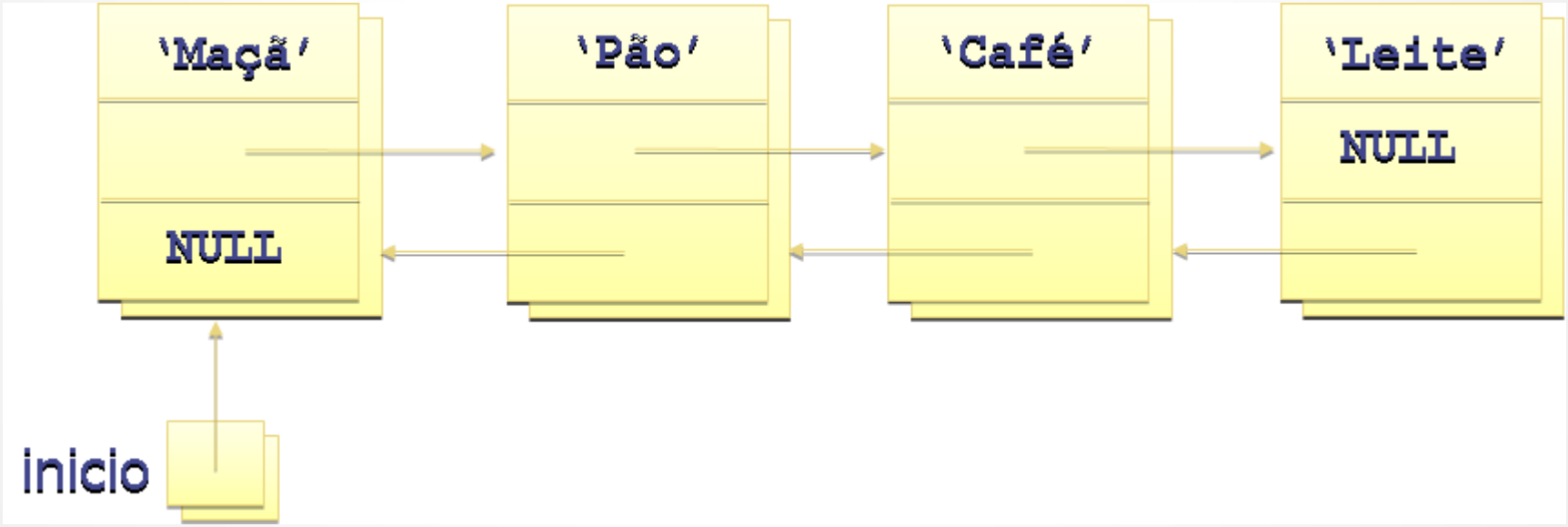
Listas Duplamente Ligadas

Listas Duplamente Ligadas

- Nesta aula vamos implementar as operações do TAD listas utilizando Listas Duplamente Ligadas



Listas Duplamente Ligadas

- Nas Listas duplamente ligadas, cada nó mantém um **ponteiro** para o **nó anterior e posterior**

Listas Duplamente Ligadas

- Nas Listas duplamente ligadas, cada nó mantém um ponteiro para o nó anterior e posterior

- A manipulação da lista é mais complexa, porém algumas operações são diretamente beneficiadas

Listas Duplamente Ligadas

- Nas Listas duplamente ligadas, cada nó mantém um ponteiro para o nó anterior e posterior

- A manipulação da lista é mais complexa, porém algumas operações são diretamente beneficiadas

- Por exemplo, as operações de **inserção** e **remoção** em uma dada posição

TAD: Listas Duplamente Ligadas

```
typedef struct {
   int chave;
   int valor;
} ITEM;
```

```
typedef struct NO {
   ITEM item;
   struct NO *proximo;
   struct NO *anterior;
} tNO;
```

```
typedef struct {
   tNO *inicio;
   tNO *fim;
} LISTA_Duplamente_Ligada;
```

Refazer os códigos considerando apenas um ponteiro para o cabeça da lista

Lista Duplamente Ligadas: Principais Operações

- Criar lista
- Limpar lista
- Verificar se a lista está vazia
- Imprime lista
- Inserir item (última posição)
- Inserir item (primeira posição)
- Remover item (última posição)
- Remover item (primeira posição)
- Recuperar item (dado uma chave)
- Recuperar item (por posição)
- Conta número de itens

Lista Duplamente Ligadas

- As operações de criar e apagar a lista são simples

```
void criar (Lista_Duplamente_Ligada *lista){
   lista->inicio = NULL;
   lista->fim = NULL;
}
```

```
void apagar_lista (Lista_Duplamente_Ligada *lista){
   if (!vazia(lista)){
      tNO *paux = lista->inicio;
      while (paux != NULL) {
        tNO *prem = paux;
        paux = paux->proximo;
        free(prem);
      }
  }
  lista->inicio=NULL;
  lista->fim = NULL;
}
```

Lista Duplamente Ligadas

- Inserir Item (Primeira Posição)

```
int inserir_inicio (Lista_Duplamente_Ligada *lista, ITEM *item){




}
```

Lista Duplamente Ligadas

- Inserir Item (Primeira Posição)

```
int inserir_inicio (Lista_Duplamente_Ligada *lista, ITEM *item){
   tNO *pnovo = (tNO *)malloc(sizeof(tNO));









}
```

Lista Duplamente Ligadas

- Inserir Item (Primeira Posição)

```
void inserir_inicio (Lista_Duplamente_Ligada *lista, ITEM *item){
   tNO *pnovo = (tNO *)malloc(sizeof(tNO));
   if (pnovo !=NULL) {
      pnovo->item = *item;
      pnovo->proximo = lista->inicio;
      pnovo->anterior = NULL;









}
```

Lista Duplamente Ligadas

- Inserir Item (Primeira Posição)

```
int inserir_inicio (Lista_Duplamente_Ligada *lista, ITEM *item){
   tNO *pnovo = (tNO *)malloc(sizeof(tNO));
   if (pnovo !=NULL) {
      pnovo->item = *item;
      pnovo->proximo = lista->inicio;
      pnovo->anterior = NULL;
      if (lista->inicio != NULL){
           lista->inicio->anterior = pnovo;
      } else {
           lista->fim = pnovo;
      }



}
```

Lista Duplamente Ligadas

- Inserir Item (Primeira Posição)

```
int inserir_inicio (Lista_Duplamente_Ligada *lista, ITEM *item){
   tNO *pnovo = (tNO *)malloc(sizeof(tNO));
   if (pnovo !=NULL) {
      pnovo->item = *item;
      pnovo->proximo = lista->inicio;
      pnovo->anterior = NULL;
      if (lista->inicio != NULL){
           lista->inicio->anterior = pnovo;
      } else {
           lista->fim = pnovo;
      }
     lista->inicio = pnovo;
     return 1;
    } else {
      return 0;
    }
}
```

Lista Duplamente Ligadas

- Inserir Item (Última Posição)

```
int inserir_fim (Lista_Duplamente_Ligada *lista, ITEM *item){
   tNO *pnovo = (tNO *)malloc(sizeof(tNO));
   if (pnovo !=NULL) {
      pnovo->item = *item;
      pnovo->proximo = NULL;
      pnovo->anterior = lista->fim;
      if (lista->fim != NULL){
           lista->fim->proximo = pnovo;
      } else {
           Lista ->inicio = pnovo;
      }
      lista->fim = pnovo;
      return 1;
    } else {
      return 0;
 }
```

Lista Duplamente Ligadas

- Remover Item (Primeira Posição)

```
int remover_inicio (Lista_Duplamente_Ligada *lista){
   if (!vazia(lista)){
      tNO *paux = lista->inicio;
      if (lista->inicio == lista->fim) {
          lista->inicio = NULL;
          lista->fim = NULL;
      } else {
          lista->inicio->proximo->anterior = NULL;
          Lista->inicio = lista ->inicio->proximo;
      }
      free(paux);
      return 1;
     }else {
      return 0;
    }
 }
```

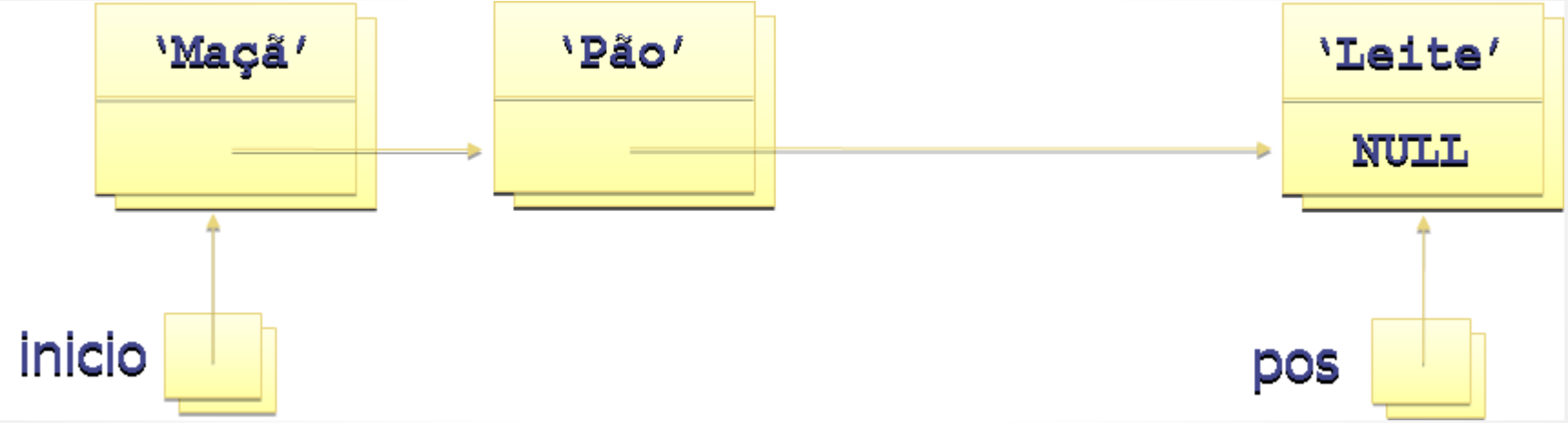
Lista Duplamente Ligadas

- Remover Item (Última Posição)

```
int remover_fim (Lista_Duplamente_Ligada *lista){
   if (!vazia(lista)){
      tNO *paux = lista->fim;
      if (lista->inicio == lista->fim) {
          lista->inicio = NULL;
          lista->fim = NULL;
      }else {
          lista->fim->anterior->proximo = NULL;
          Lista->fim = lista->fim->anterior;
      }
      free(paux);
      return 1;
    }else {
     return 0;
    }
 }
```

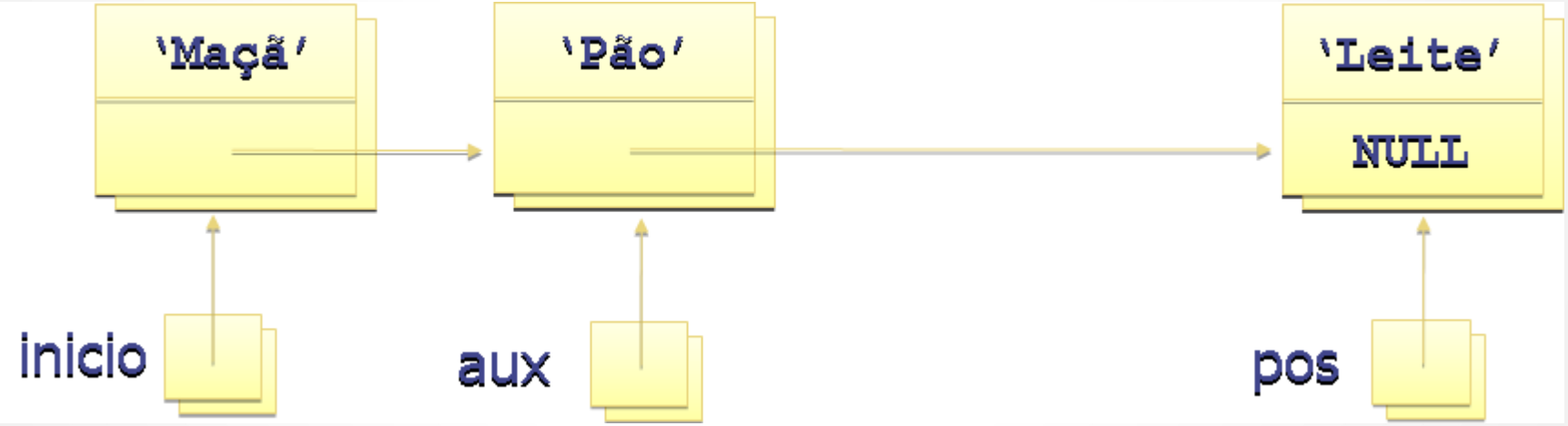
Inserir Item (Por posição)

- Na lista ligada simples, a operação de inserção dada uma posição também envolve encontrar o nó anterior ao que será inserido



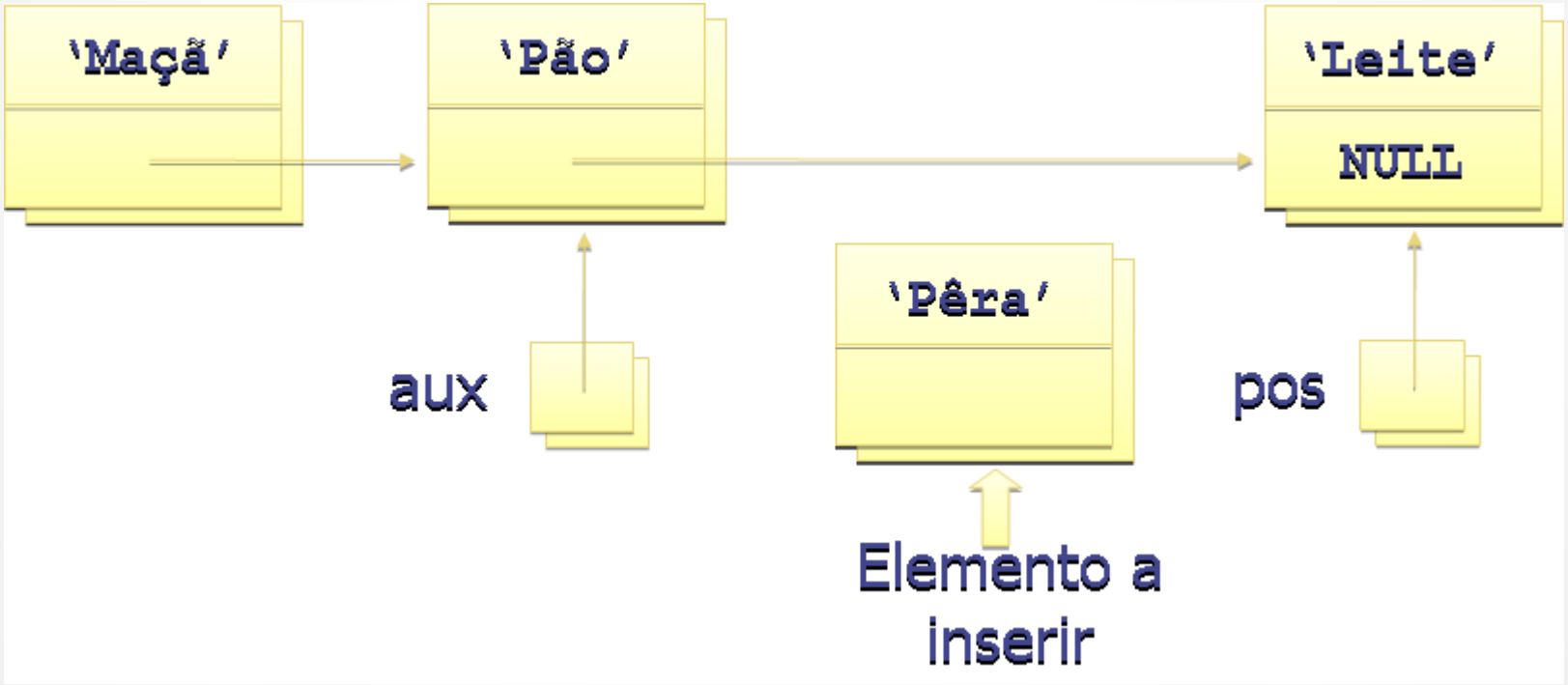
Inserir Item (Por posição)

- Na lista ligada simples, a operação de inserção dada uma posição também envolve encontrar o nó anterior ao que será inserido



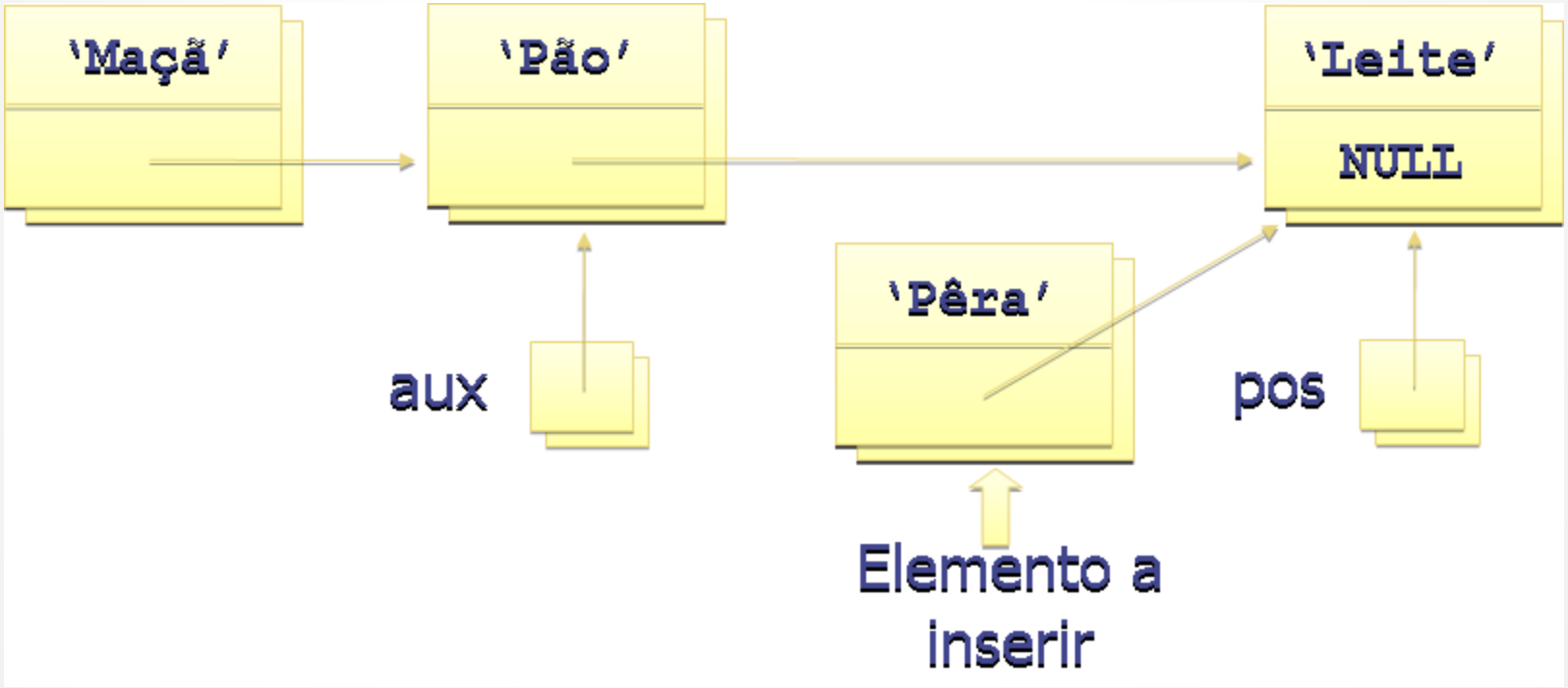
Inserir Item (Por posição)

- Na lista ligada simples, a operação de inserção dada uma posição também envolve encontrar o nó anterior ao que será inserido



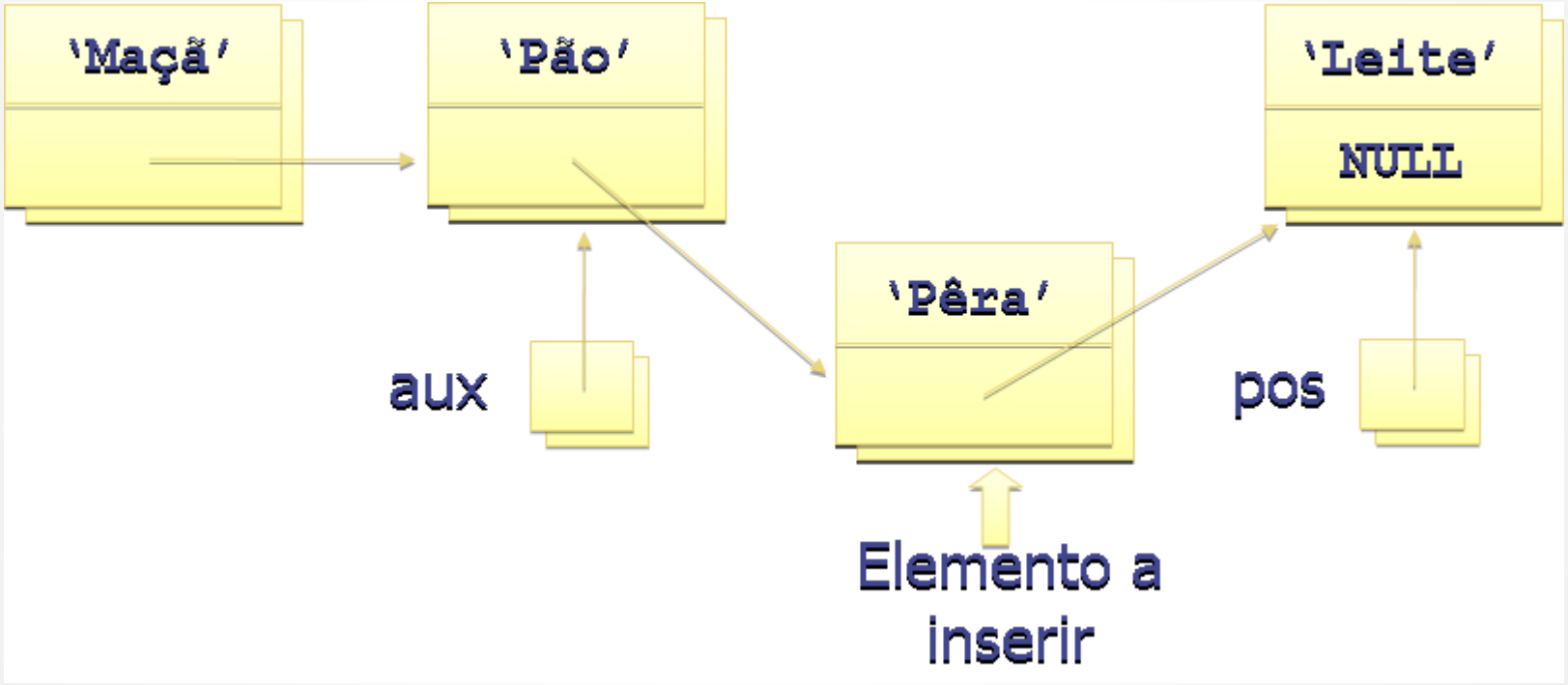
Inserir Item (Por posição)

- Na lista ligada simples, a operação de inserção dada uma posição também envolve encontrar o nó anterior ao que será inserido



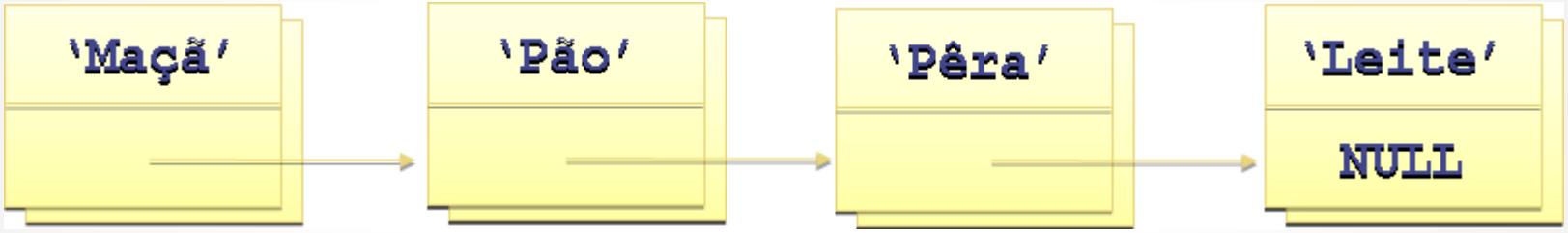
Inserir Item (Por posição)

- Na lista ligada simples, a operação de inserção dada uma posição também envolve encontrar o nó anterior ao que será inserido



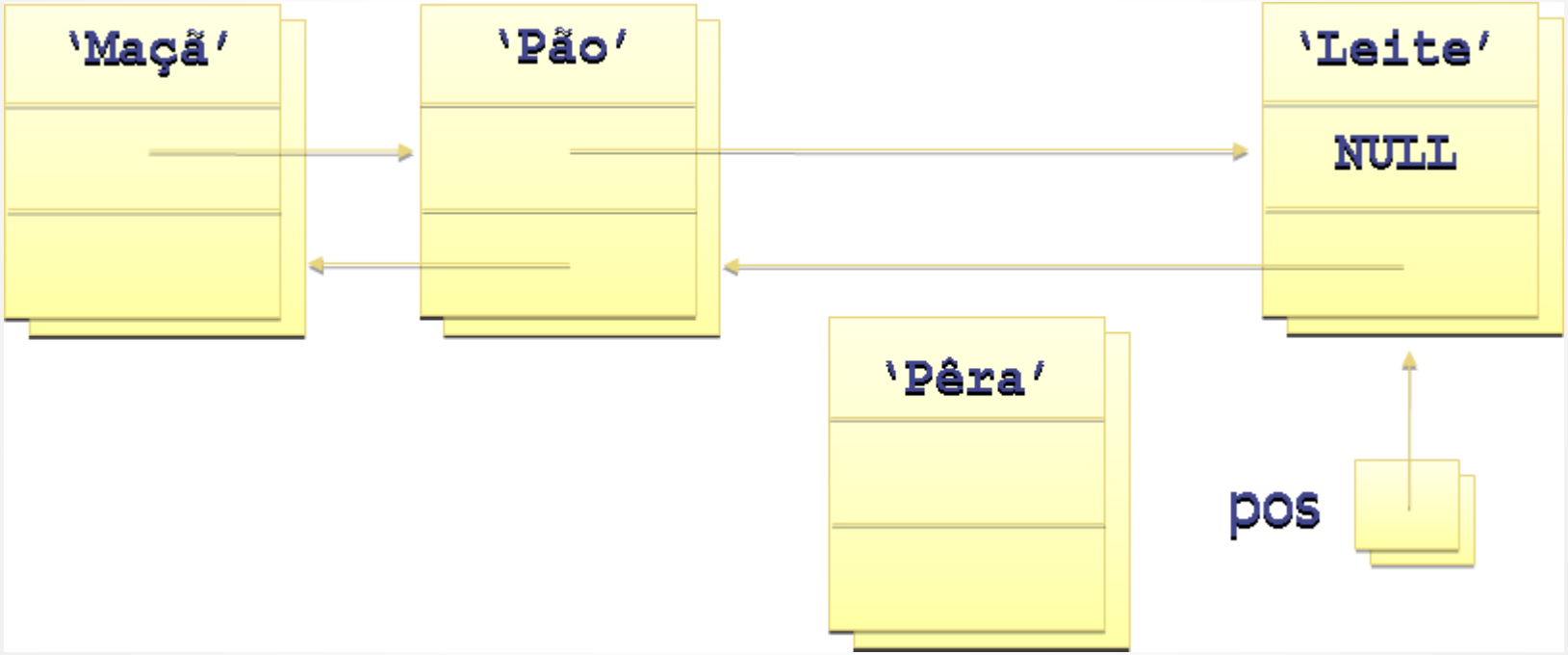
Inserir Item (Por posição)

- Na lista ligada simples, a operação de inserção dada uma posição também envolve encontrar o nó anterior ao que será inserido



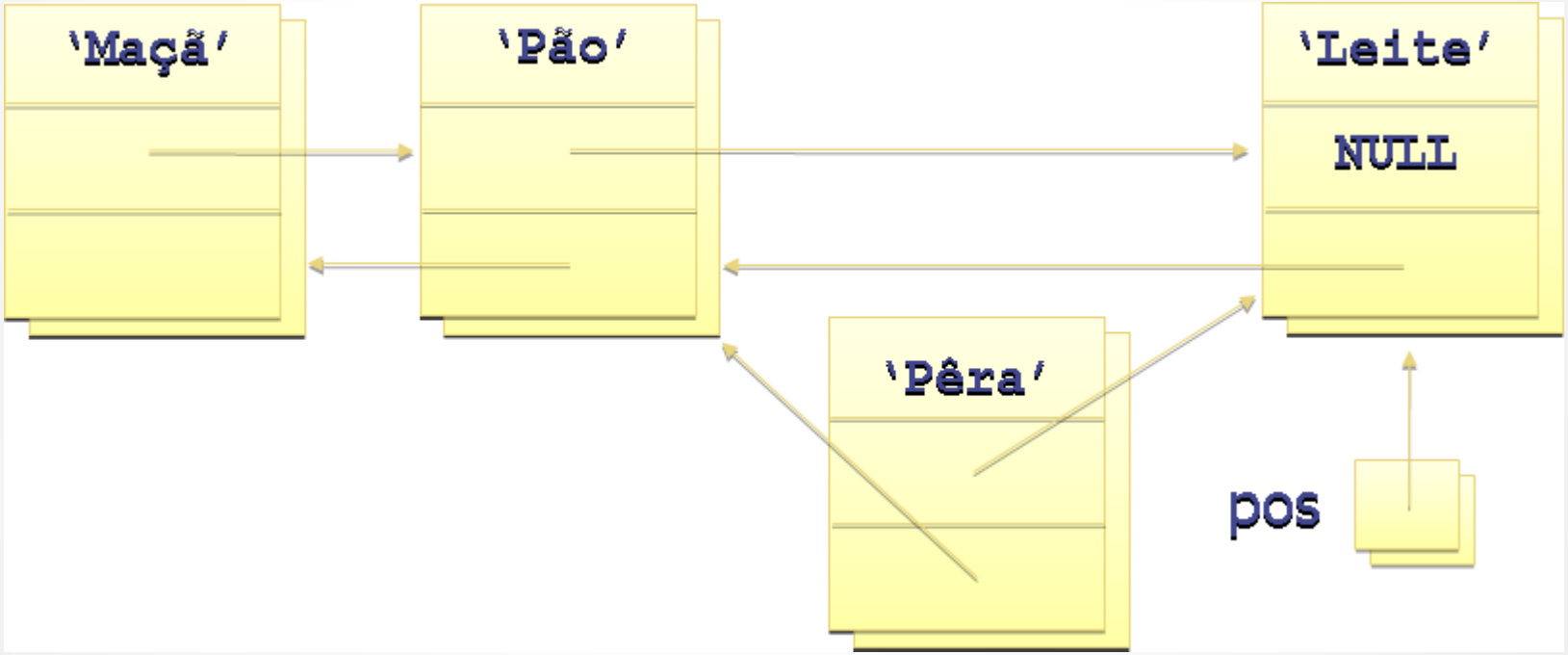
Inserir Item (Por posição)

- Na lista ligada duplamente ligada, a operação de remoção dado uma posição também envolve apenas manipular os ponteiros



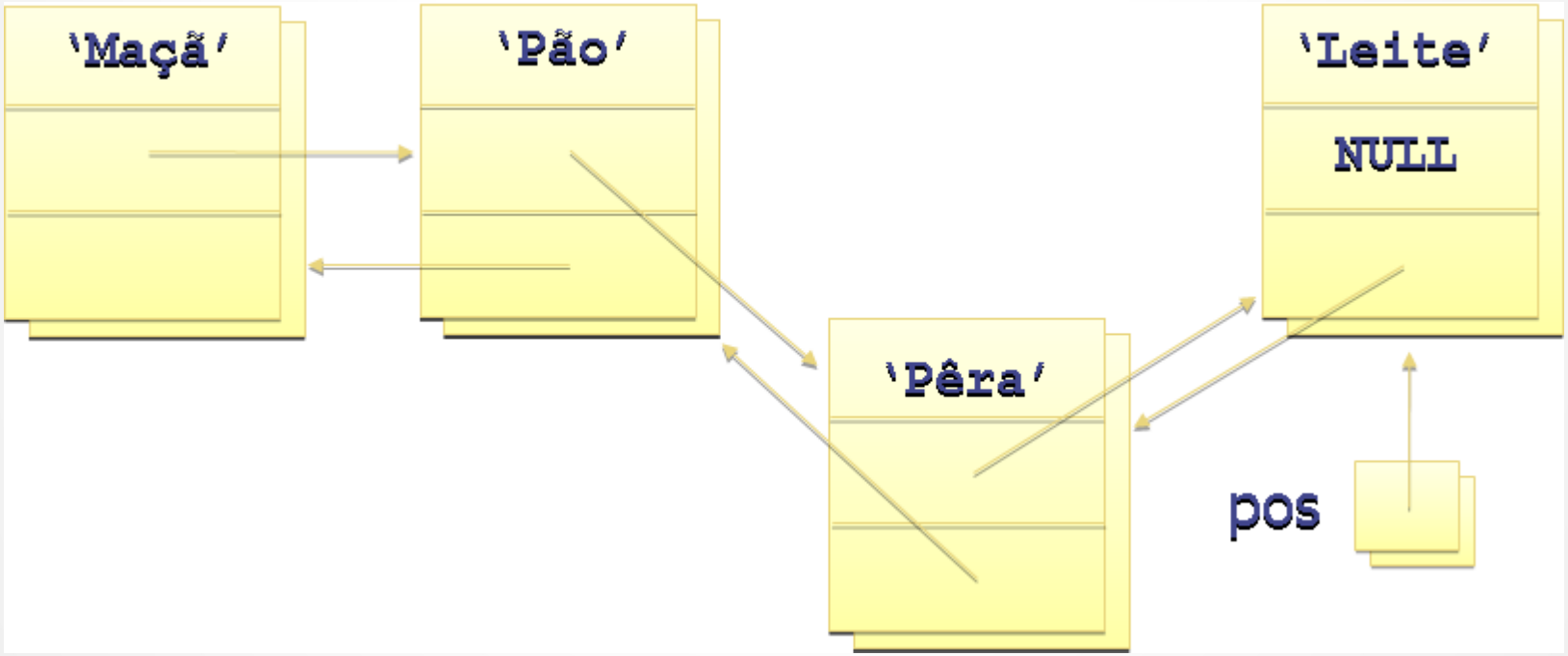
Inserir Item (Por posição)

- Na lista ligada duplamente ligada, a operação de remoção dado uma posição também envolve apenas manipular os ponteiros



Inserir Item (Por posição)

- Na lista ligada duplamente ligada, a operação de remoção dado uma posição também envolve apenas manipular os ponteiros



```
int inserir_posicao (Lista_Duplamente_Ligada *lista, int pos, ITEM *item){

   tNO *pnovo = (tNO *)malloc (sizeof(tNO)); //cria um novo nó
   if (pnovo != NULL) {//verifica se existe memoria disponivel
      pnovo->item = *item;
      if (pos==0) {//adiciona na primeira posicao
         pnovo->proximo = lista->inicio;
         lista->inicio->anterior = pnovo;
         lista->inicio = pnovo;
         pnovo->anterior = NULL;
       } else {
       tNO *paux = lista->inicio;
       //encontra a posição de inserção
       for (int i=0; i<pos; i++){
           if (paux != lista->fim) {
               paux = paux->proximo;
           } else{
           return 0;
           }
        }
        // faz as ligações para a inserção do novo elemento
        pnovo->proximo =paux;
        pnovo->anterior = paux->anterior;
        paux->anterior->proximo = pnovo;
        paux->anterior = pnovo;
     }
     return 1
    }
    else {
      return 0;}
}
```

Inserir Item (por posição)

- Diferente da implementação de lista simplemente ligada, o ponteiro auxiliar para inserção aponta para a posição que será inserida, não a anterior

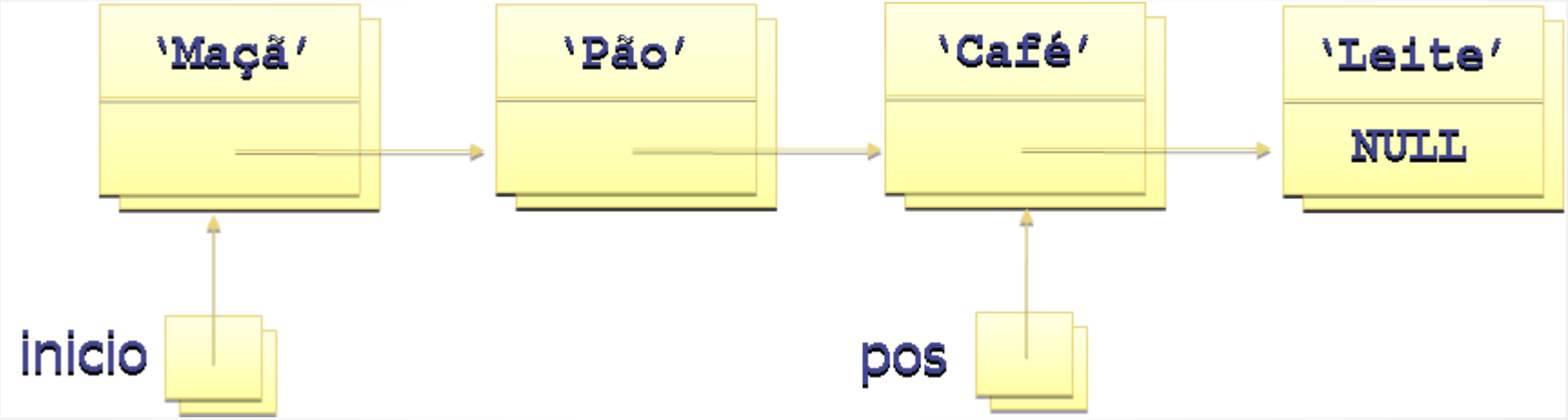
Inserir Item (por posição)

- Diferente da implementação simplemente ligada, o ponteiro auxiliar para inserção aponta para a posição que será inserida, não a anterior

- O ponteiro para fim precisa ser ajustado?

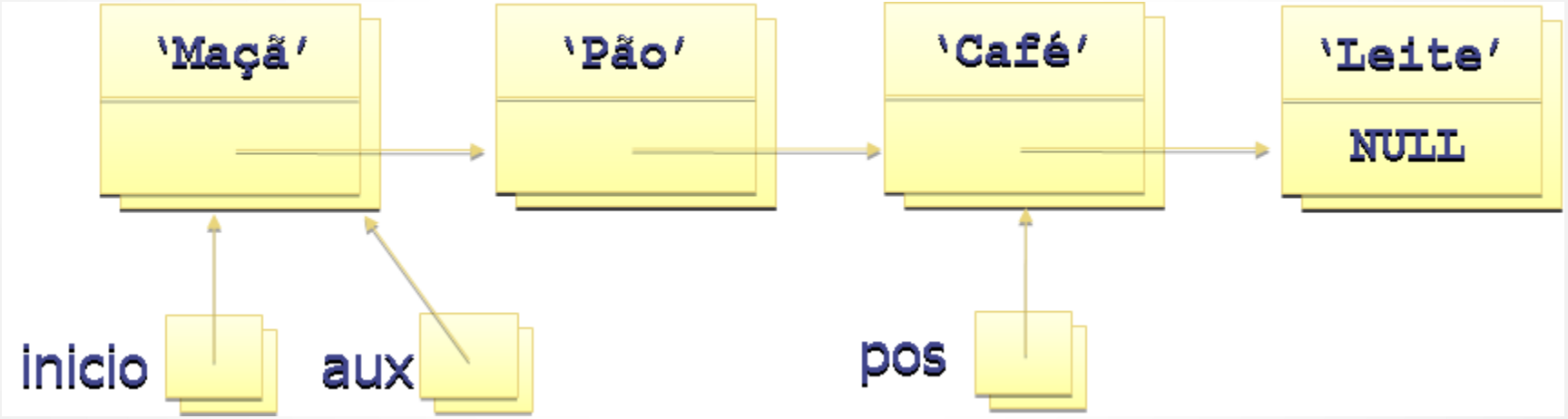
Remover Item (por posição)

- Na lista ligada simples, a operação de remoção dado uma posição envolve encontrar o nó anterior ao que será removido



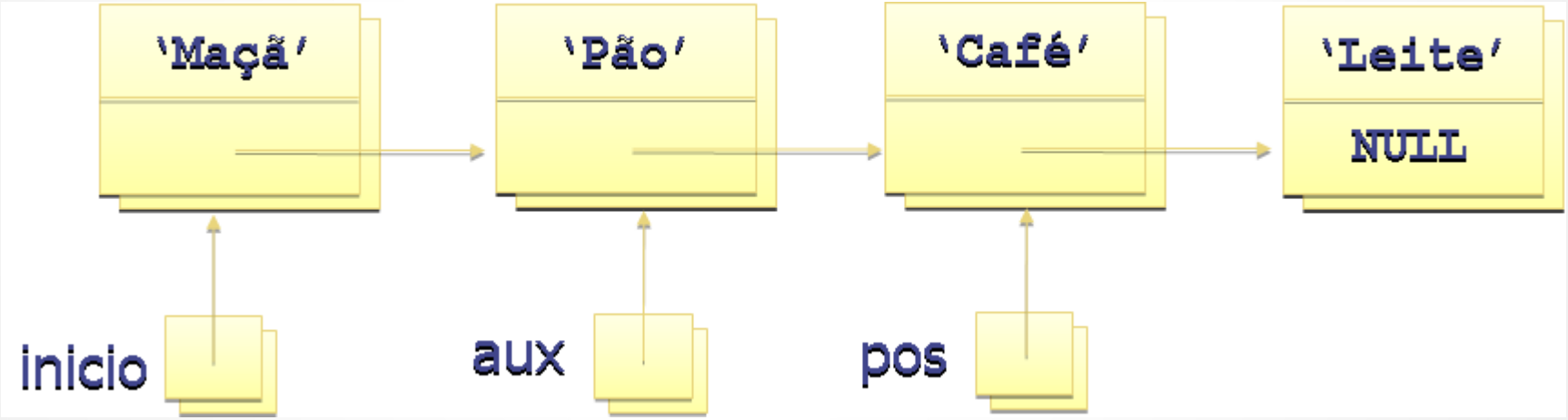
Remover Item (por posição)

- Na lista ligada simples, a operação de remoção dado uma posição envolve encontrar o nó anterior ao que será removido



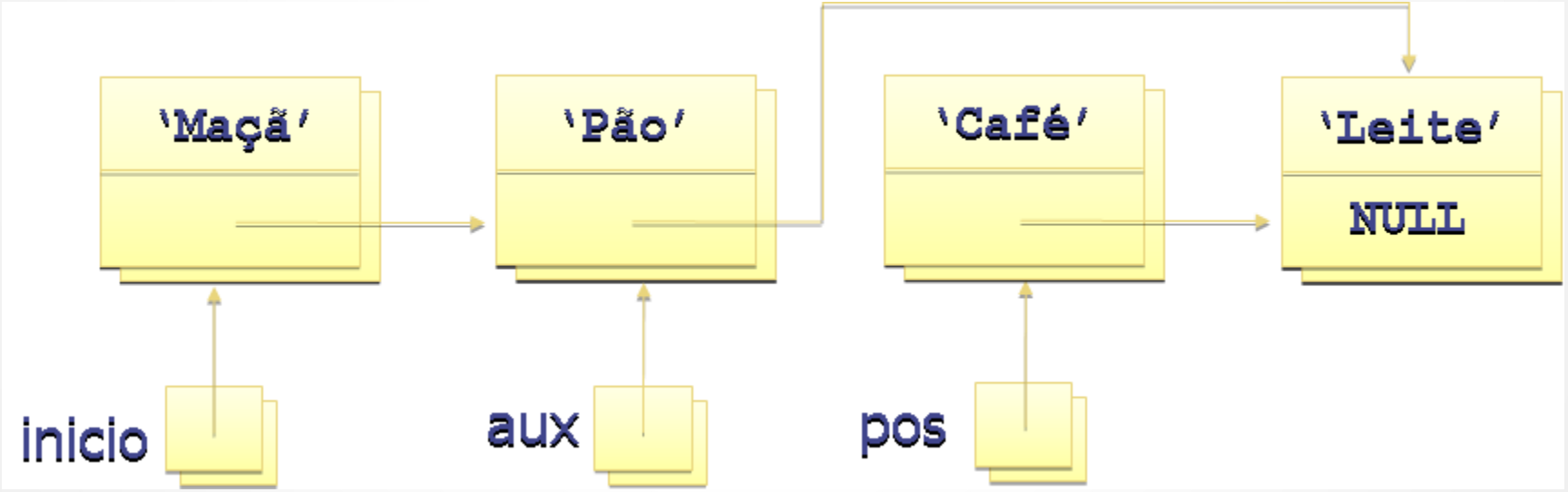
Remover Item (por posição)

- Na lista ligada simples, a operação de remoção dado uma posição envolve encontrar o nó anterior ao que será removido



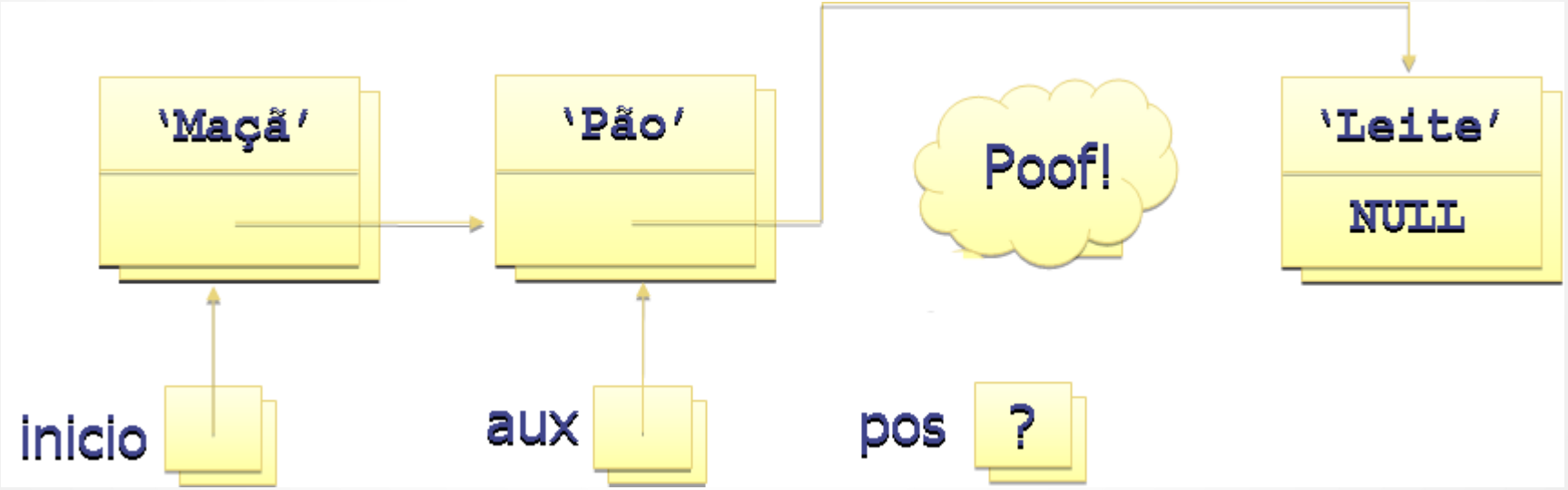
Remover Item (por posição)

- Na lista ligada simples, a operação de remoção dado uma posição envolve encontrar o nó anterior ao que será removido



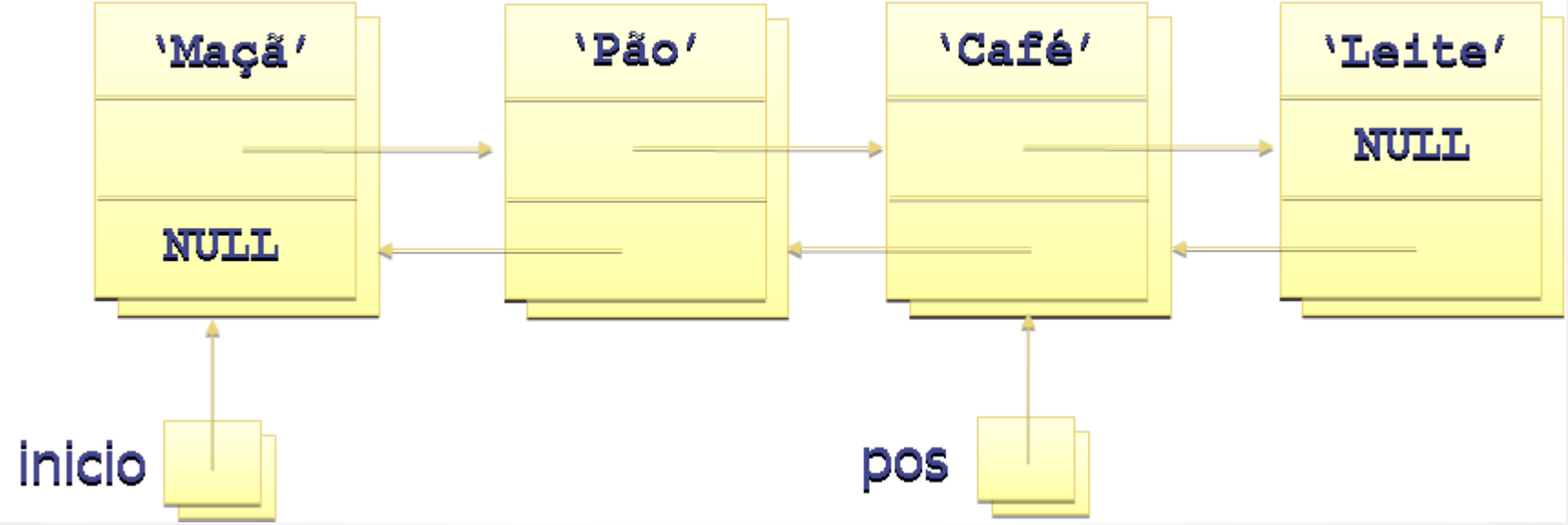
Remover Item (por posição)

- Na lista ligada simples, a operação de remoção dado uma posição envolve encontrar o nó anterior ao que será removido



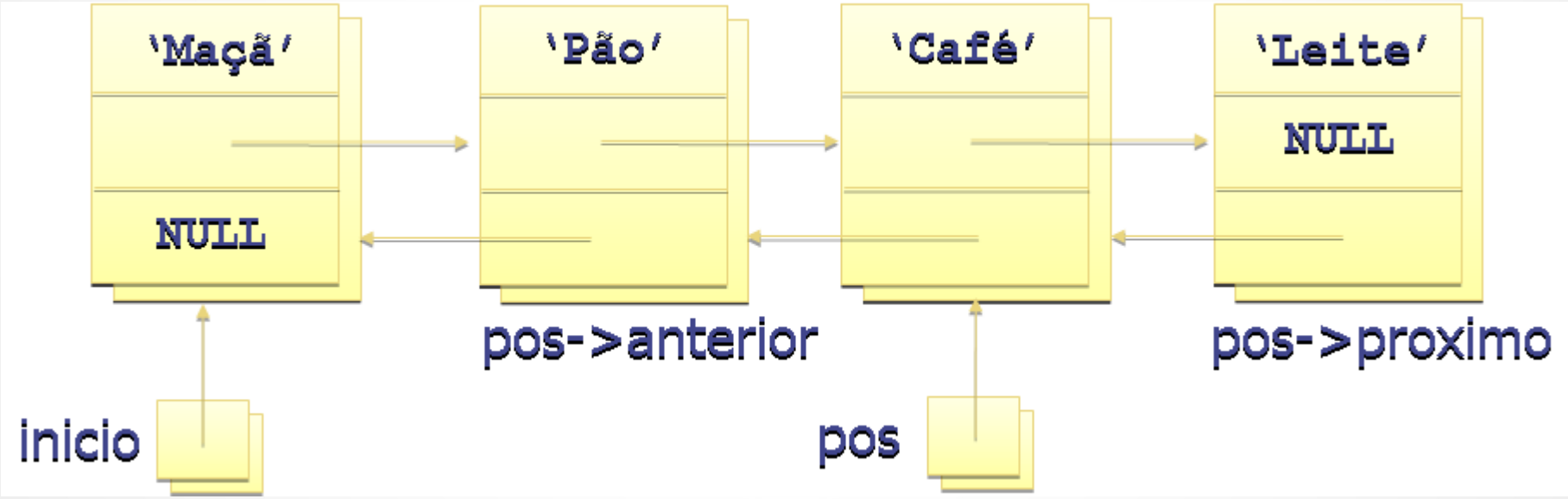
Remover Item (por posição)

- Na lista duplamente ligada, a operação de remoção dado uma posição envolve encontrar manipular os ponteiros



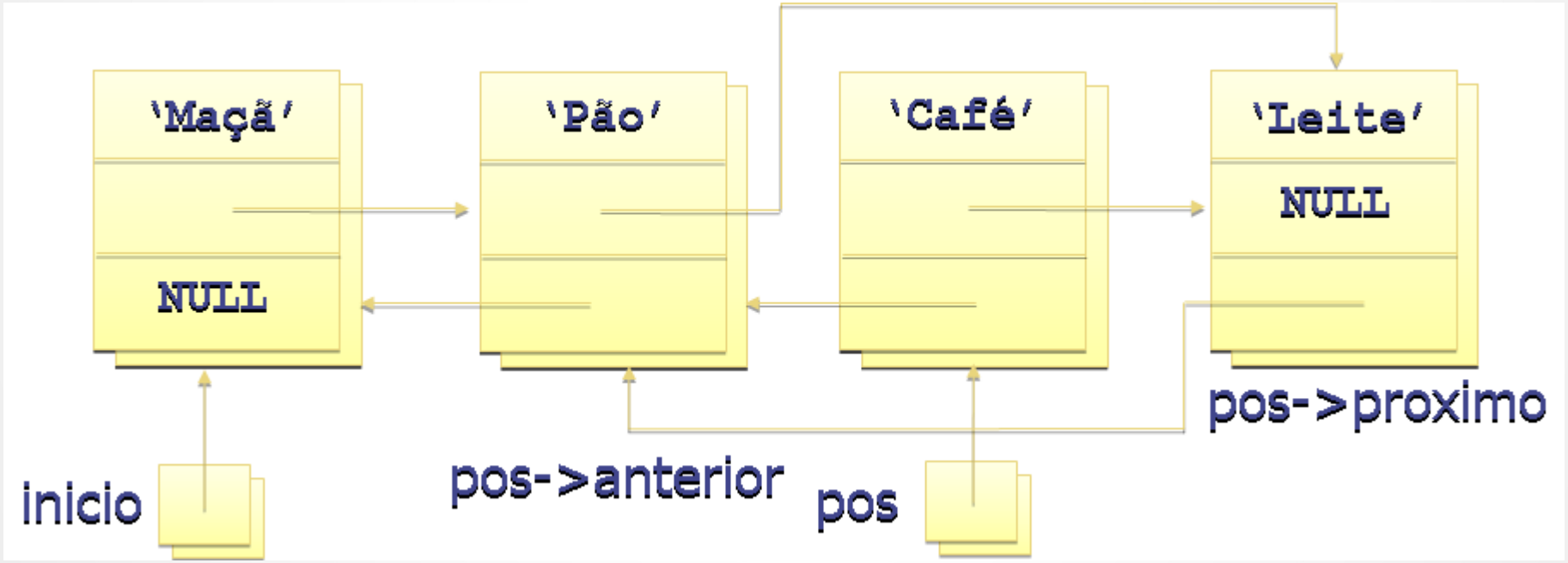
Remover Item (por posição)

- Na lista duplamente ligada, a operação de remoção dado uma posição envolve apenas manipular os ponterios



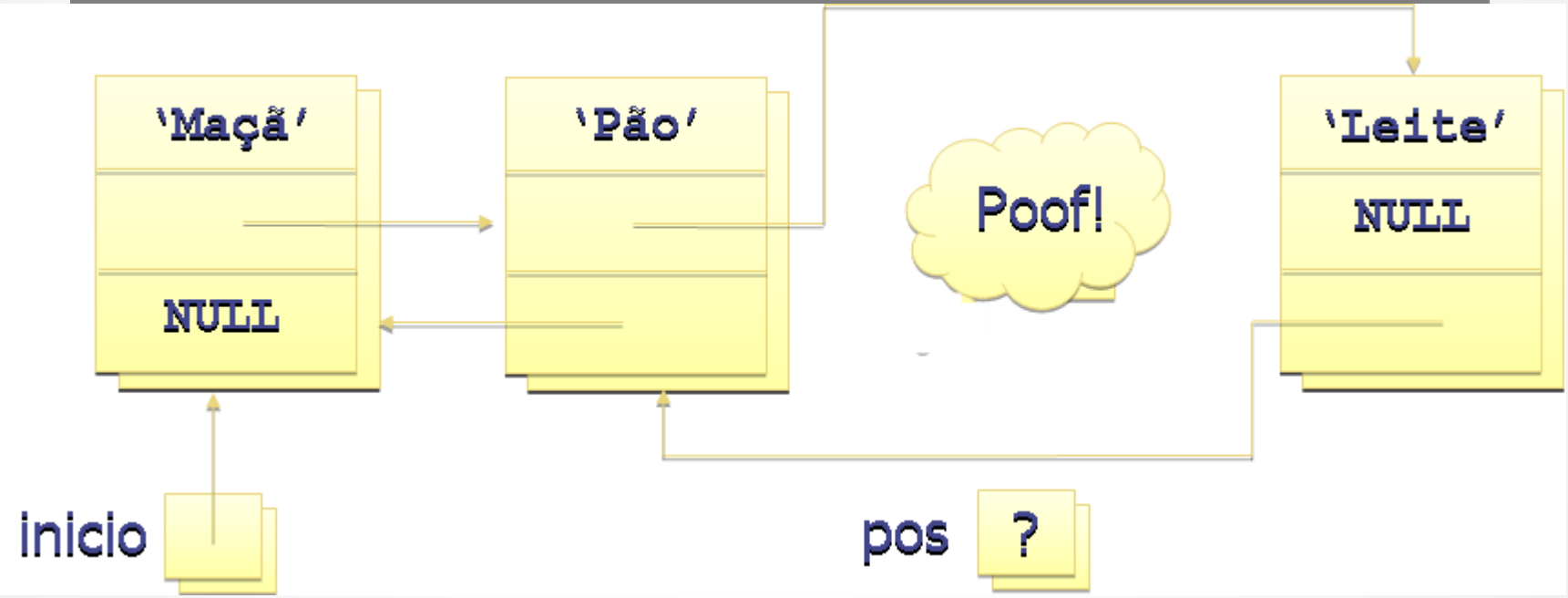
Remover Item (por posição)

- Na lista duplamente ligada, a operação de remoção dado uma posição envolve apenas manipular os ponterios



Remover Item (por posição)

- Na lista duplamente ligada, a operação de remoção dado uma posição envolve apenas manipular os ponteiros



```
int remover_posicao (Lista_Duplamente_Ligada *lista, int pos){
   if (!vazia(lista)) {//verifica se a lista não esta vazia
       int i;
       tNO *paux = lista->inicio
       //encontra a posição de remoção
       for (i=0;i<pos;i++) {
            if (paux != lista->fim) {
                paux = paux->proximo;
            }else { //posição fora da lista
                return 0;
            }
        }
        if (paux == lista->inicio ) {//remove o primeiro item
            lista->inicio = paux->proximo;
        } else {//remove item no meio da lista
            paux->anterior->proximo = paux->proximo;
        }
        if (paux == lista->fim) {//remove o ultimo item
           lista->fim = paux->anterior;
        }else {//remove item no meio da lista
           paux->proximo-anterior = paux->anterior;
        }
       free(paux); //remove o item da memoria
       return 1;
     }
    return 0;
  }
```

Exercício

- Se a lista for duplamente encadeada circular, as exceções no momento da remoção e inserção por posição são evitadas

Exercício

- Se a lista for duplamente encadeada circular, as exceções no momento da remoção e inserção por posição são evitadas

- Se a lista apresenta um nó cabeça (sentinela), a busca pode ser melhorada

Exercício

- Se a lista for duplamente encadeada circular, as exceções no momento da remoção e inserção por posição são evitadas
- Se a lista apresenta um nó cabeça (sentinela), a busca pode ser melhorada
- Implemente todas as operações do TAD lista (apresentadas anteriormente) usando uma lista circular duplamente encadeada com nó cabeça