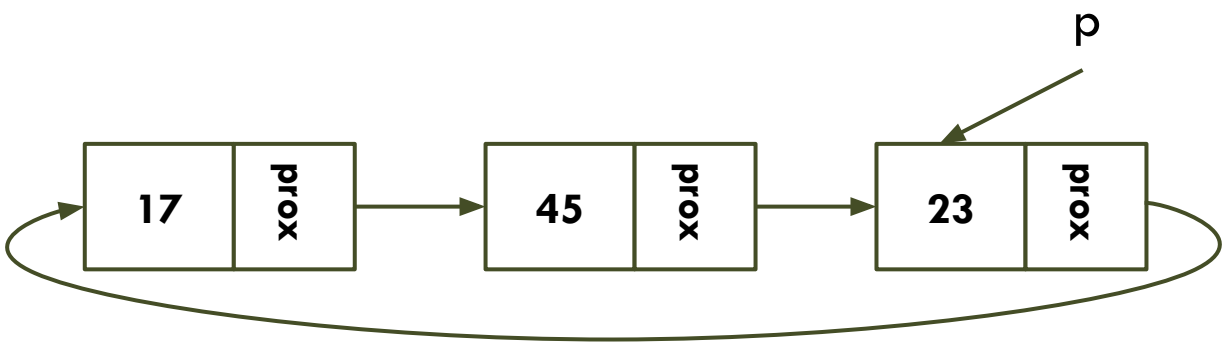


Lista encadeada circular

- **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

- **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

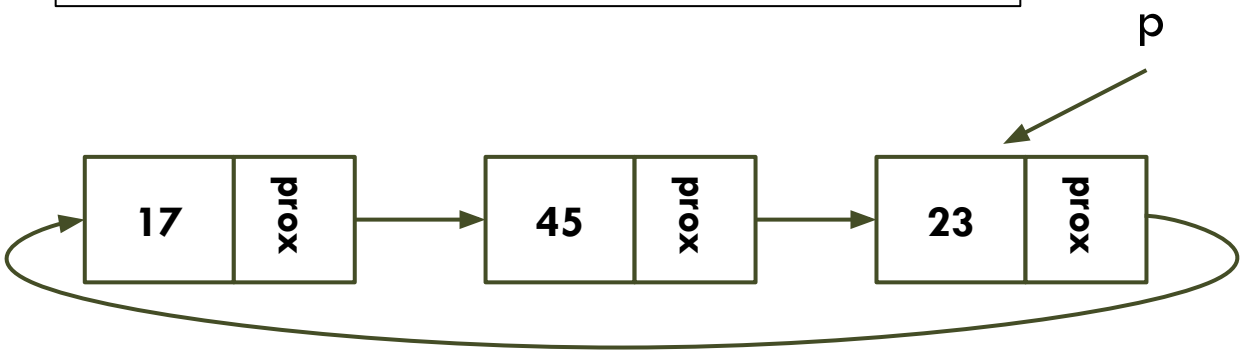
```
ListaC* busca (ListaC* l, int v){
    ListaC* p = l;
    if (p){
        do {
            p = p->prox; /* avança para o próximo nó */
            if(p->info == v){
                return p;
            }
        } while (p != l);
    }
    return NULL;
}
```

Lista encadeada circular

- **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

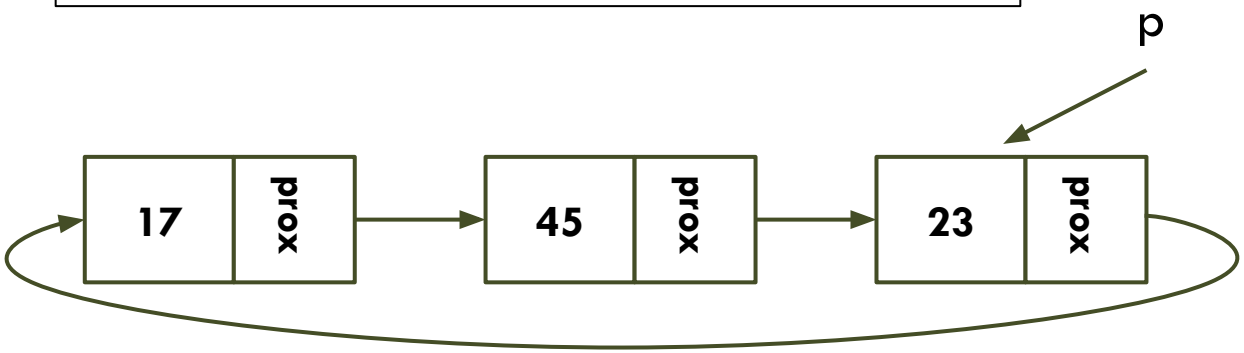


Lista encadeada circular

- **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

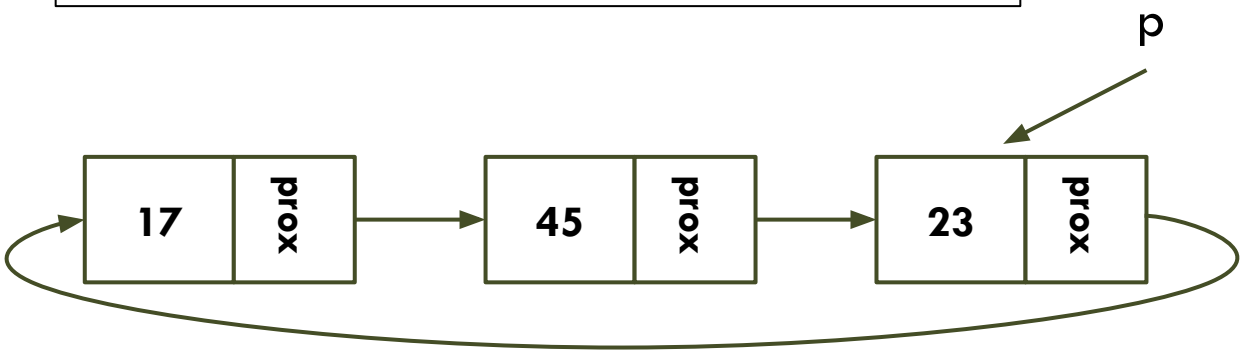


Lista encadeada circular

- **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

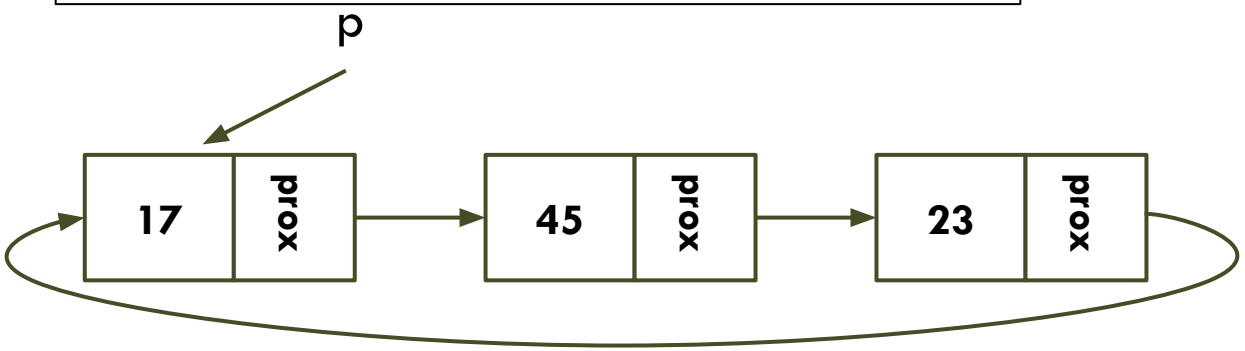


Lista encadeada circular

- **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

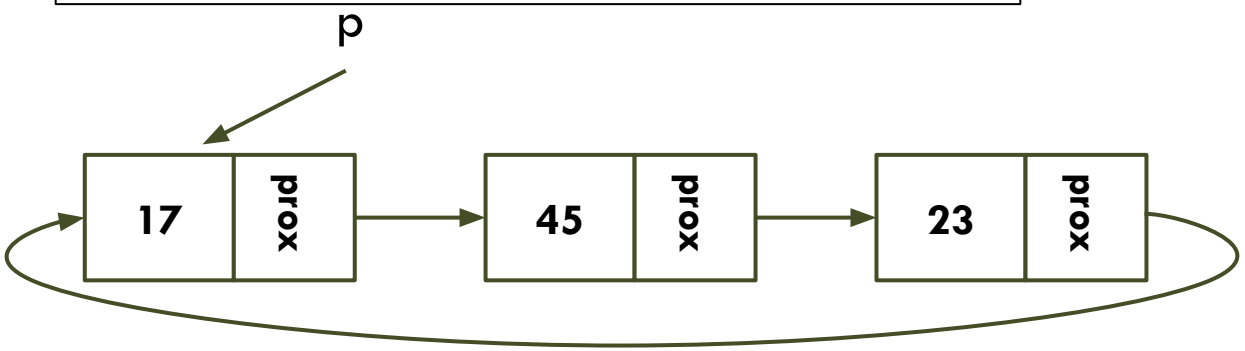


Lista encadeada circular

- **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

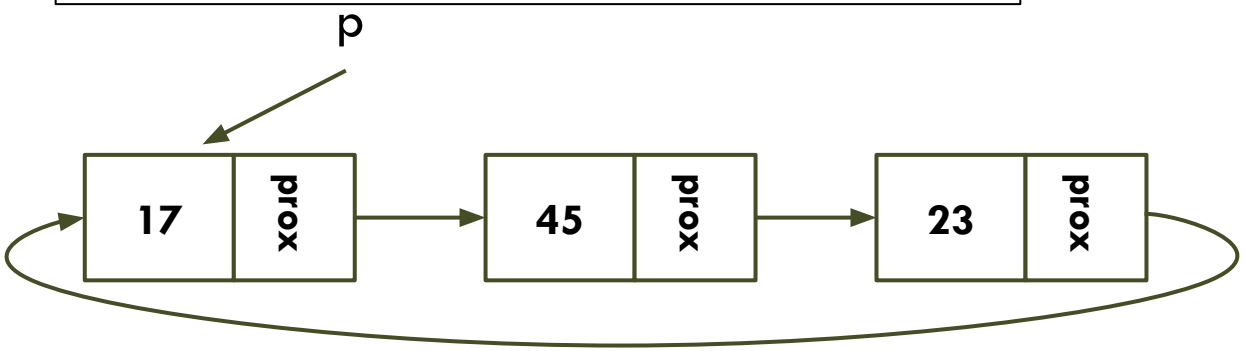


Lista encadeada circular

- **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

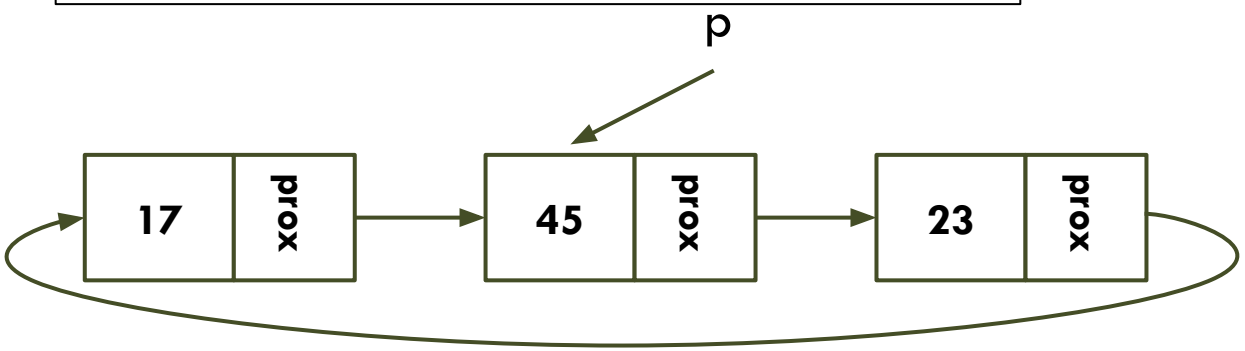


Lista encadeada circular

- **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

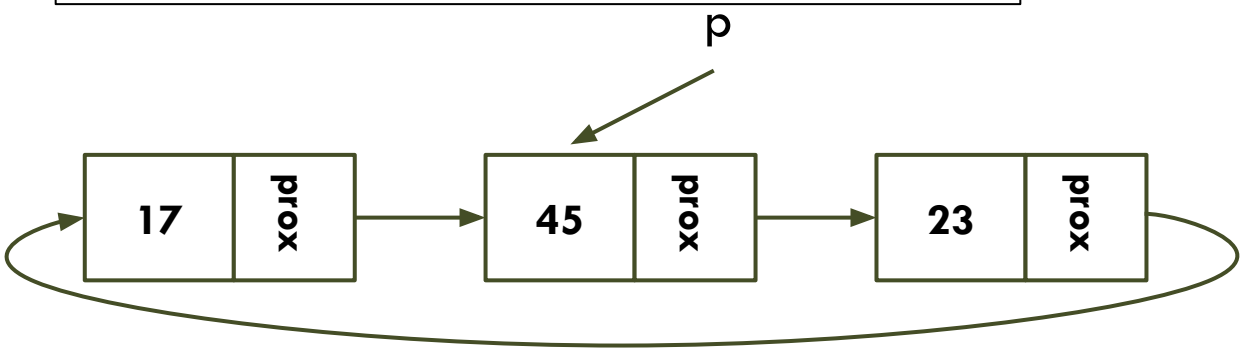


Lista encadeada circular

- **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



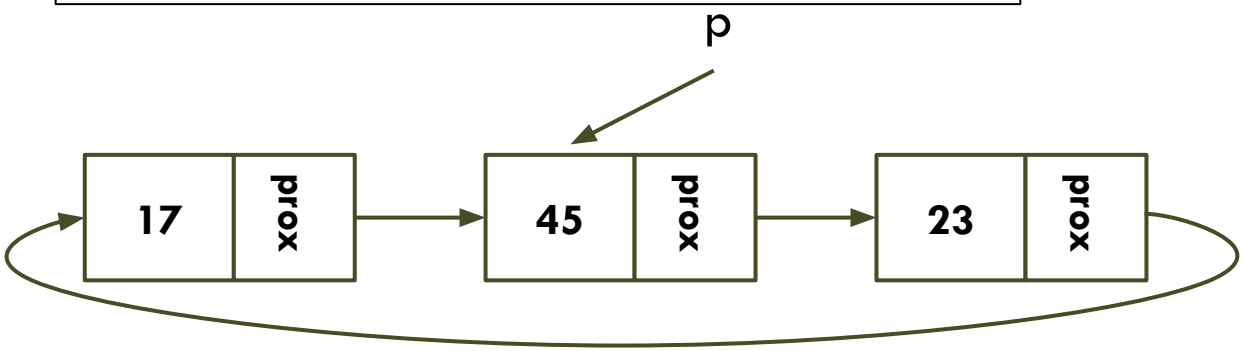
Lista encadeada circular

- **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;
}
```

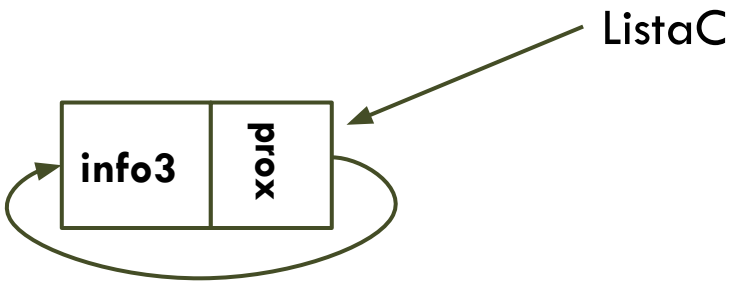
Buscando: 45

Retorna p



Lista encadeada circular

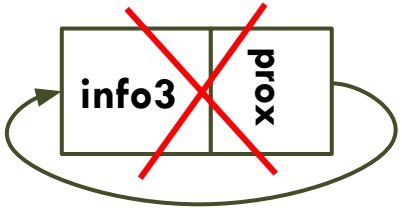
- **Função retira**
  - Recebe como entrada a lista e o valor do elemento a retirar
  - Se existe um único elemento na lista, remove o elemento e retorna NULL



Lista encadeada circular

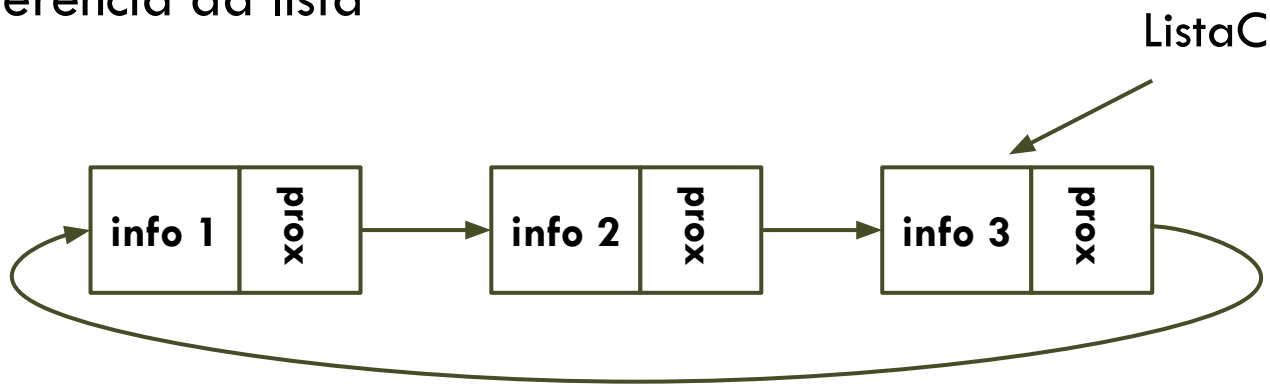
- **Função retira**
  - Recebe como entrada a lista e o valor do elemento a retirar
  - Se existe um único elemento na lista, remove o elemento e retorna NULL

ListaC → NULL



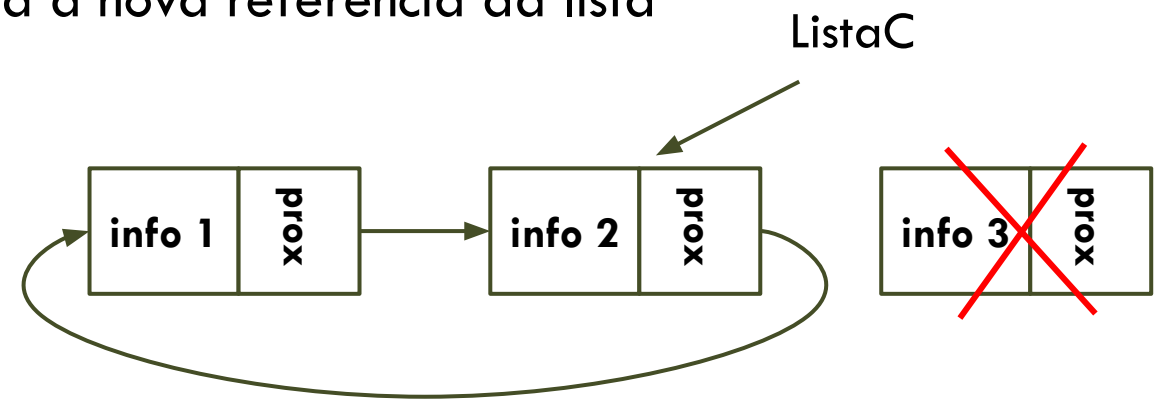
Lista encadeada circular

- **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



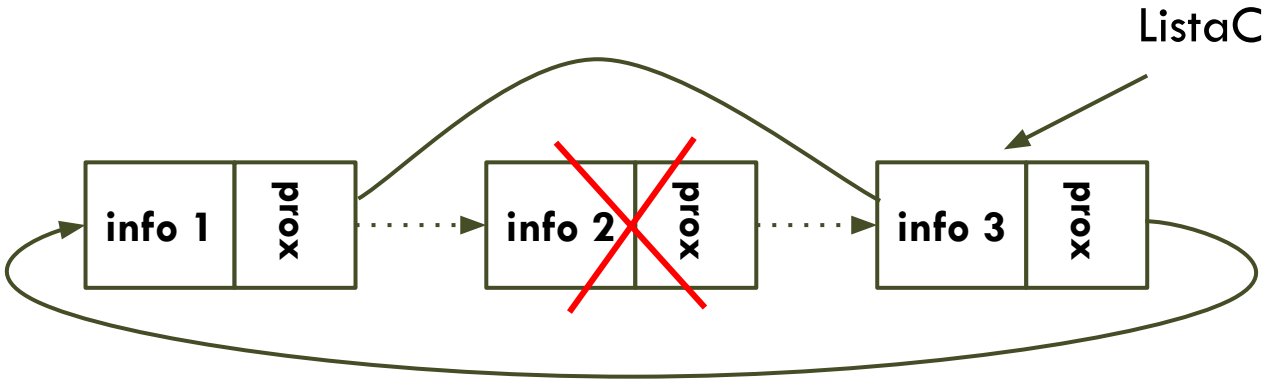
Lista encadeada circular

- **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

- **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

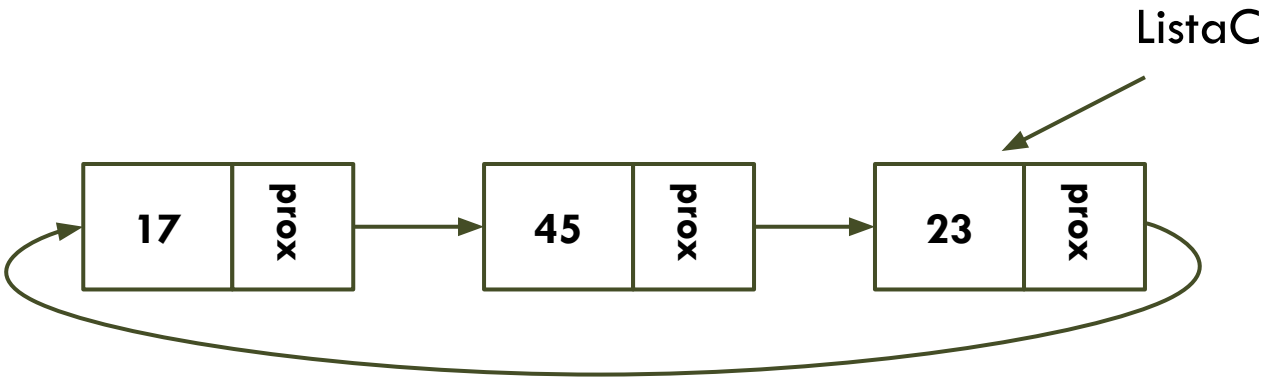
- **Função retira**

```
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

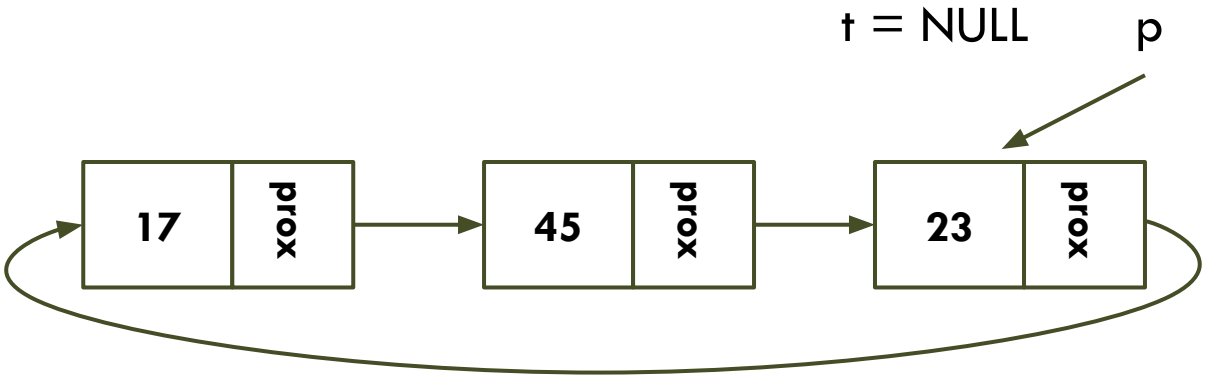
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

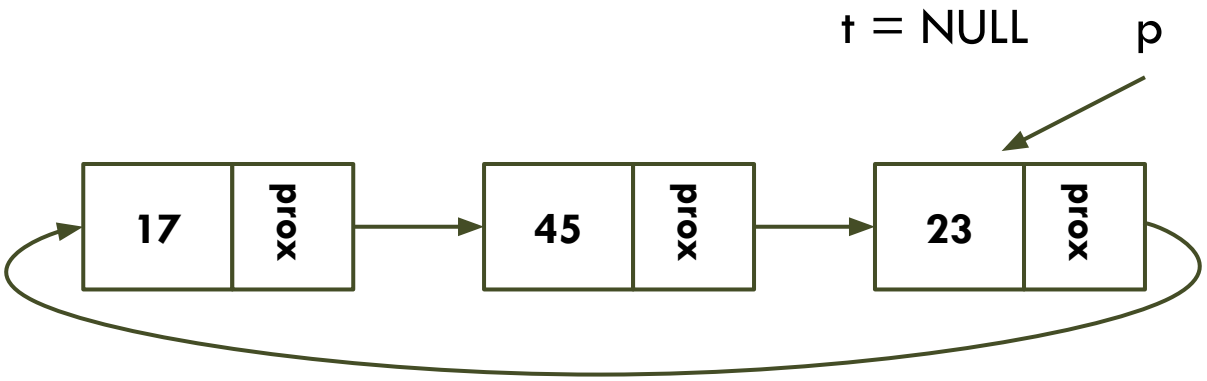
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

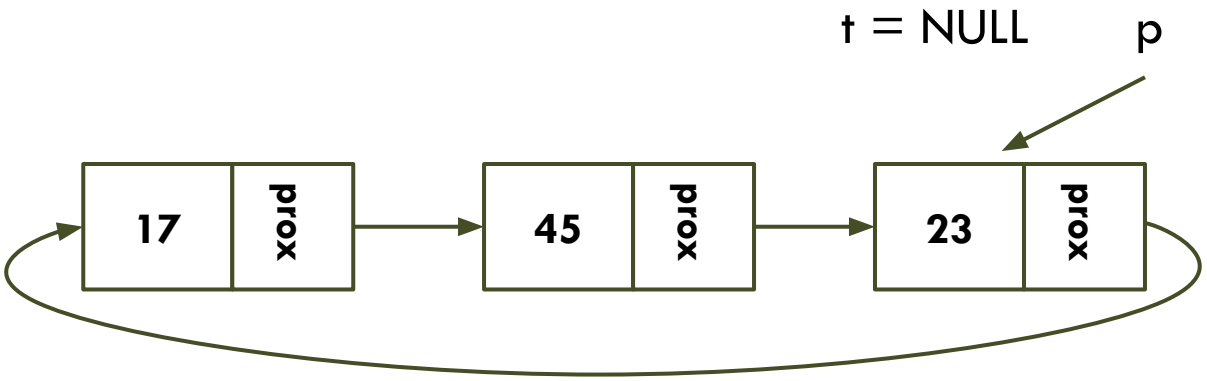
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

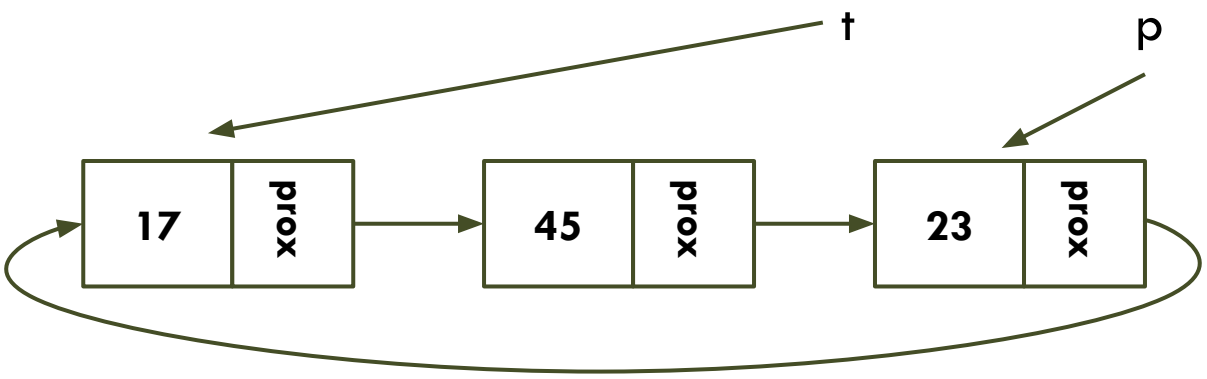
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

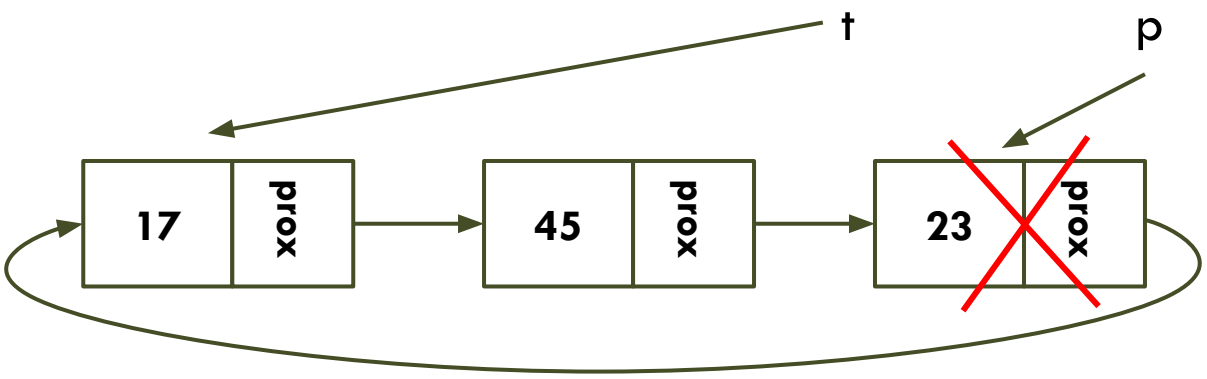
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

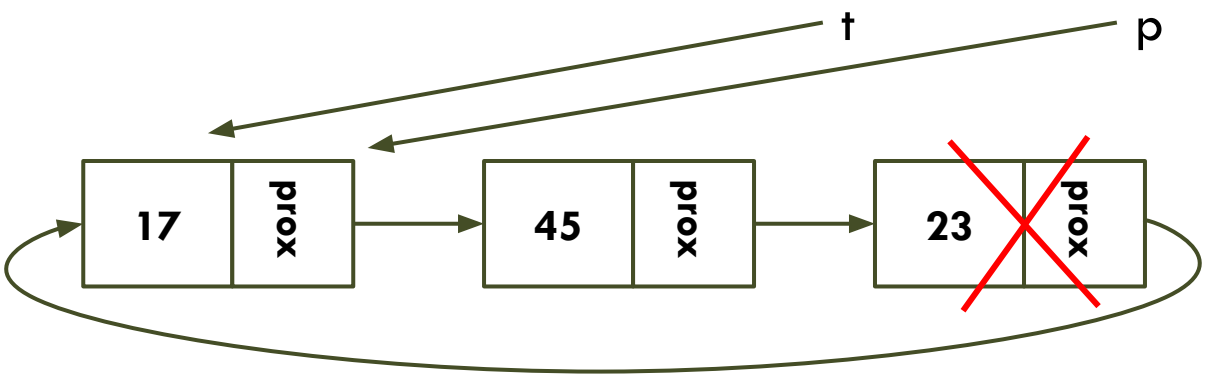
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

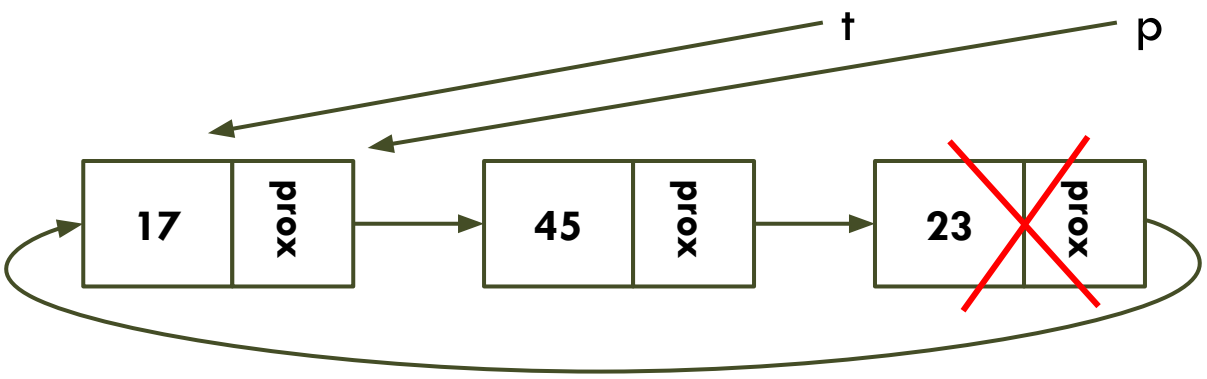
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

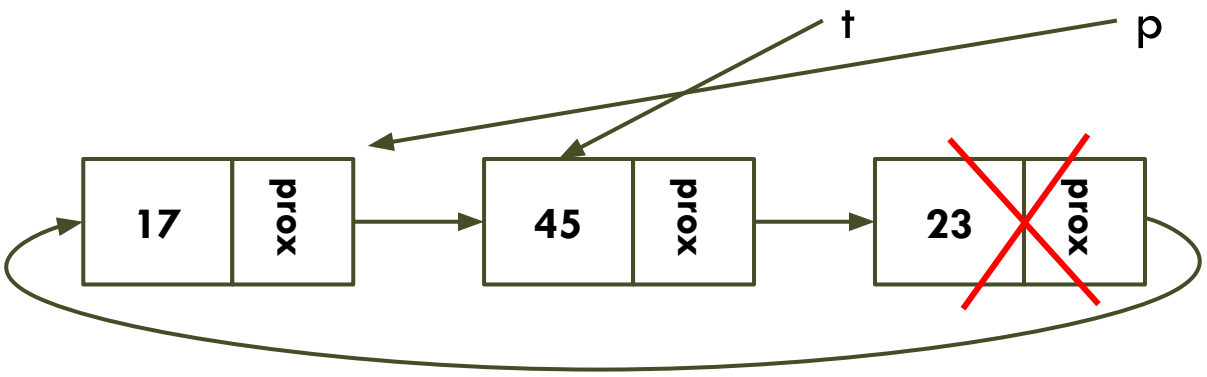
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

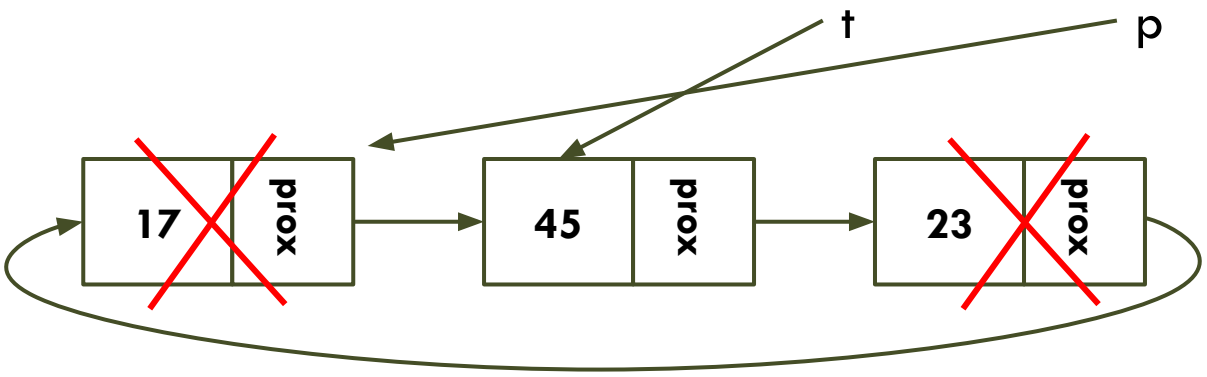
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

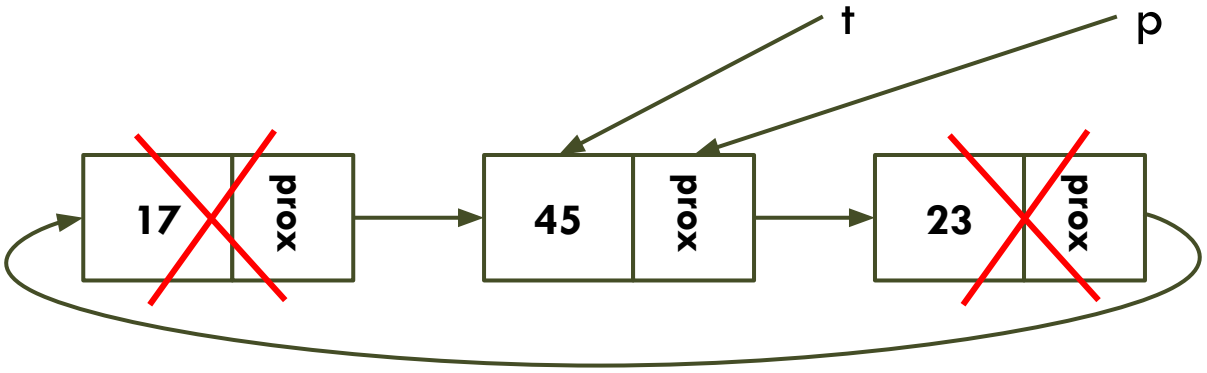
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

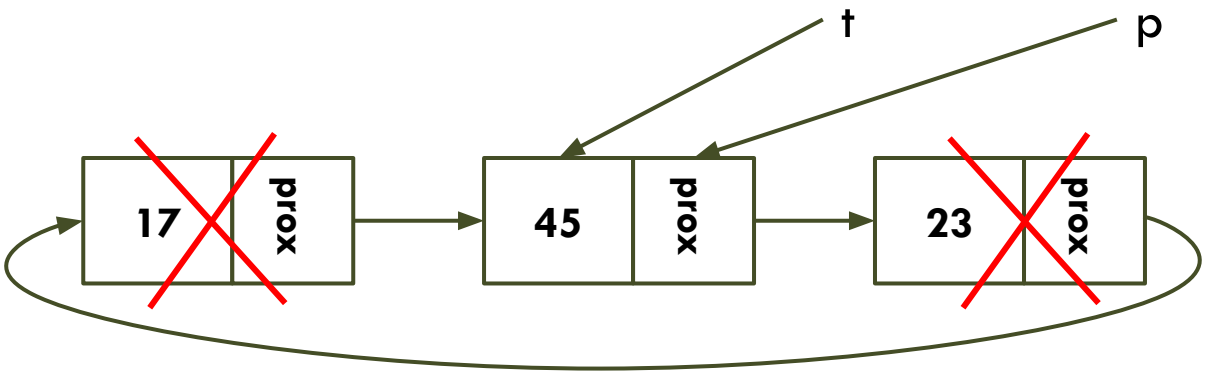
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

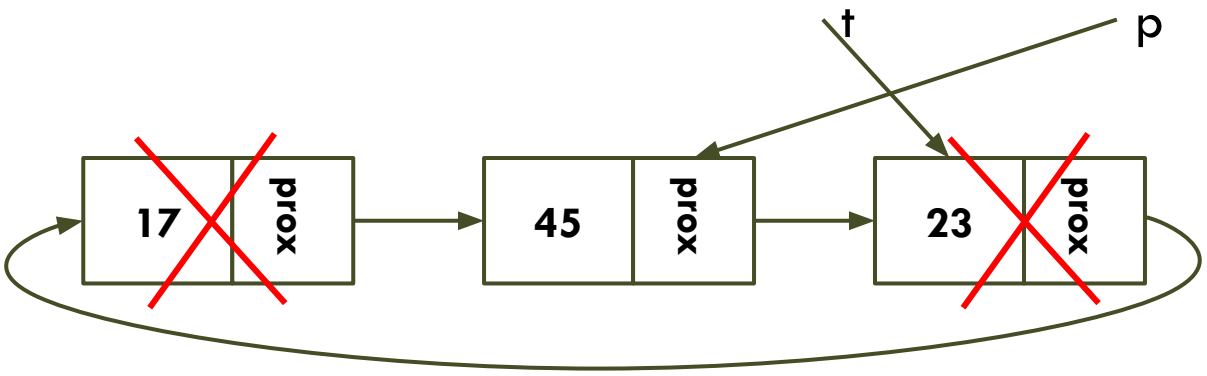
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

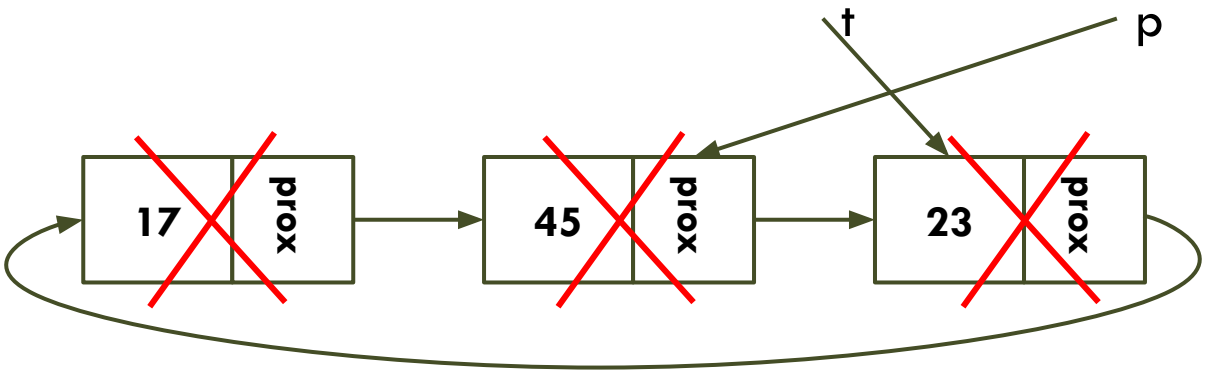
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

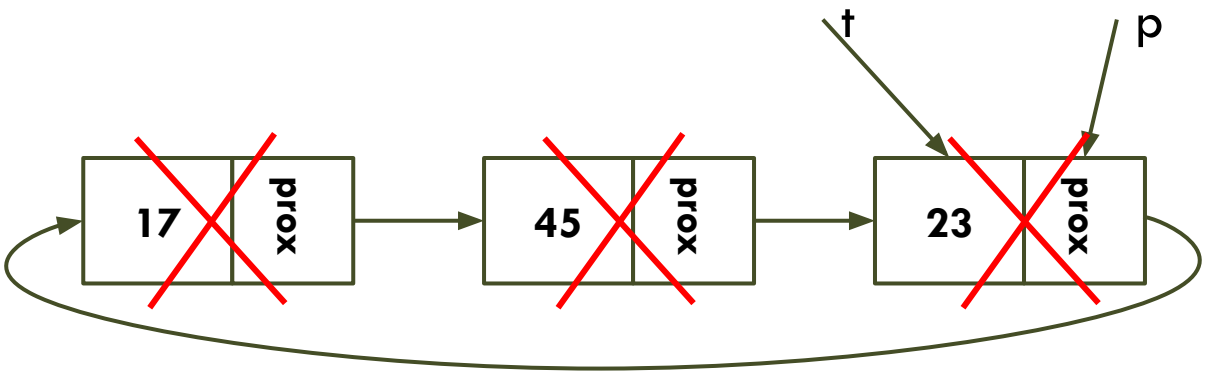
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

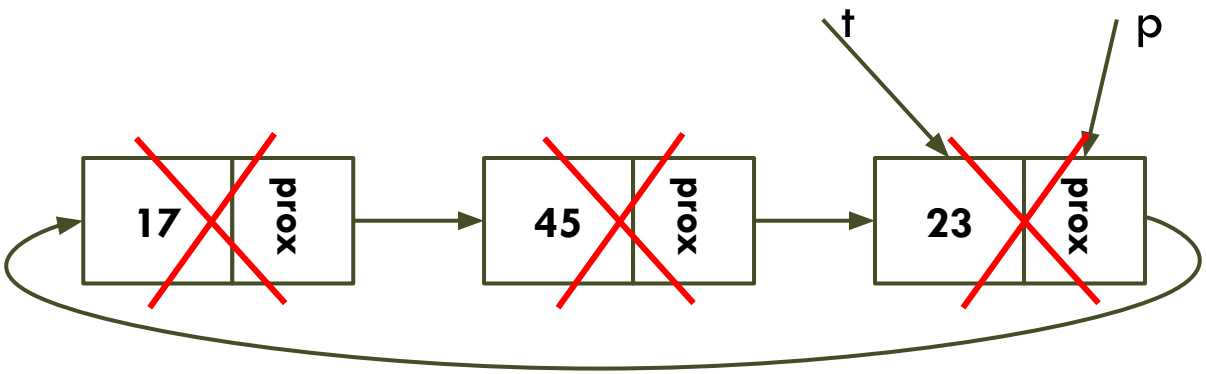
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

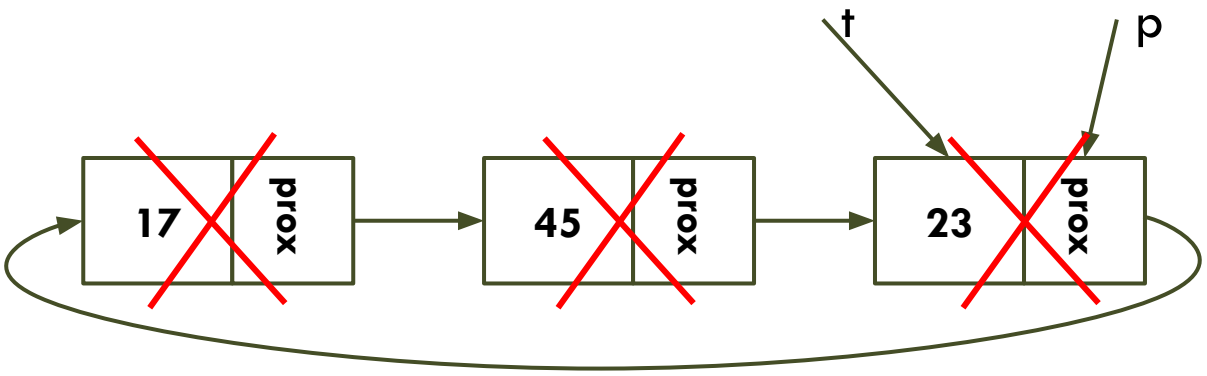
- **Função libera**
  - Libera todos os elementos (nós) alocados




```
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

- **Função libera**
  - Libera todos os elementos (nós) alocados

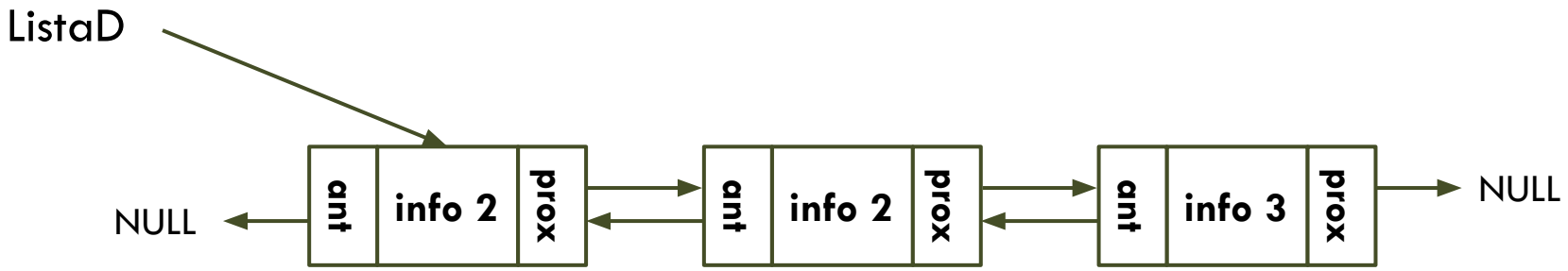



```
ListaC* libera (ListaC* l){
    ListaC* p = l;
    ListaC* t = NULL;
    if(p){
        do{
            t = p->prox;
            free(p);
            p = t;
        }while(p != l);
    }
    return NULL;
}
```

Lista duplamente encadeada e lista circular duplamente encadeada

Lista duplamente encadeada

- 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

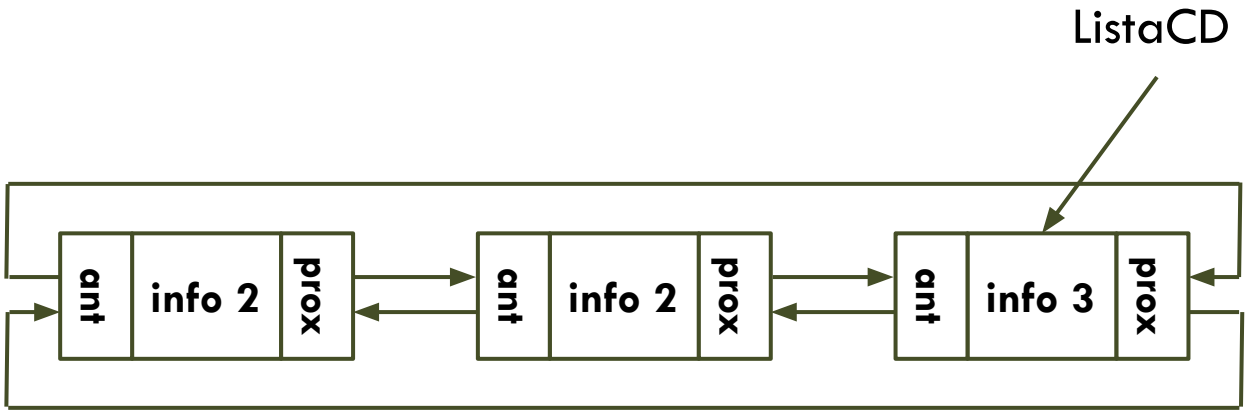
- **Exemplo**
  - Lista duplamente encadeada armazenando valores inteiros

```
typedef struct listaD ListaD;

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

Lista circular duplamente encadeada

- 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

- **Exemplo**
  - Lista circular duplamente encadeada armazenando valores inteiros

```
typedef struct listaCD ListaCD;

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

Vantagens de listas

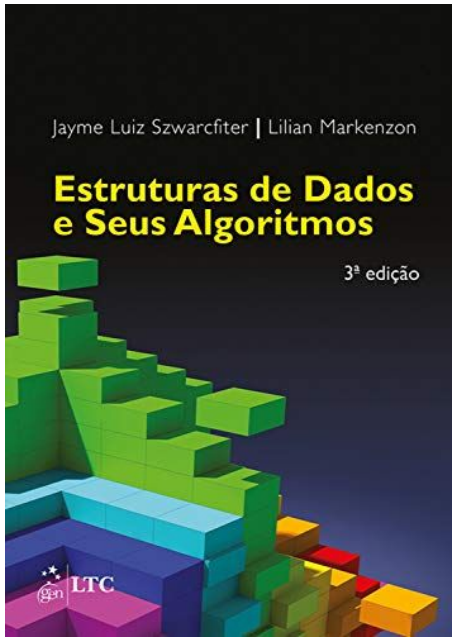
- Maior flexibilidade
  - Conjunto de dados pode crescer ou diminuir
- Evita o desperdício de memória
- Elementos podem ser inseridos e removidos em posições específicas

Atividade

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

Referências

SCHILDT, 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.