Listas Duplamente Ligadas

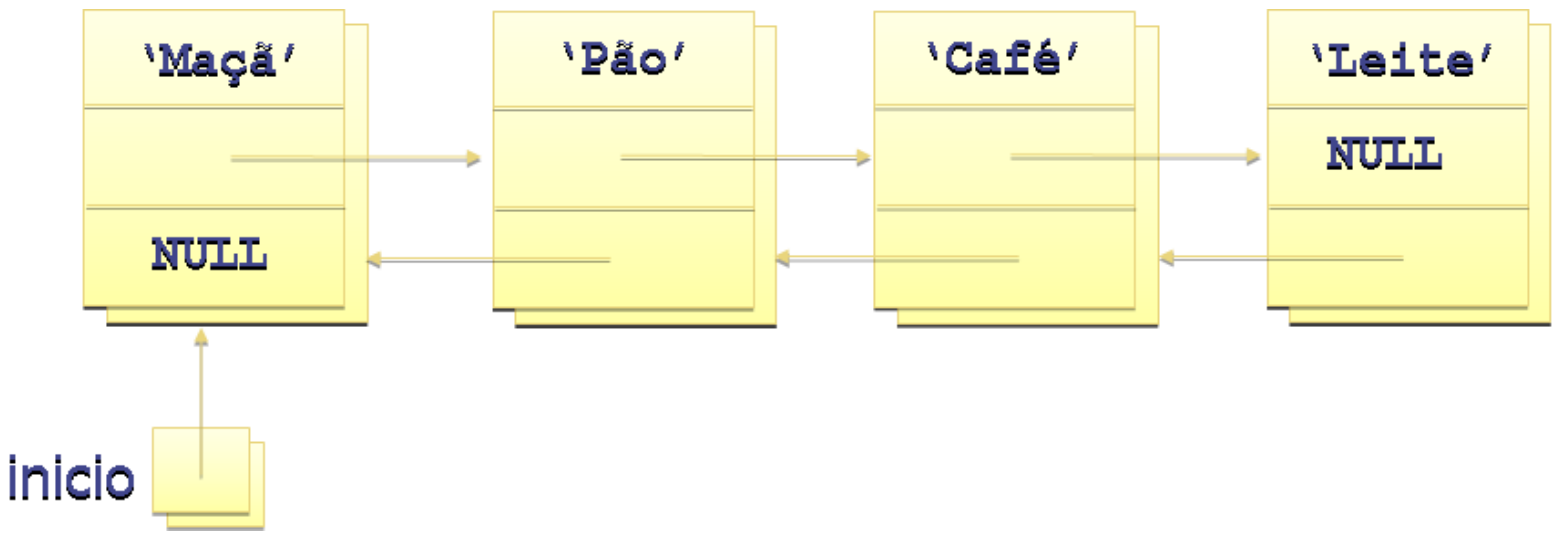
Estruturas de Dados I

Departamento de Computação

Universidade Federal de São Carlos (UFSCar)

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
- 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

Listas Duplamente Ligadas

```
1  typedef struct {
2      int chave;
3      int valor;
4  } ITEM;
5
6  typedef struct NO {
7      ITEM item;
8      struct NO *proximo;
9      struct NO *anterior;
10 } NO;
11
12 typedef struct {
13     NO *inicio;
14     NO *fim;
15 } LISTA_DUPLAMENTE_LIGADA;
```

TAD Listas 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)
- Contar número de itens

Operações Básicas

- As operações de criar e apagar a lista são simples

```
1  void criar(LISTA_DUPLAMENTE_LIGADA *lista) {
2    lista->inicio = NULL;
3    lista->fim = NULL;
4  }
5
6  void apagar_lista(LISTA_DUPLAMENTE_LIGADA *lista) {
7    if (!vazia(lista)) {
8      NO *paux = lista->inicio;
9
10     while (paux != NULL) {
11       NO *prem = paux;
12       paux = paux->proximo;
13       free(prem);
14     }
15   }
16
17   lista->inicio = NULL;
18   lista->fim = NULL;
19 }
```

Inserir Item (Primeira Posição)

```
1  int inserir_inicio(LISTA_DUPLAMENTE_LIGADA *lista, ITEM *item) {
2    NO *pnovo = (NO *)malloc(sizeof(NO)); //crio um novo nó
3
4    if (pnovo != NULL) { //verifica se existe memória disponível
5      //preencho os dados
6      pnovo->item = *item;
7      pnovo->proximo = lista->inicio;
8      pnovo->anterior = NULL;
9
10     if (lista->inicio != NULL) {
11       lista->inicio->anterior = pnovo;
12     } else {
13       lista->fim = pnovo; //ajusta ponteiro para fim
14     }
15
16     lista->inicio = pnovo; //ajusta ponteiro início
17
18     return 1;
19   } else {
20     return 0;
21   }
22 }
```

Inserir Item (Última Posição)

```
1  int inserir_fim(LISTA_DUPLAMENTE_LIGADA *lista, ITEM *item) {
2    NO *pnovo = (NO *)malloc(sizeof(NO)); //crio um novo nó
3
4    if (pnovo != NULL) { //verifica se existe memória disponível
5      //preencho os dados
6      pnovo->item = *item;
7      pnovo->proximo = NULL;
8      pnovo->anterior = lista->fim;
9
10     if (lista->fim != NULL) {
11       lista->fim->proximo = pnovo;
12     } else {
13       lista->inicio = pnovo; //ajusta ponteiro para início
14     }
15
16     lista->fim = pnovo; //ajusta ponteiro fim
17
18     return 1;
19   } else {
20     return 0;
21   }
22 }
```

Remover Item (Primeira Posição)

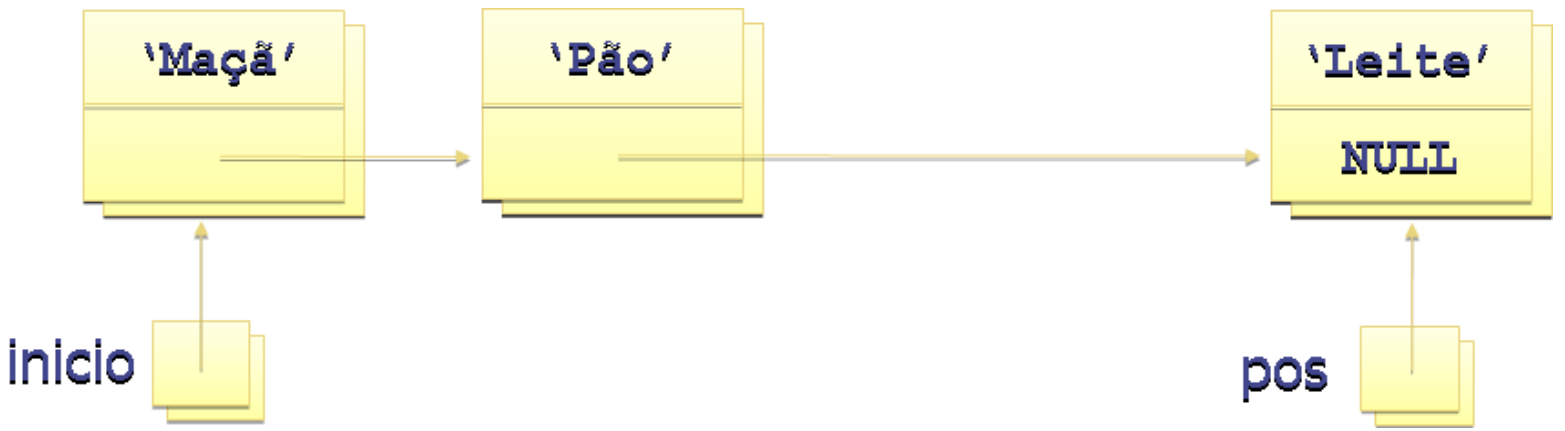
```
1  int remover_inicio(LISTA_DUPLAMENTE_LIGADA *lista) {
2    if (!vazia(lista)) { //verifica se a lista não está vazia
3      NO *paux = lista->inicio;
4
5      if (lista->inicio == lista->fim) { //lista com um elemento
6        lista->inicio = NULL;
7        lista->fim = NULL;
8      } else { //lista com mais de um elemento
9        lista->inicio->proximo->anterior = NULL;
10       lista->inicio = lista->inicio->proximo;
11     }
12
13     free(paux); //removo o nó da memória
14
15     return 1;
16   } else {
17     return 0;
18   }
19 }
```

Remover Item (Última Posição)

```
1  int remover_fim(LISTA_DUPLAMENTE_LIGADA *lista) {
2    if (!vazia(lista)) { //verifica se a lista não está vazia
3      NO *paux = lista->fim;
4
5      if (lista->inicio == lista->fim) { //lista com um elemento
6        lista->inicio = NULL;
7        lista->fim = NULL;
8      } else { //lista com mais de um elemento
9        lista->fim->anterior->proximo = NULL;
10       lista->fim = lista->fim->anterior;
11     }
12
13     free(paux); //removo o nó da memória
14
15     return 1;
16   } else {
17     return 0;
18   }
19 }
```

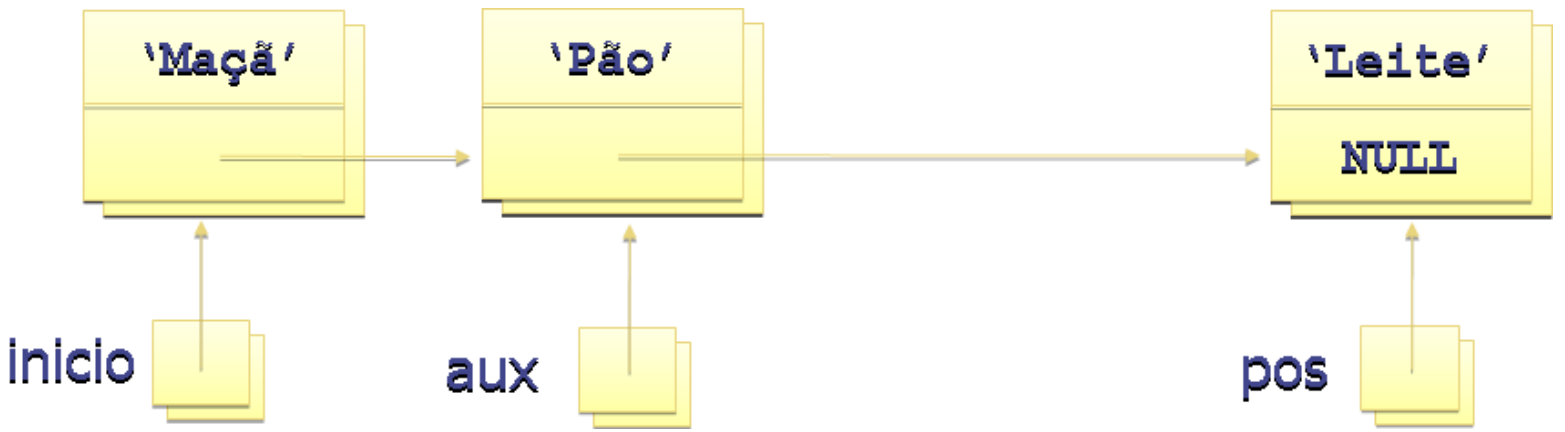
Insere Item (por posição)

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



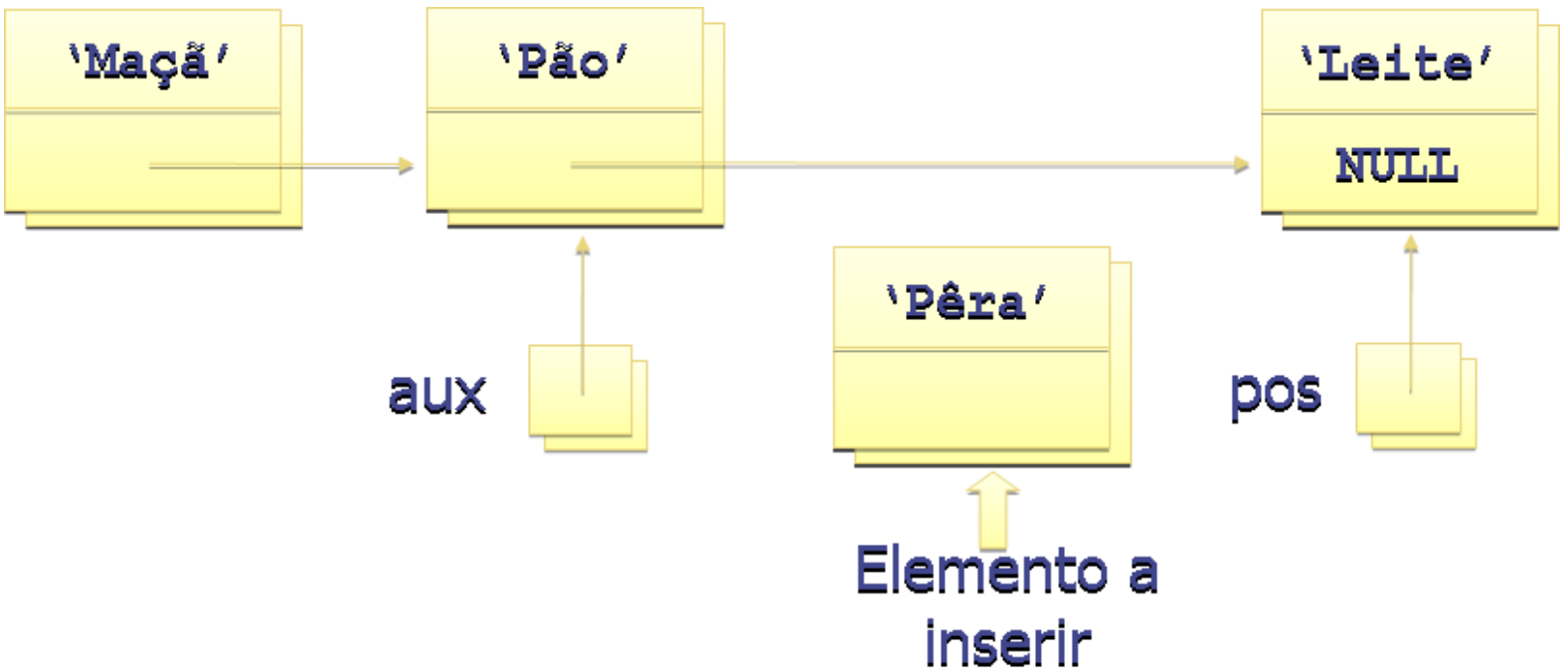
Insere Item (por posição)

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



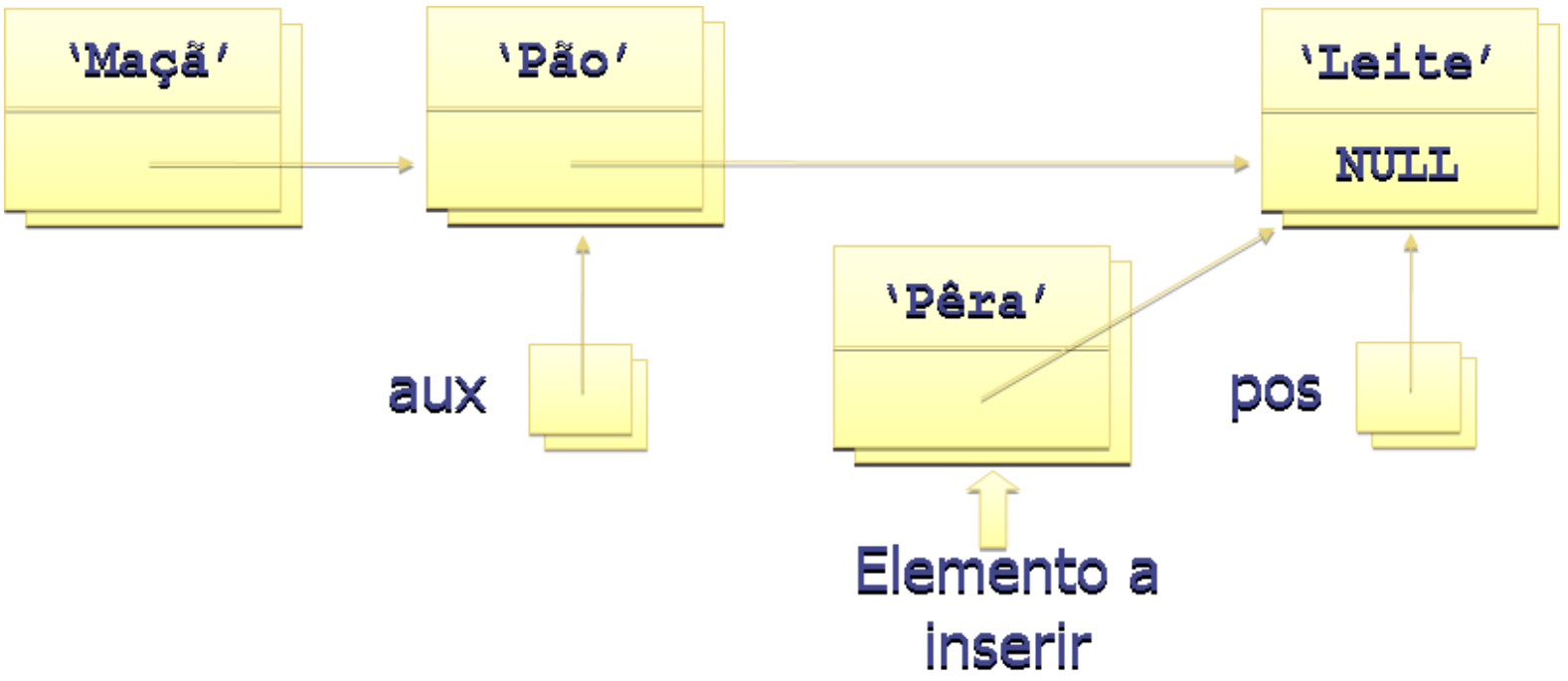
Insere Item (por posição)

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



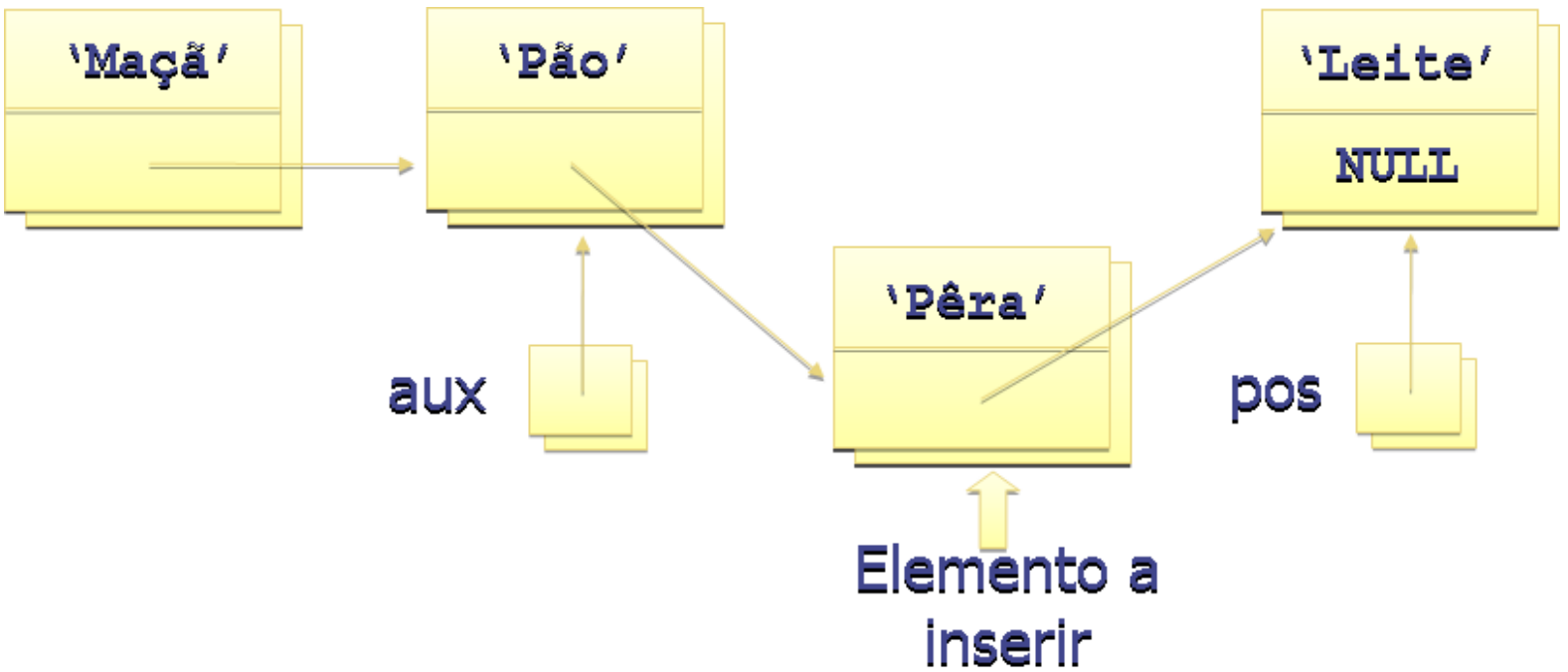
Insere Item (por posição)

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



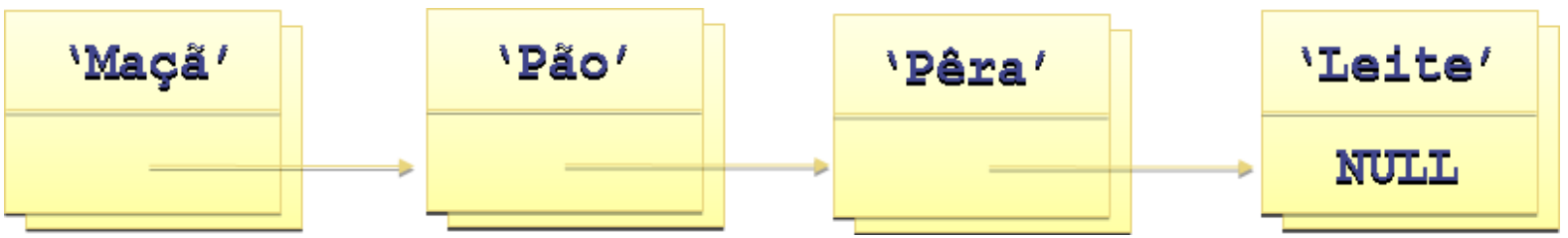
Insere Item (por posição)

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



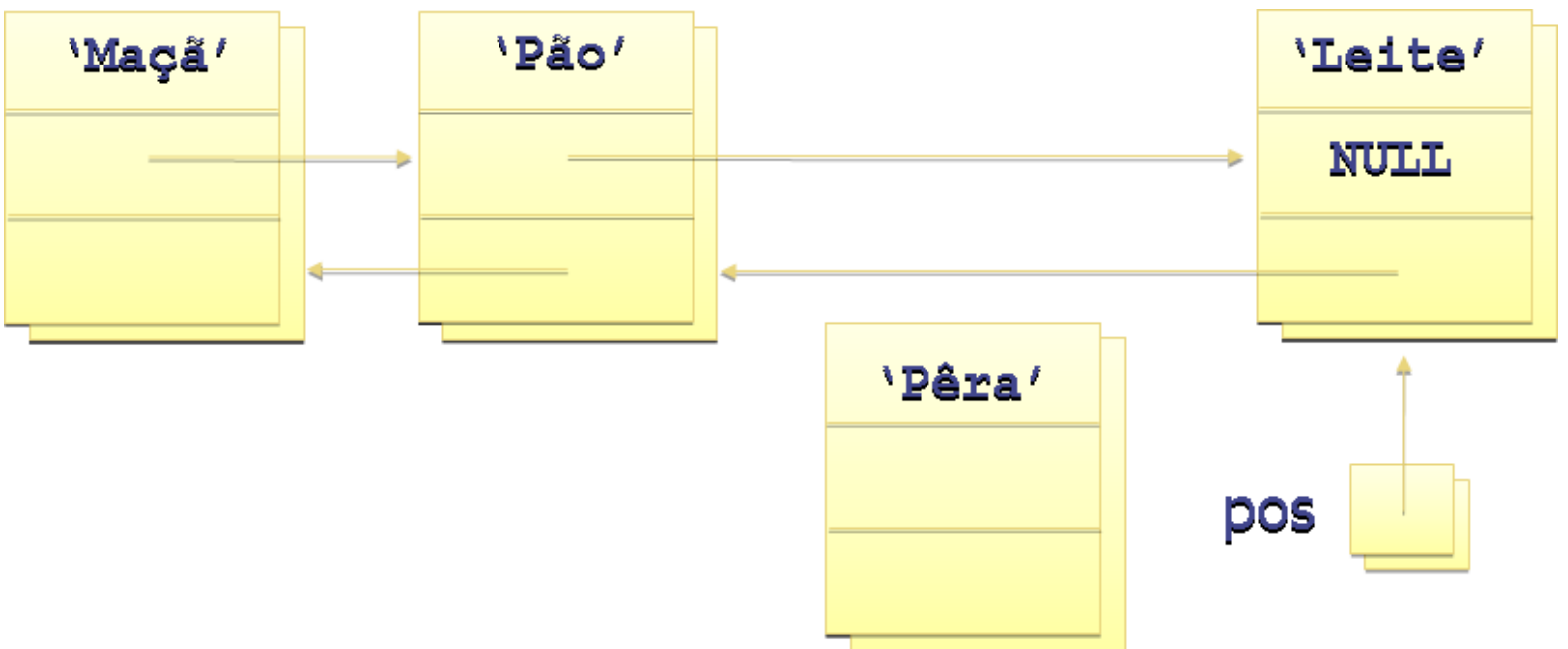
Insere Item (por posição)

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



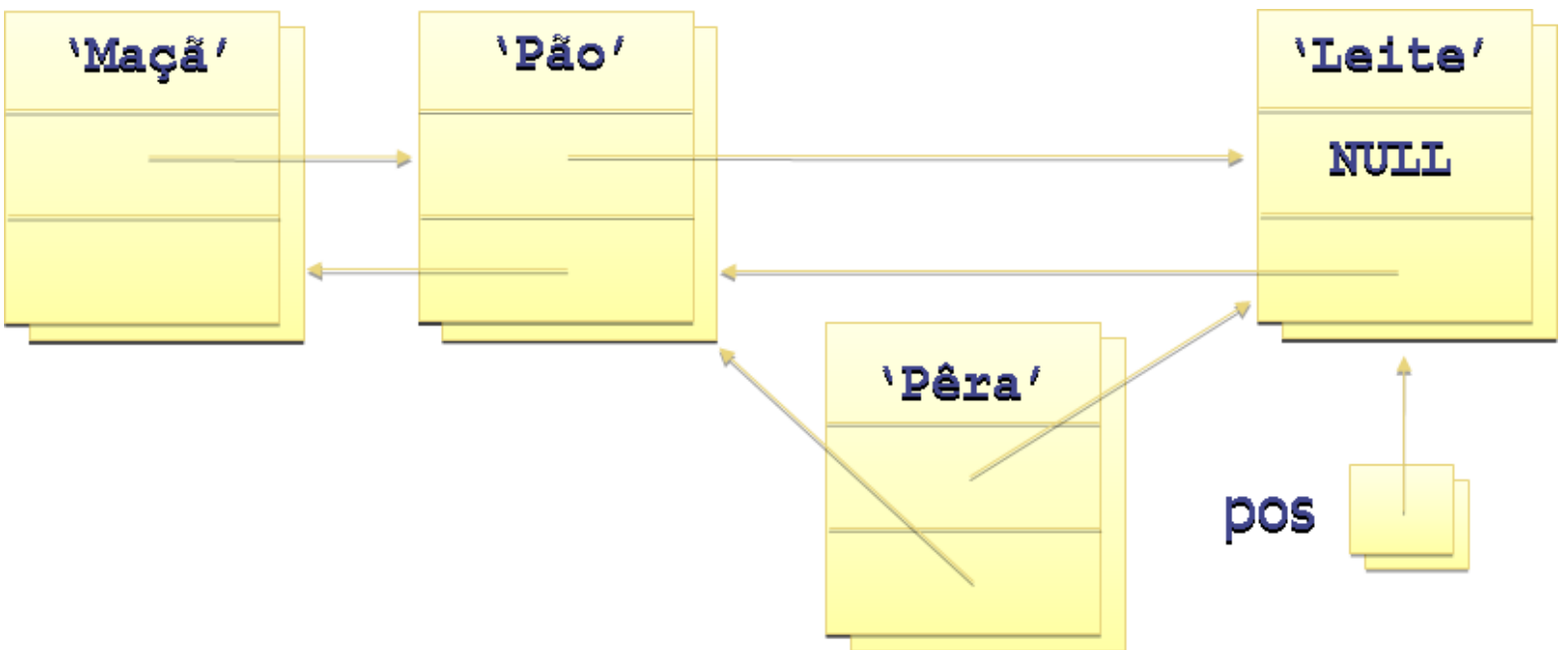
Insere Item (por posição)

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



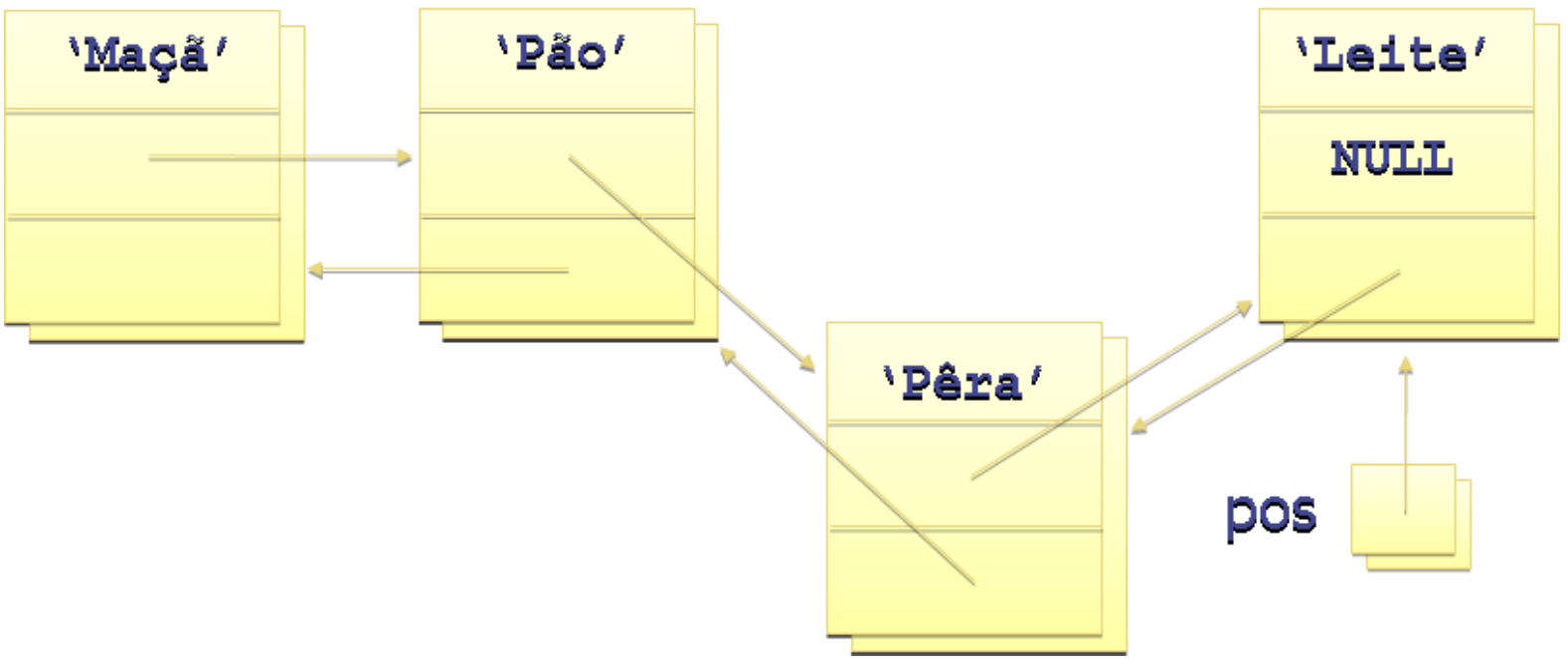
Insere Item (por posição)

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



Insere Item (por posição)

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



Insere Item (por posição)

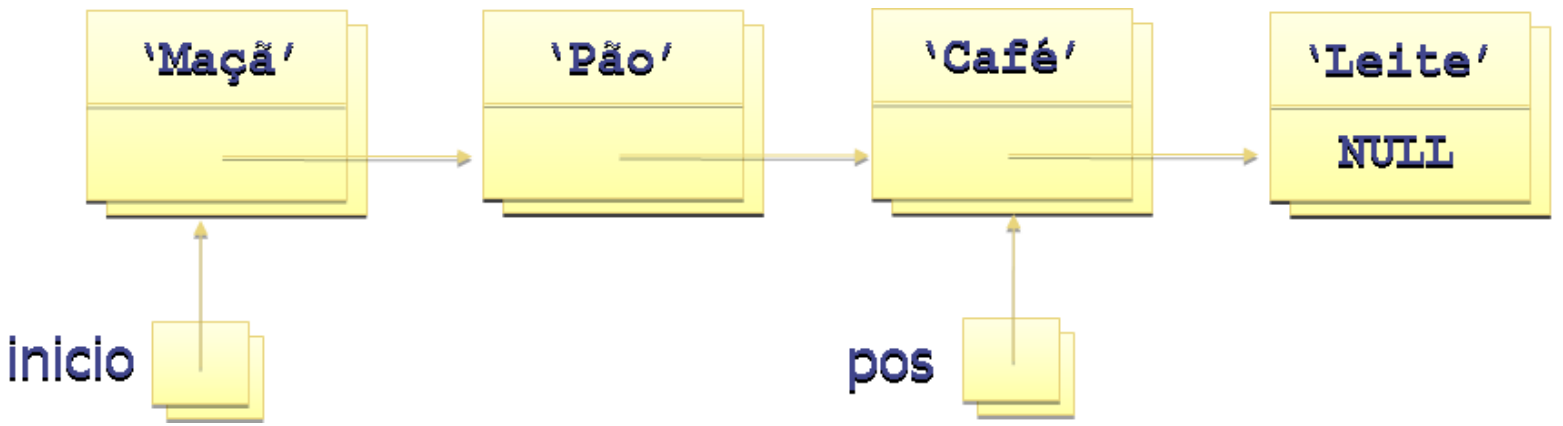
```
1  int inserir_posicao(LISTA_DUPLAMENTE_LIGADA *lista, int pos, ITEM *item) {
2    int i;
3
4    if (!vazia(lista)) { //verifica se a lista não está vazia
5      NO *pnovo = (NO *)malloc(sizeof(NO)); //crio um novo nó
6
7      if (pnovo != NULL) { //verifica se existe memória disponível
8        pnovo->item = *item;
9
10       if (pos == 0) { //adiciona na primeira posição
11         pnovo->proximo = lista->inicio;
12         lista->inicio->anterior = pnovo;
13         lista->inicio = pnovo;
14       } else {
15         NO *paux = lista->inicio;
16
17         //encontra a posição de inserção
18         for (i = 0; i < pos; i++) {
19           if (paux != lista->fim) {
20             paux = paux->proximo;
21           } else {
22             return 0;
23           }
24         }
25
26         //faz as ligações para a inserção do novo elemento
27         pnovo->proximo = paux;
28         pnovo->anterior = paux->anterior;
29         paux->anterior->proximo = pnovo;
30         paux->anterior = pnovo;
31       }
32
33       return 1;
34     }
35   }
36
37   return 0;
38 }
```

Insere Item (por posição)

- Diferente da implementação simplesmente ligada, o ponteiro auxiliar para inserção aponta para a posição que será inserida, não a anterior
- O ponteiro para fim precisa ser ajustado?

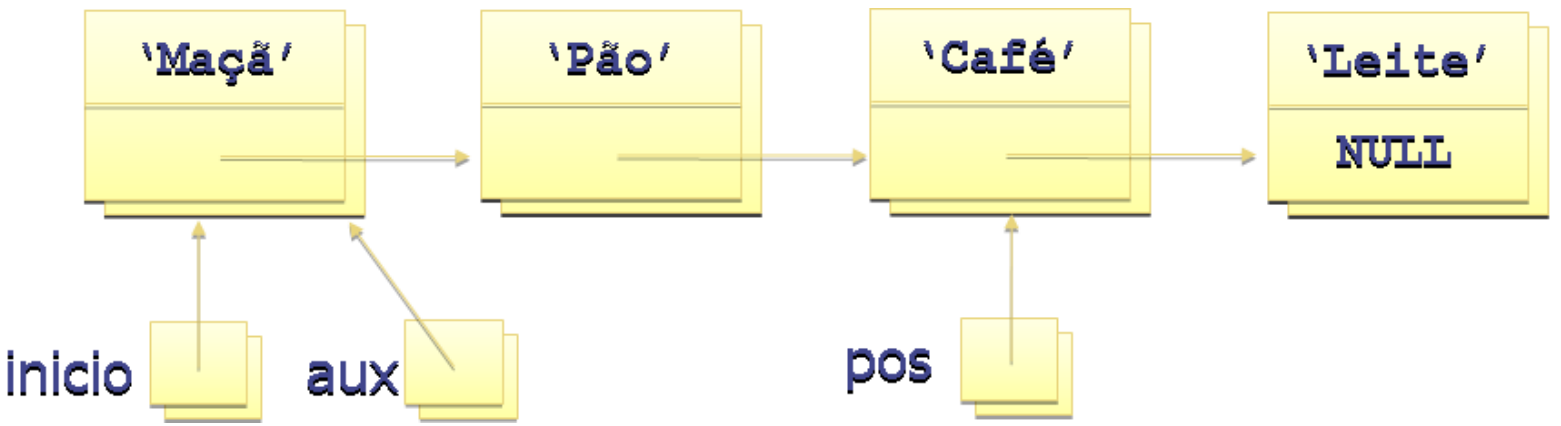
Remover Item (por posição)

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



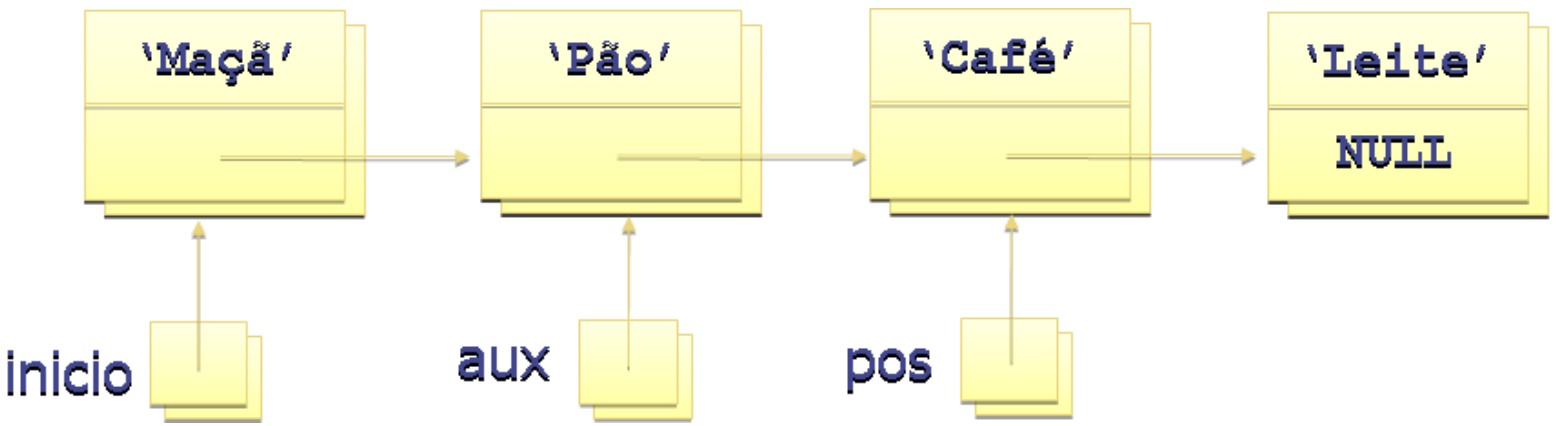
Remover Item (por posição)

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



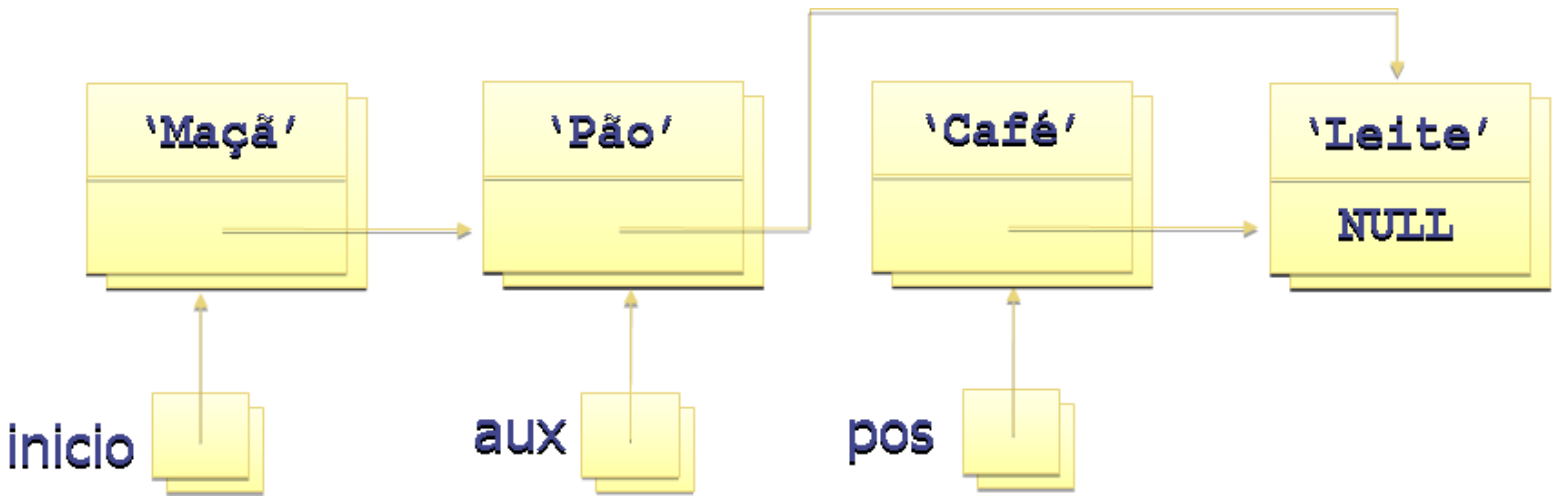
Remover Item (por posição)

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



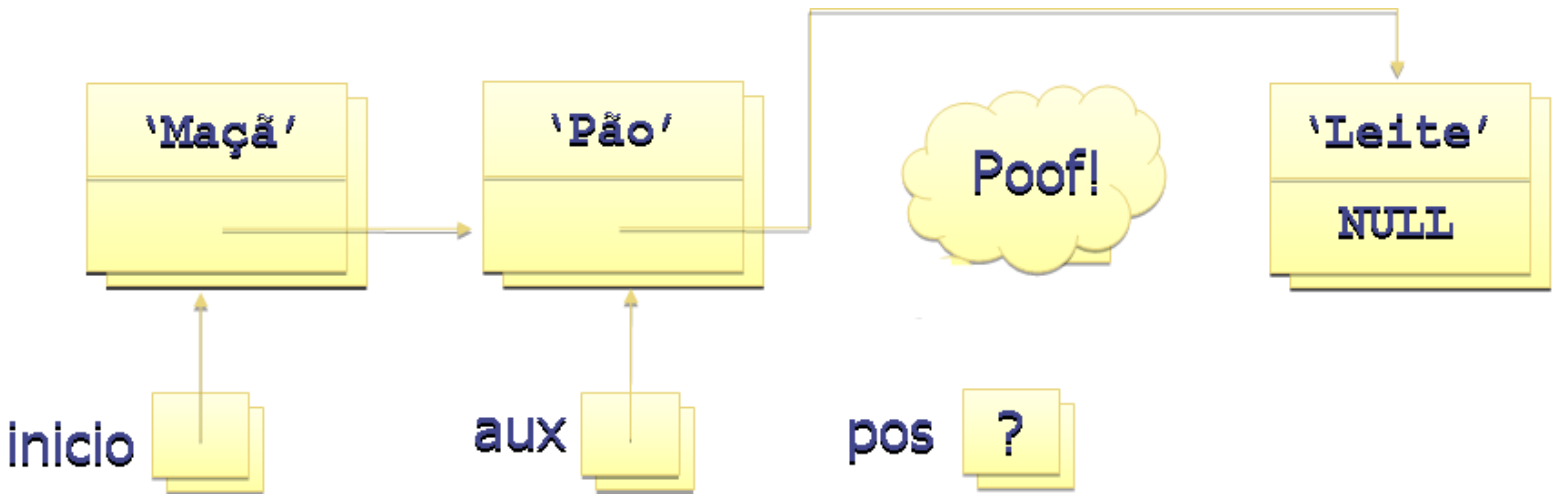
Remover Item (por posição)

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



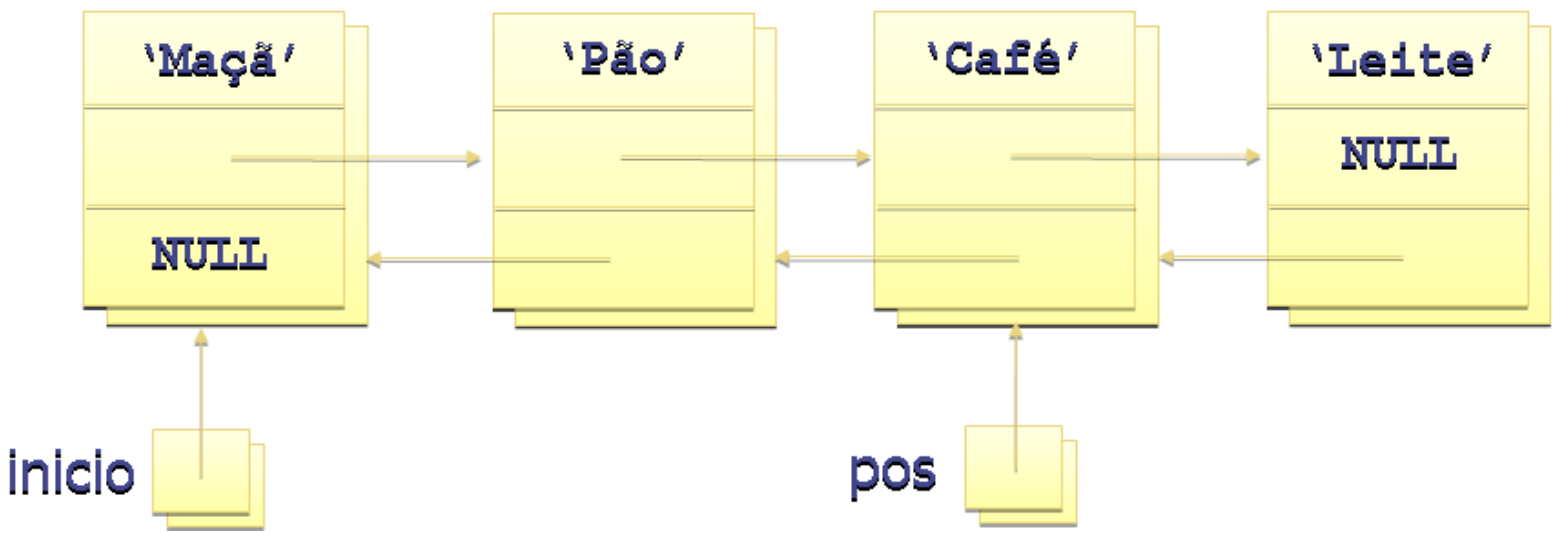
Remover Item (por posição)

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



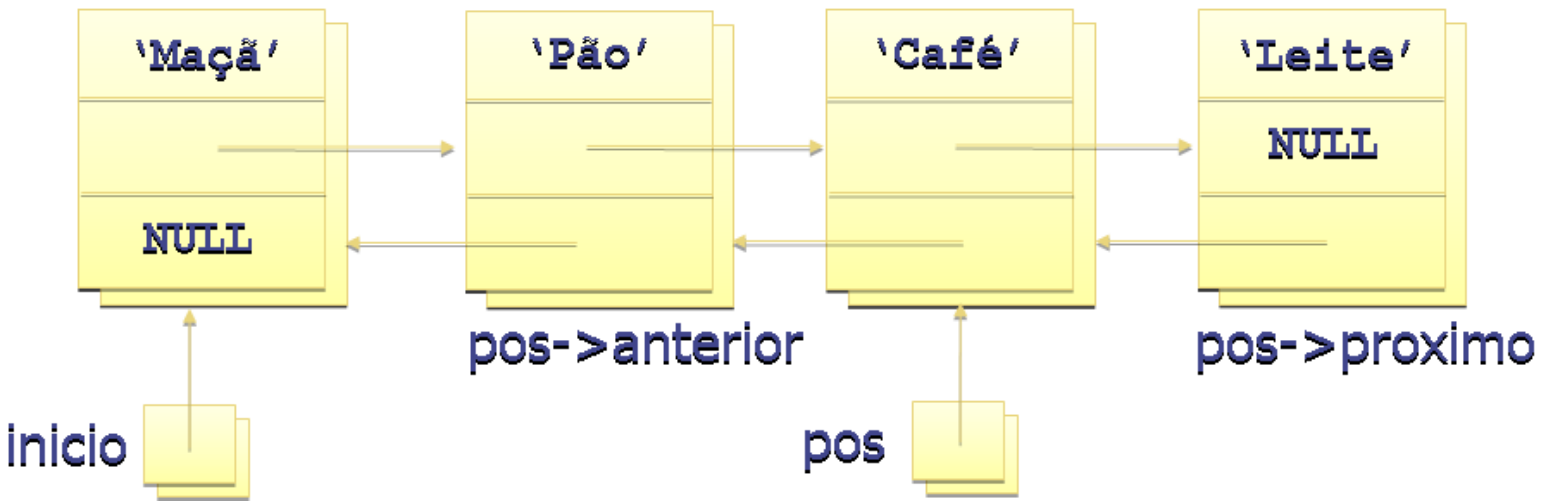
Remover Item (por posição)

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



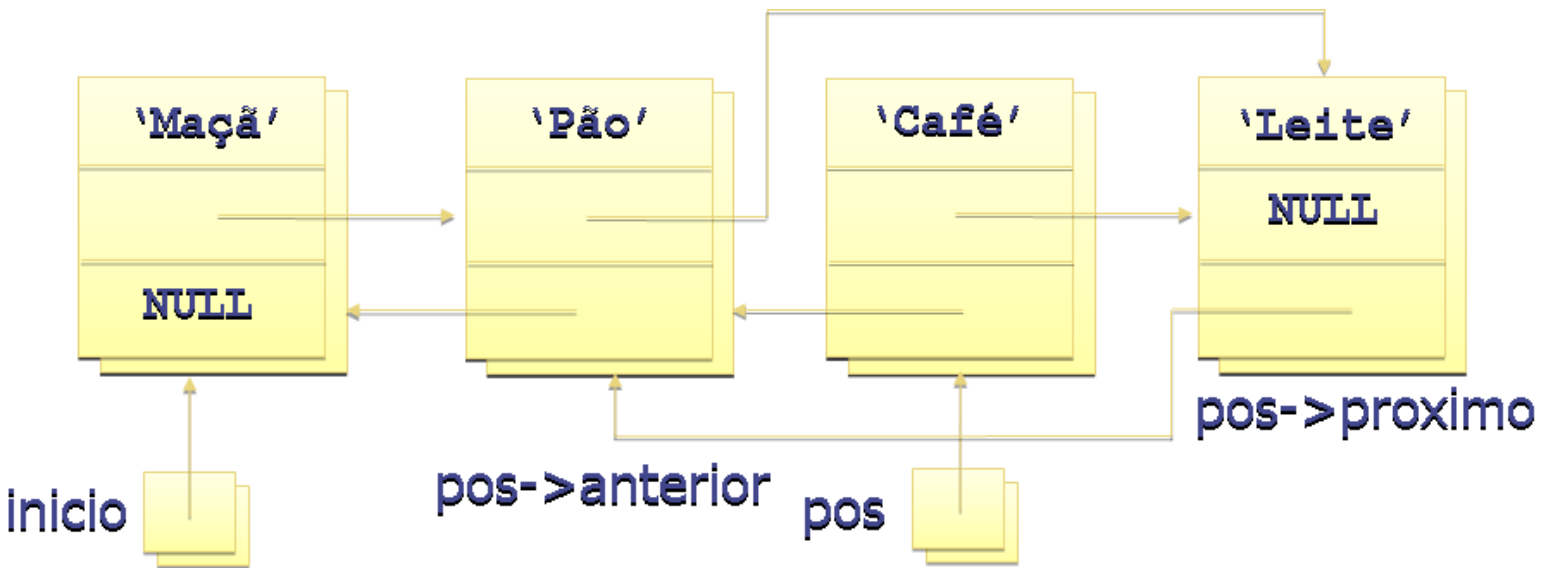
Remover Item (por posição)

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



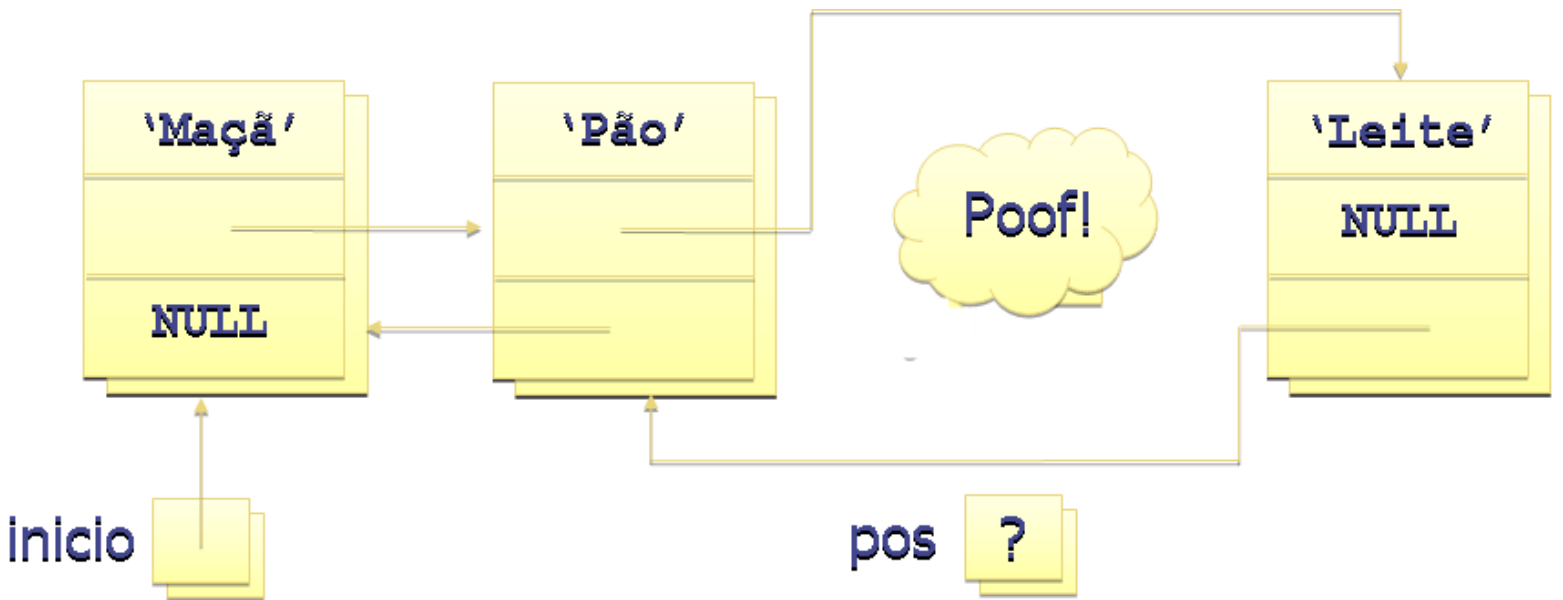
Remover Item (por posição)

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



Remover Item (por posição)

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



Remover Item (por posição)

```
1  int remover_posicao(LISTA_DUPLAMENTE_LIGADA *lista, int pos) {
2    if (!vazia(lista)) { //verifica se a lista está vazia
3      int i;
4      NO *paux = lista->inicio;
5
6      //encontra a posição de remoção
7      for (i = 0; i < pos; i++) {
8        if (paux != lista->fim) {
9          paux = paux->proximo;
10       } else { //posição fora da lista
11         return 0;
12       }
13     }
14
15     if (paux == lista->inicio) { //remove o primeiro item
16       lista->inicio = paux->proximo;
17     } else { //remove item no meio da lista
18       paux->anterior->proximo = paux->proximo;
19     }
20
21     if (paux == lista->fim) { //remove o último item
22       lista->fim = paux->anterior;
23     } else { //remove item no meio da lista
24       paux->proximo->anterior = paux->anterior;
25     }
26
27     free(paux); //remove o item da memória
28
29     return 1;
30   }
31
32   return 0;
33 }
```

Exercício

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