

Listas Ligadas¹

Algoritmos e Programação 2

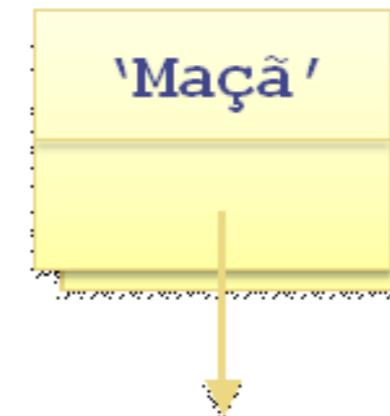
Departamento de Computação

Universidade Federal de São Carlos (UFSCar)

¹Aula baseada nos tópicos de aula e slides criados pelo prof. Fernando Vieira Paulovich (ICMC).

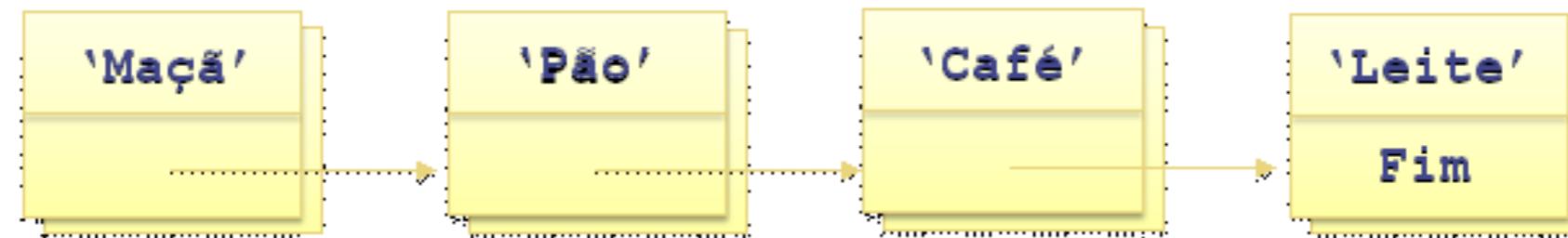
Discussão Intuitiva

- Ponteiros podem ser usados para construir estruturas, tais como listas, a partir de componentes simples chamados **nó**



Discussão Intuitiva

- Um **nó** possui uma seta apontando para fora. Essa seta representa um ponteiro que aponta para outro **nó**, formando uma **lista ligada**

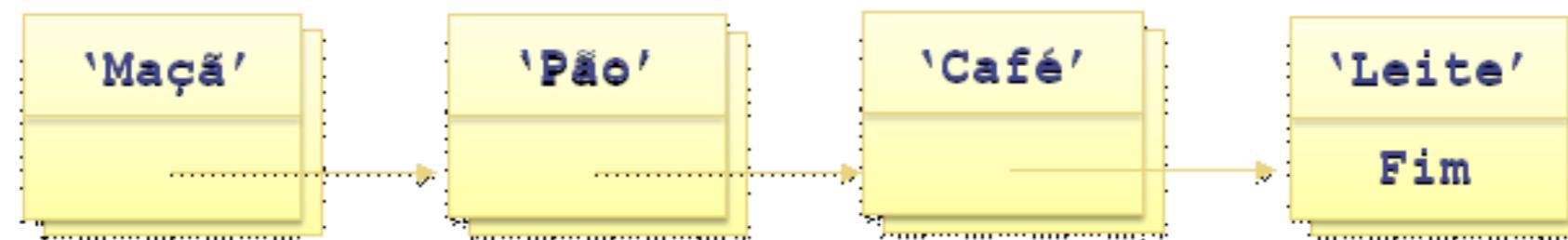


Discussão Intuitiva

- Listas ligadas são úteis pois podem ser utilizadas para implementar o TAD lista. Nesse caso, as operações inserção e remoção no meio da lista podem ser mais eficientes
- Uma segunda vantagem é o fato de não ser necessário informar o número de elementos em tempo de compilação

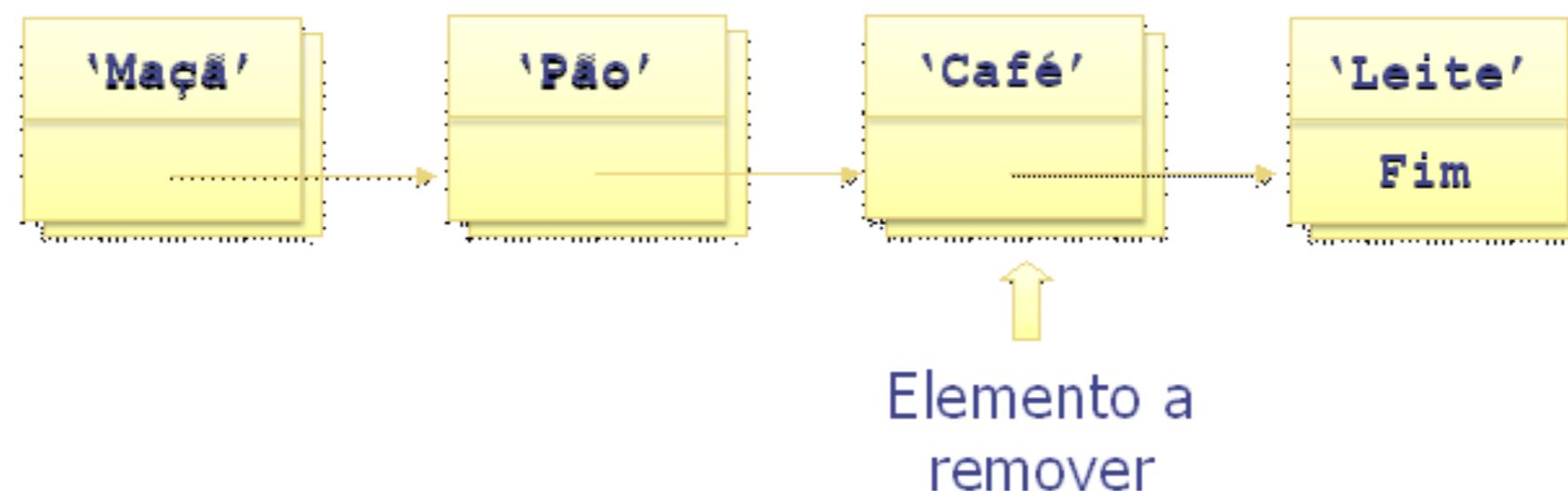
Discussão Intuitiva

- Por exemplo, uma operação de **remoção** pode ser feita da seguinte maneira



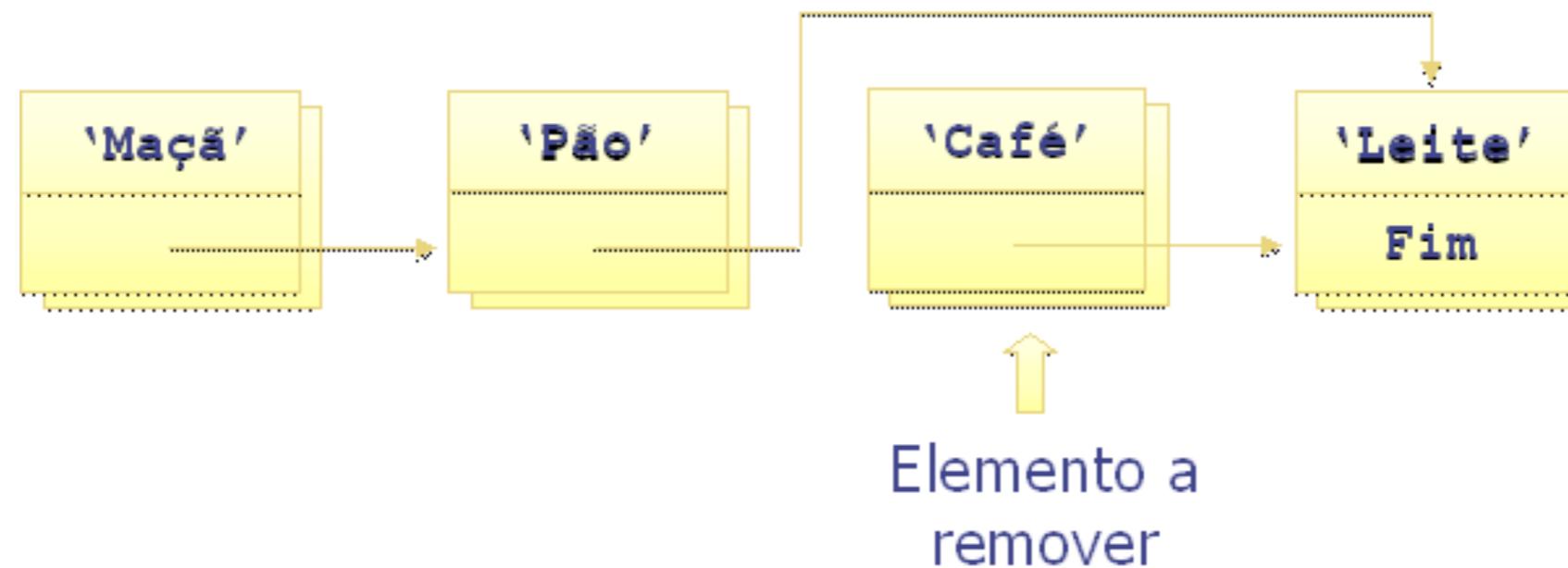
Discussão Intuitiva

- Por exemplo, uma operação de **remoção** pode ser feita da seguinte maneira



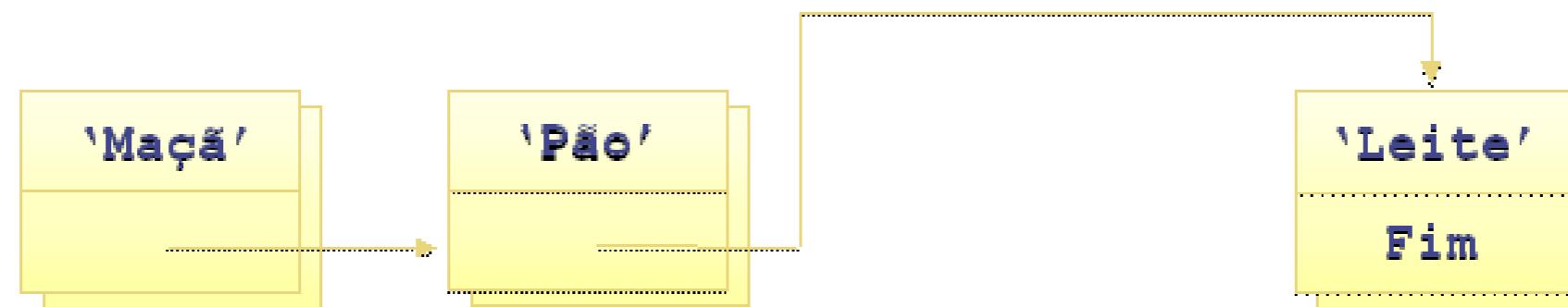
Discussão Intuitiva

- Por exemplo, uma operação de **remoção** pode ser feita da seguinte maneira



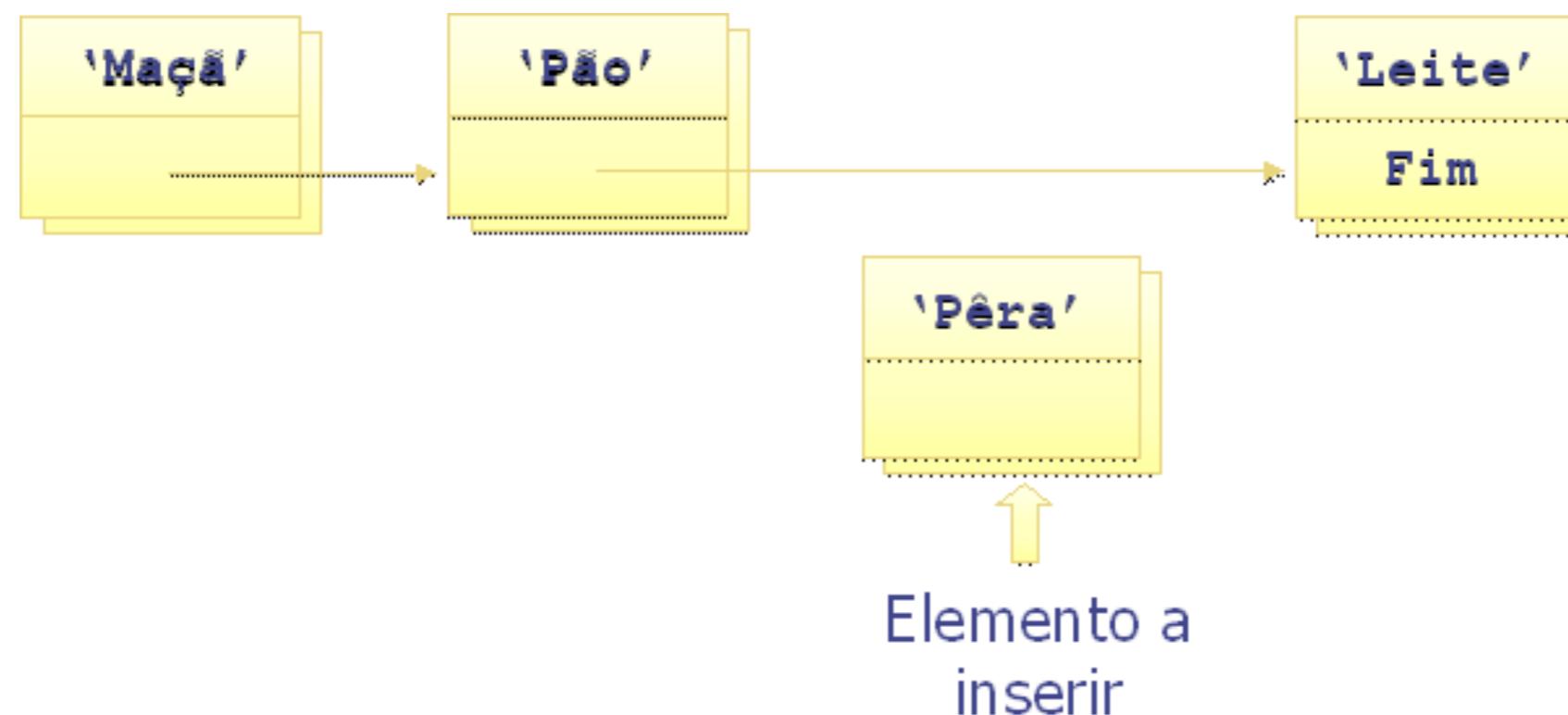
Discussão Intuitiva

- Por exemplo, uma operação de **remoção** pode ser feita da seguinte maneira



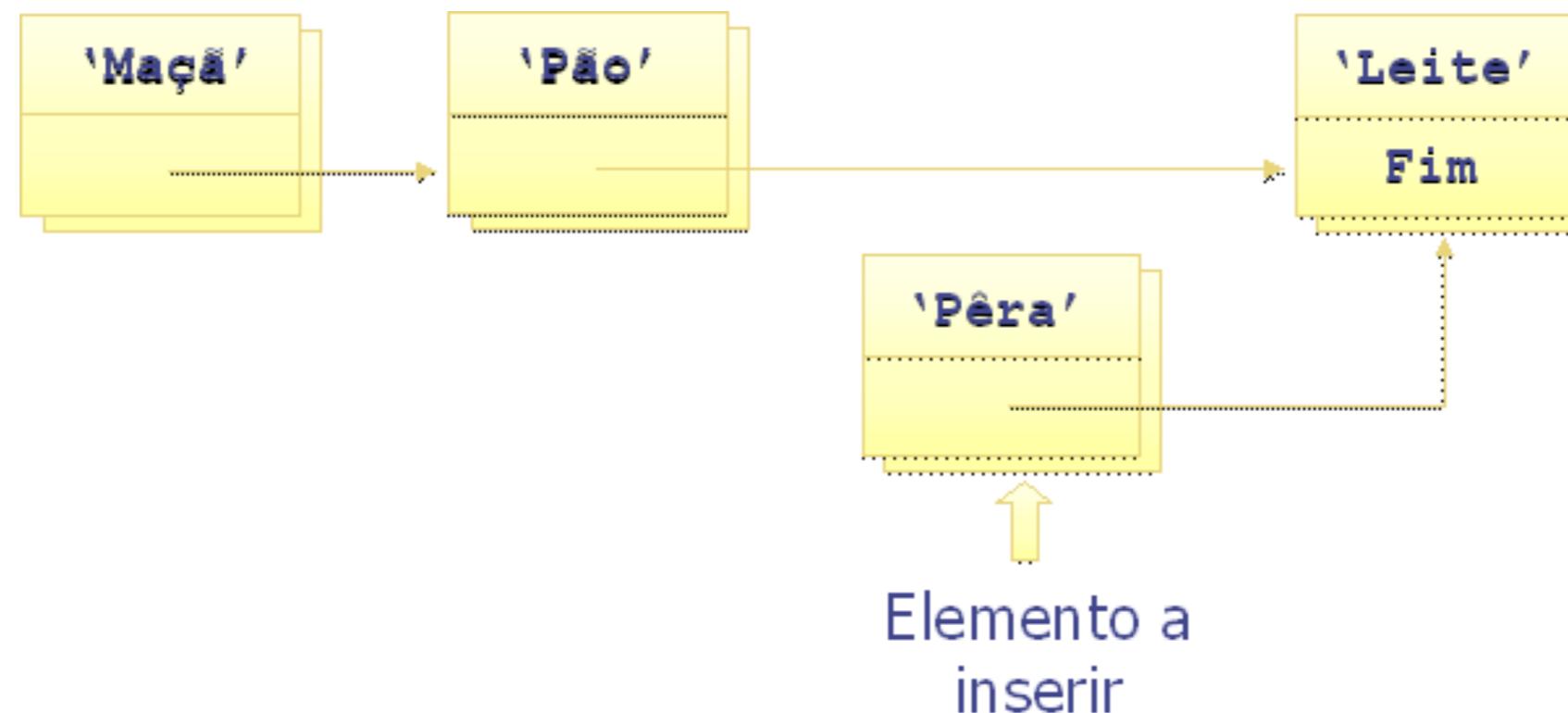
Discussão Intuitiva

- Por exemplo, uma operação de **inserção** pode ser feita da seguinte maneira



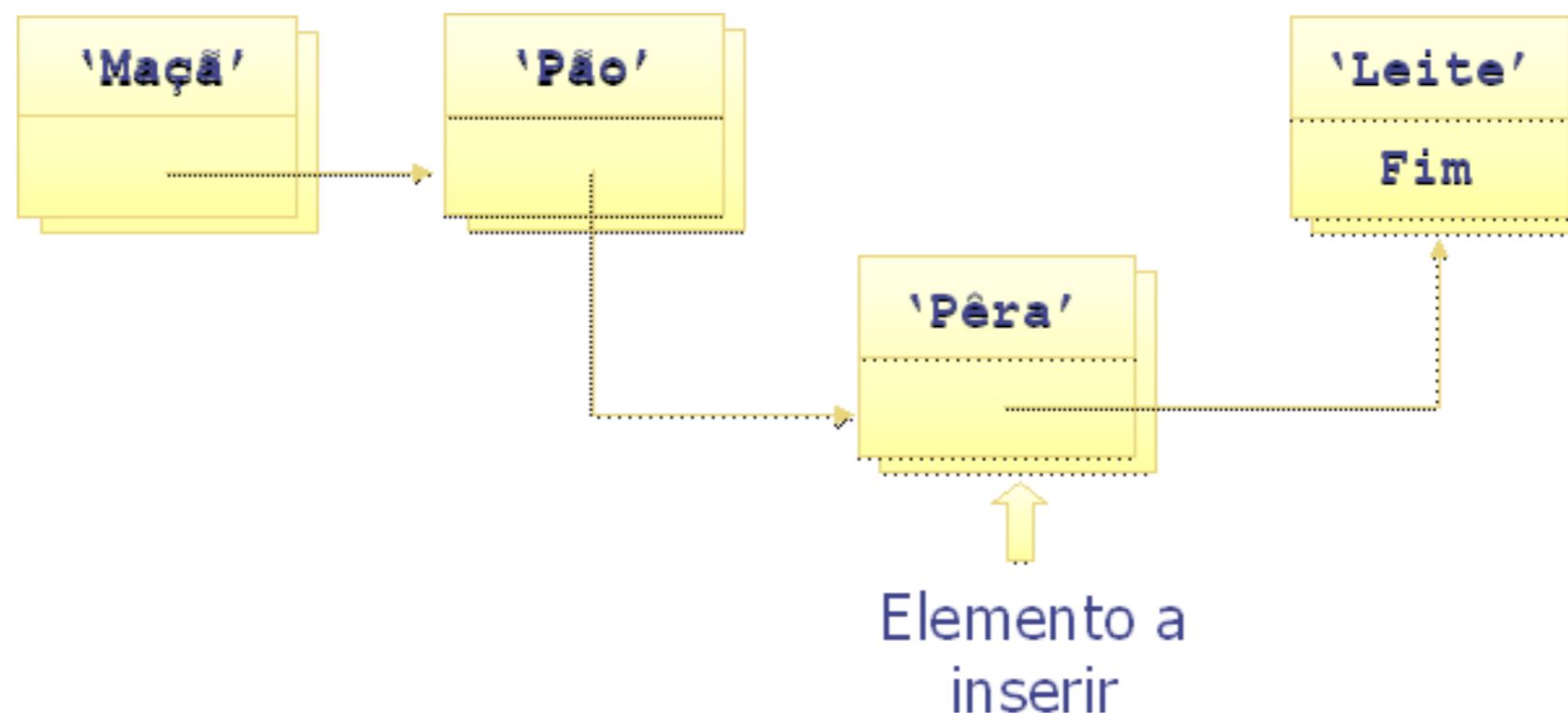
Discussão Intuitiva

- Por exemplo, uma operação de **inserção** pode ser feita da seguinte maneira



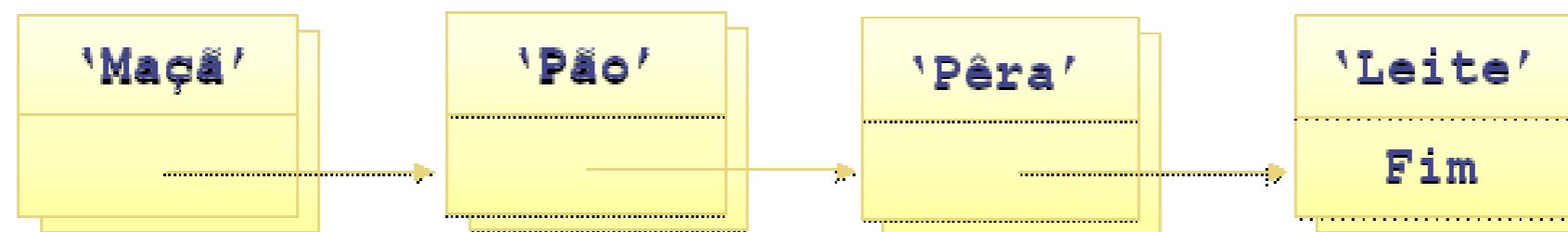
Discussão Intuitiva

- Por exemplo, uma operação de **inserção** pode ser feita da seguinte maneira



Discussão Intuitiva

- Por exemplo, uma operação de **inserção** pode ser feita da seguinte maneira



Relembrando: TAD Listas

Principais operações

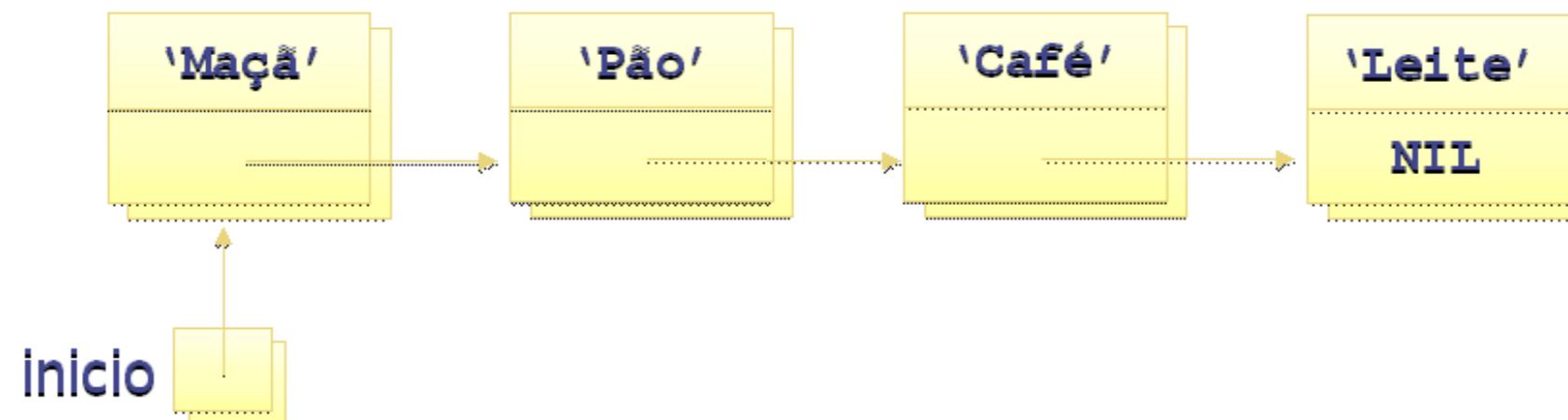
- Criar lista
- Limpar lista
- Inserir item (última posição)
- Remover item (última posição)
- Recuperar item (dado uma chave)
- Recuperar item (por posição)
- Contar número de itens
- Verificar se a lista está vazia
- Verificar se a lista está cheia
- Imprimir lista

TAD Listas e Listas Ligadas

- Antes de começarmos, precisamos definir como a lista será representada
- Uma forma bastante comum é manter uma variável ponteiro para o primeiro elemento da lista ligada

TAD Listas e Listas Ligadas

- Antes de começarmos, precisamos definir como a lista será representada
- Uma forma bastante comum é manter uma variável ponteiro para o primeiro elemento da lista ligada



TAD Listas e Listas Ligadas

- Convenciona-se que essa variável ponteiro deve ter valor **NULL** quando a lista estiver vazia
- Portanto, essa deve ser a iniciação da lista e também a forma de se verificar se ela se encontra vazia

TAD Listas e Listas Ligadas

- Outro detalhe importante é quanto as posições
 - Na implementação com vetores, uma posição é um valor inteiro entre 0 e o campo **fim**
 - Com listas ligadas, uma posição passa ser um ponteiro que aponta um determinado nó da lista
- Vamos analisar cada uma das operações do TAD Lista

TAD Listas I

Criar lista

- Pré-condição: nenhuma
- Pós-condição: inicia a estrutura de dados

Limpar lista

- Pré-condição: nenhuma
- Pós-condição: coloca a estrutura de dados no estado inicial

TAD Listas II

Inserir item (última posição)

- Pré-condição: **existe memória disponível**
- Pós-condição: insere um item na última posição, retorna **true** se a operação foi executada com sucesso, **false** caso contrário

Remover item (última posição)

- Pré-condição: a lista não está vazia
- Pós-condição: o último item da lista é removido, retorna **true** se a operação foi executada com sucesso,

TAD Listas III

Remover item (por posição)

- Pré-condição: uma posição válida da lista é informada
- Pós-condição: o item na posição fornecida é removido da lista, retorna **true** se a operação foi executada com sucesso, **false** caso contrário

Recuperar item (dado uma chave)

- Pré-condição: nenhuma
- Pós-condição: recupera o item dado uma chave, retorna **true** se a operação foi executada com sucesso, **false** caso contrário

TAD Listas IV

Recuperar item (por posição)

- Pré-condição: uma posição válida da lista é informada
- Pós-condição: recupera o item na posição fornecida, retorna **true** se a operação foi executada com sucesso, **false** caso contrário

Contar número de itens

- Pré-condição: nenhuma
- Pós-condição: retorna o número de itens na lista

TAD Listas V

Verificar se a lista está vazia

- Pré-condição: nenhuma
- Pós-condição: retorna **true** se a lista estiver vazia e **false** caso-contrário

Verificar se a lista está cheia (???)

- Pré-condição: nenhuma
- Pós-condição: retorna **true** se a lista estiver cheia e **false** caso-contrário

TAD Listas VI

Imprime lista

- Pré-condição: nenhuma
- Pós-condição: imprime na tela os itens da lista

Lista Ligada

- Para se criar uma lista ligada, é necessário criar um **nó** que possua um ponteiro para outro **nó**

```
1  typedef struct {  
2      int chave;  
3      int valor;  
4  } ITEM;  
  
5  
6  typedef struct NO {  
7      ITEM item;  
8      struct NO *proximo;  
9  } NO;
```

Lista Ligada

- Considerando a estrutura NO, para a definição da lista ligada o que falta é a indicação da posição de memória do primeiro nó
- Também incluiremos a posição para o último nó para acelerar a inserção de itens no final da lista

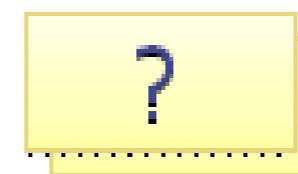
```
1 typedef struct {  
2     NO *inicio;  
3     NO *fim;  
4 } LISTA_LIGADA;
```

Criar lista

- Pré-condição: nenhuma
- Pós-condição: inicia a estrutura de dados

Antes

inicio



Depois

inicio

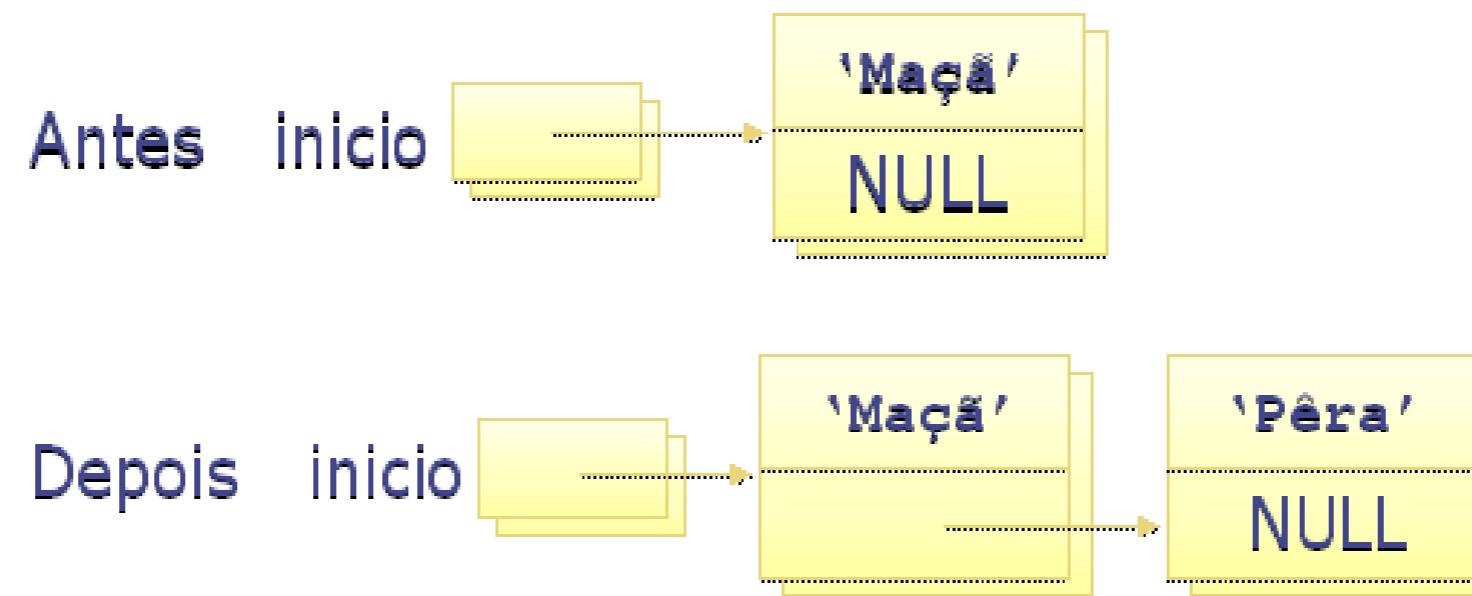


Criar lista

```
1 void criar(struct LISTA_LIGADA *lista){  
2     lista->inicio = NULL;  
3     lista->fim = NULL;  
4 }
```

Inserir item (última posição)

- Pré-condição: existe memória disponível
- Pós-condição: insere um item na última posição, retorna **true** se a operação foi executada com sucesso, **false** caso contrário



Memória Disponível

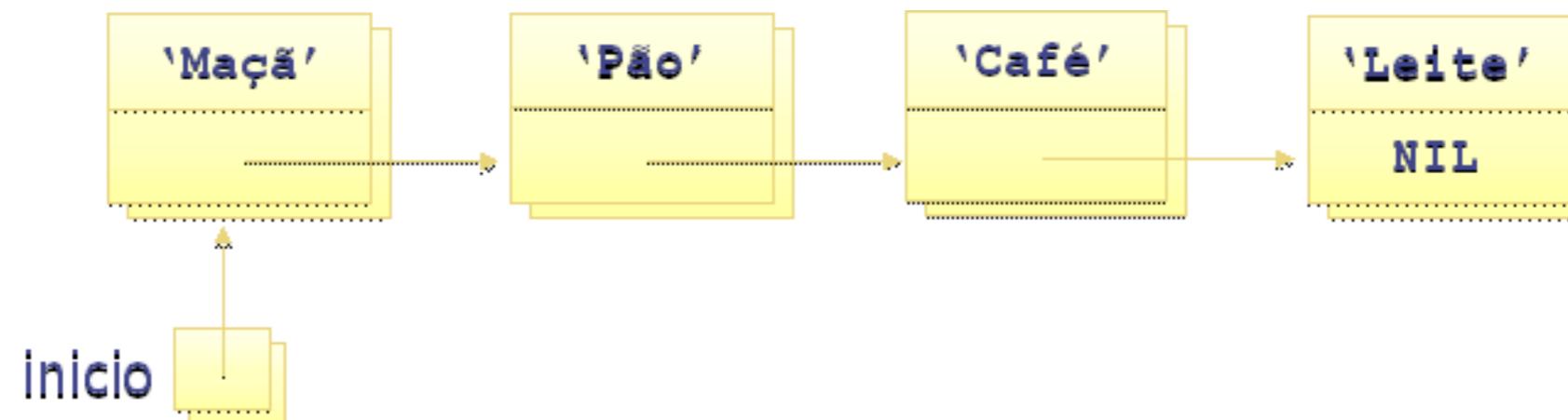
- Diferente da implementação com vetores, a lista ligada não requer especificar um tamanho para a estrutura
- Entretanto, a memória *heap* não é ilimitada e é sempre importante verificar se existe memória disponível ao chamar **malloc()**
- Em C, o procedimento **malloc()** atribui o valor **NULL** à variável ponteiro quando não existe memória disponível

Inserir item (última posição)

```
1 int inserir(LISTA_LIGADA *lista, ITEM *item) {
2     NO *pnovo = (NO *)malloc(sizeof(NO)); //cria um novo nó
3
4     if (pnovo != NULL) { //verifica se a memória foi alocada
5         pnovo->item = *item; //preenche os dados
6         pnovo->proximo = NULL; //define que o próximo é nulo
7
8         if (lista->inicio == NULL) { //se a lista for vazia
9             lista->inicio = pnovo; //inicio aponta para novo
10        } else {
11            lista->fim->proximo = pnovo; //proximo do fim aponto para novo
12        }
13
14        lista->fim = pnovo; //fim aponta para novo
15
16        return 1;
17    } else {
18        return 0;
19    }
20}
```

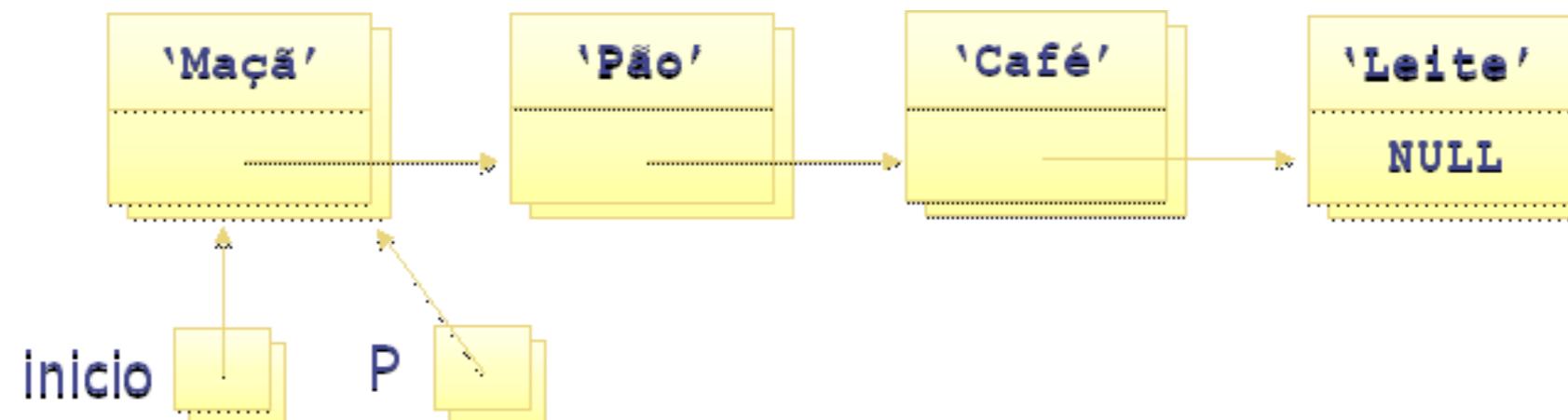
Recuperar item (dado uma chave)

- **Pré-condição:** nenhuma
- **Pós-condição:** recupera o item dado uma chave, retorna **true** se a operação foi executada com sucesso, **false** caso contrário



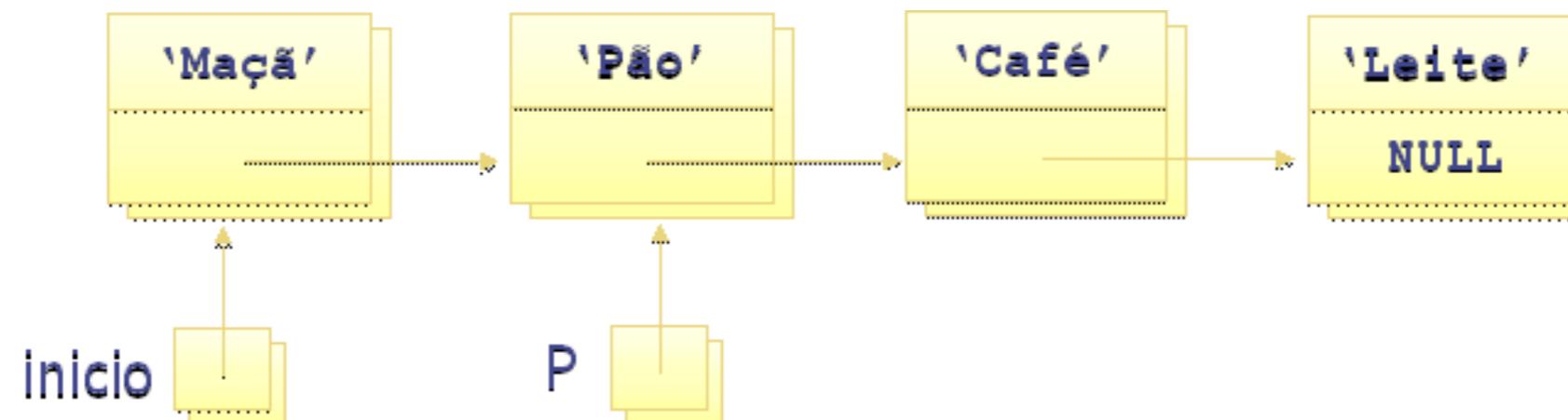
Recuperar item (dado uma chave)

- **Pré-condição:** nenhuma
- **Pós-condição:** recupera o item dado uma chave, retorna **true** se a operação foi executada com sucesso, **false** caso contrário



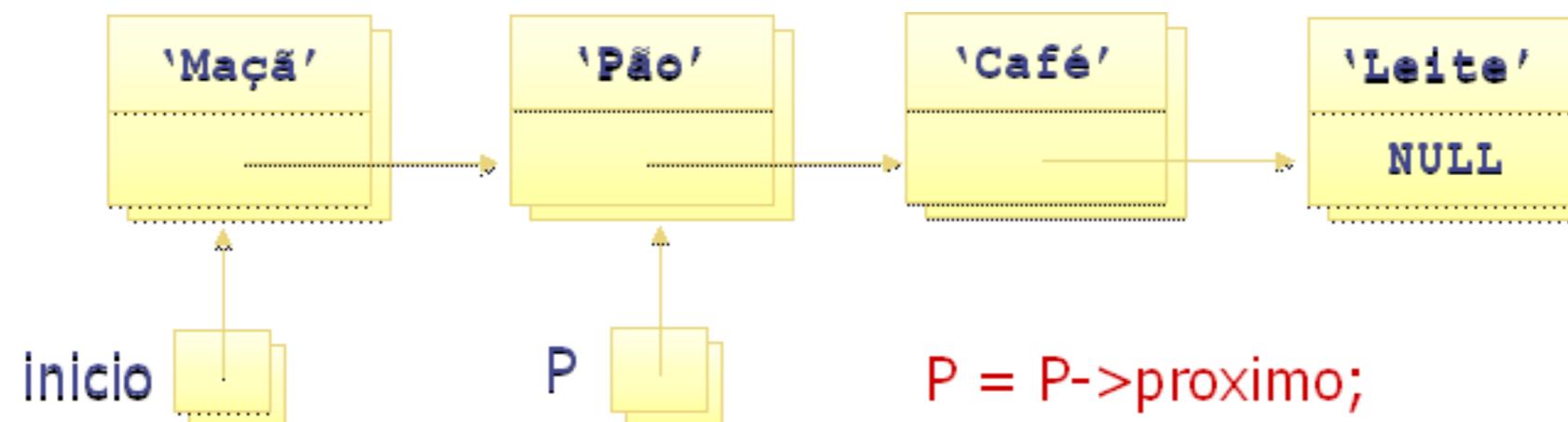
Recuperar item (dado uma chave)

- **Pré-condição:** nenhuma
- **Pós-condição:** recupera o item dado uma chave, retorna **true** se a operação foi executada com sucesso, **false** caso contrário



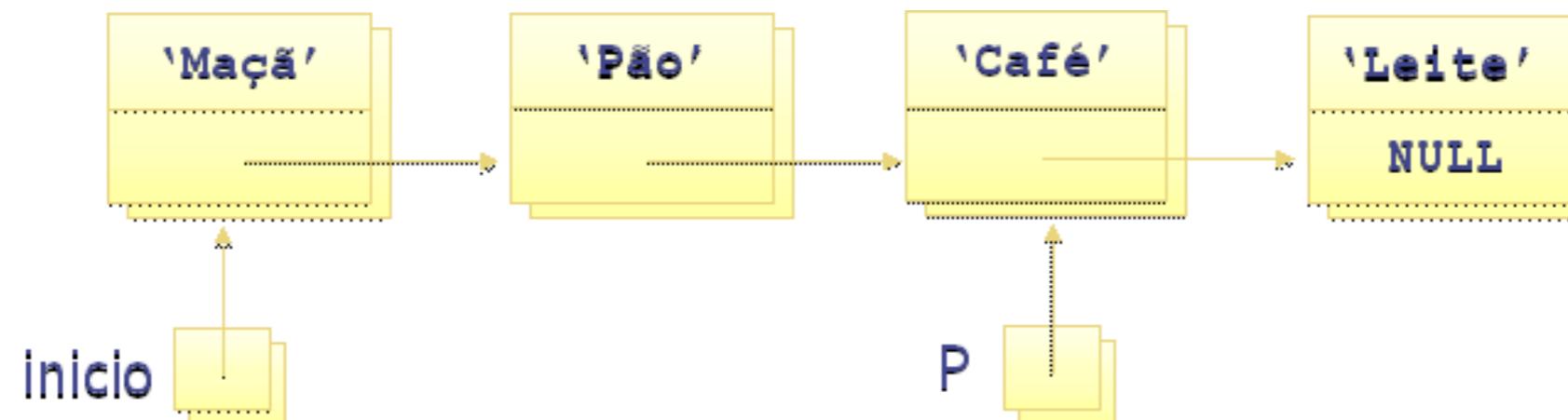
Recuperar item (dado uma chave)

- **Pré-condição:** nenhuma
- **Pós-condição:** recupera o item dado uma chave, retorna **true** se a operação foi executada com sucesso, **false** caso contrário



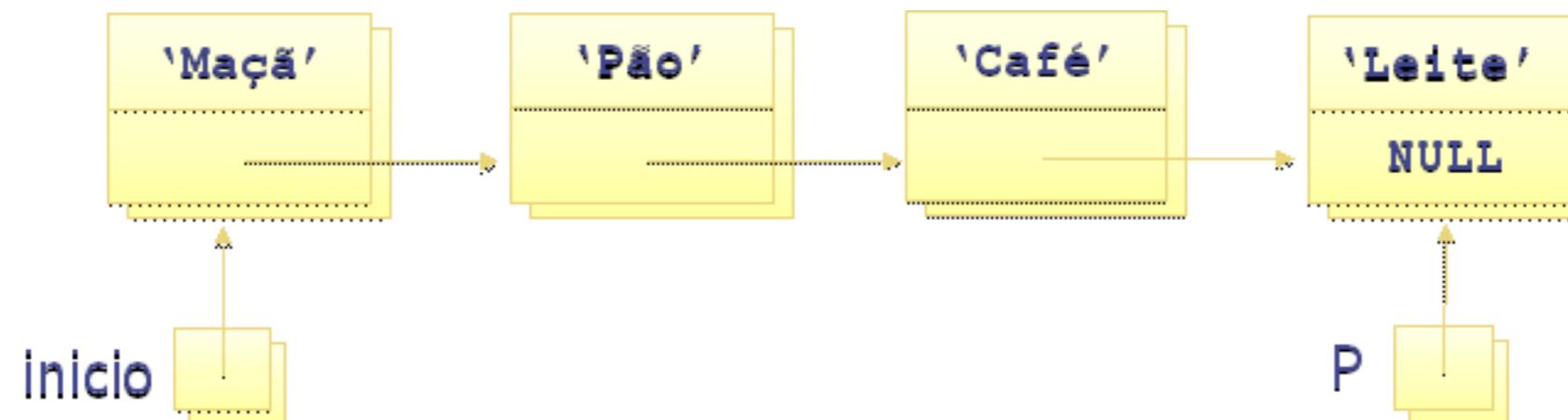
Recuperar item (dado uma chave)

- **Pré-condição:** nenhuma
- **Pós-condição:** recupera o item dado uma chave, retorna **true** se a operação foi executada com sucesso, **false** caso contrário



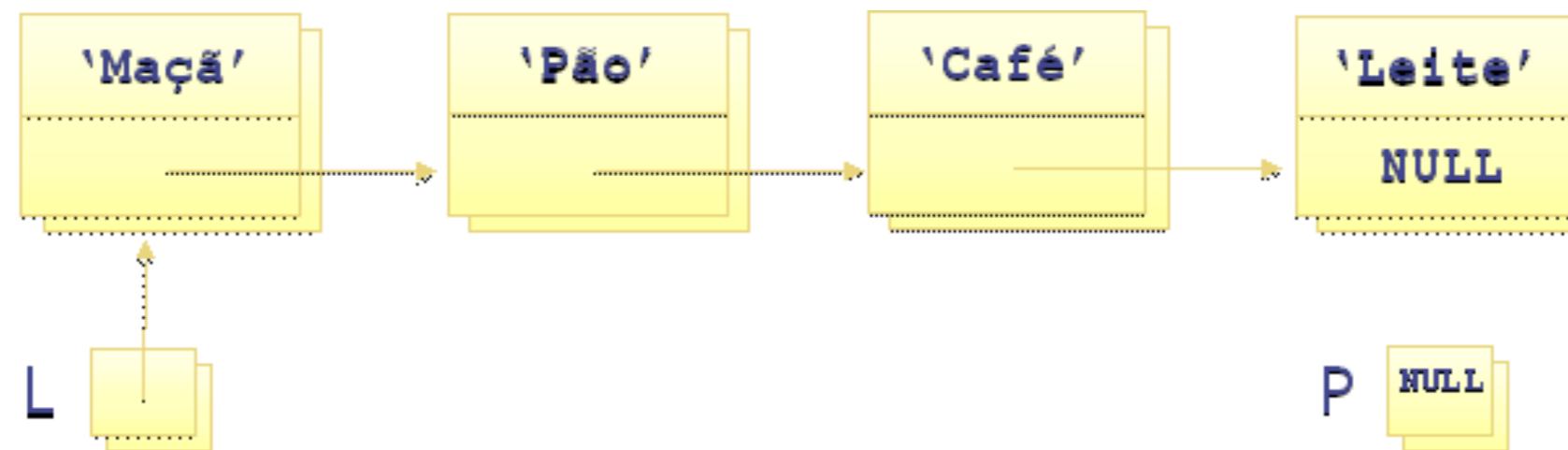
Recuperar item (dado uma chave)

- **Pré-condição:** nenhuma
- **Pós-condição:** recupera o item dado uma chave, retorna **true** se a operação foi executada com sucesso, **false** caso contrário



Recuperar item (dado uma chave)

- **Pré-condição:** nenhuma
- **Pós-condição:** recupera o item dado uma chave, retorna **true** se a operação foi executada com sucesso, **false** caso contrário



Recuperar item (dado uma chave)

```
1 int buscar(LISTA_LIGADA *lista, int chave, ITEM *item) {
2     NO *paux = lista->inicio;
3
4     while (paux != NULL) {
5         if (paux->item.chave == chave) {
6             *item = paux->item;
7             return 1;
8         }
9         paux = paux->proximo;
10    }
11
12    return 0;
13 }
```

Verificar se a lista está vazia

- **Pré-condição:** nenhuma
- **Pós-condição:** retorna **true** se a lista estiver vazia e **false** caso-contrário

```
1 int vazia(LISTA_LIGADA *lista) {  
2     return (lista->inicio == NULL);  
3 }
```

Remover item (última posição)

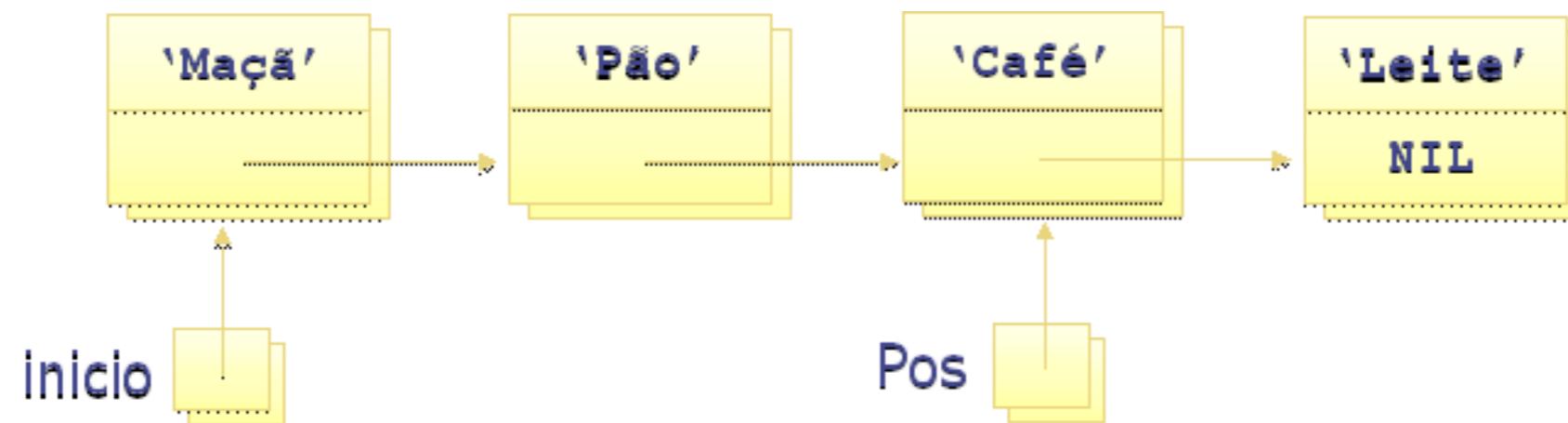
- **Pré-condição:** a lista não está vazia
- **Pós-condição:** o último item da lista é removido, retorna **true** se a operação foi executada com sucesso, **false** caso contrário

Remover item (última posição)

```
1 int remover_fim(LISTA_LIGADA *lista) {
2     if (!vazia(lista)) {
3         //procura o penúltimo nó
4         NO *paux = lista->inicio;
5         while (paux->proximo != NULL && paux->proximo != lista->fim) {
6             paux = paux->proximo;
7         }
8
9         if (lista->inicio == lista->fim) {//a lista tem um nó
10            free(paux->proximo); //libera o único nó
11            lista->inicio = lista->fim = NULL; //lista vazia
12        } else {
13            free(lista->fim); //libera último nó
14            lista->fim = paux; //penúltimo nó vira último
15            lista->fim->proximo = NULL;
16        }
17
18        return 1;
19    } else {
20        return 0;
21    }
22 }
```

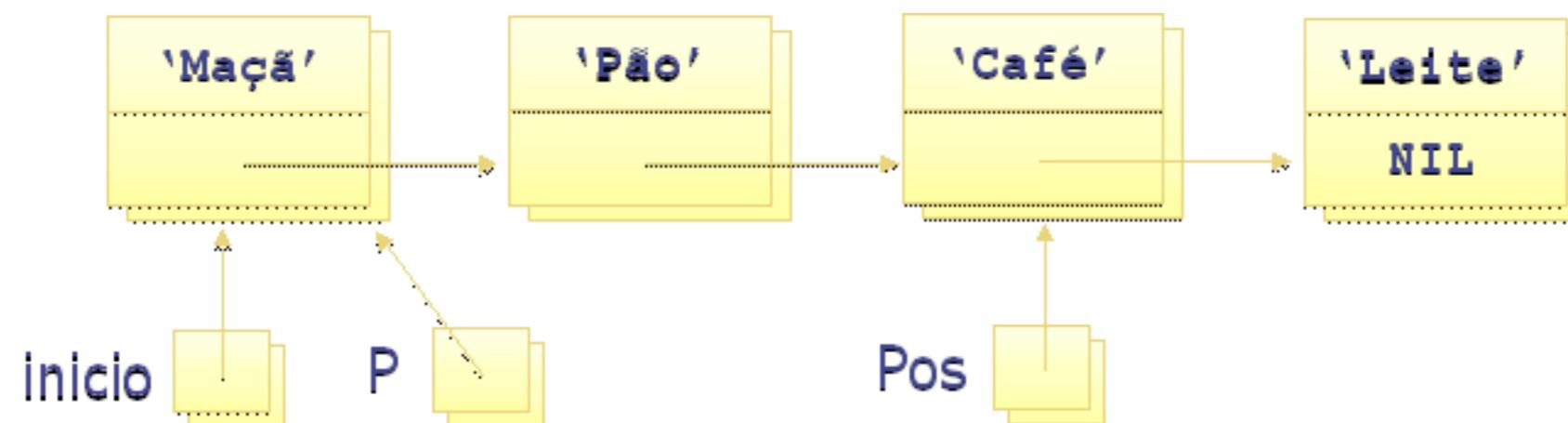
Remover item (por posição)

- **Pré-condição:** uma posição válida da lista é informada
- **Pós-condição:** o item na posição fornecida é removido da lista, retorna **true** se a operação foi executada com sucesso, **false** caso contrário



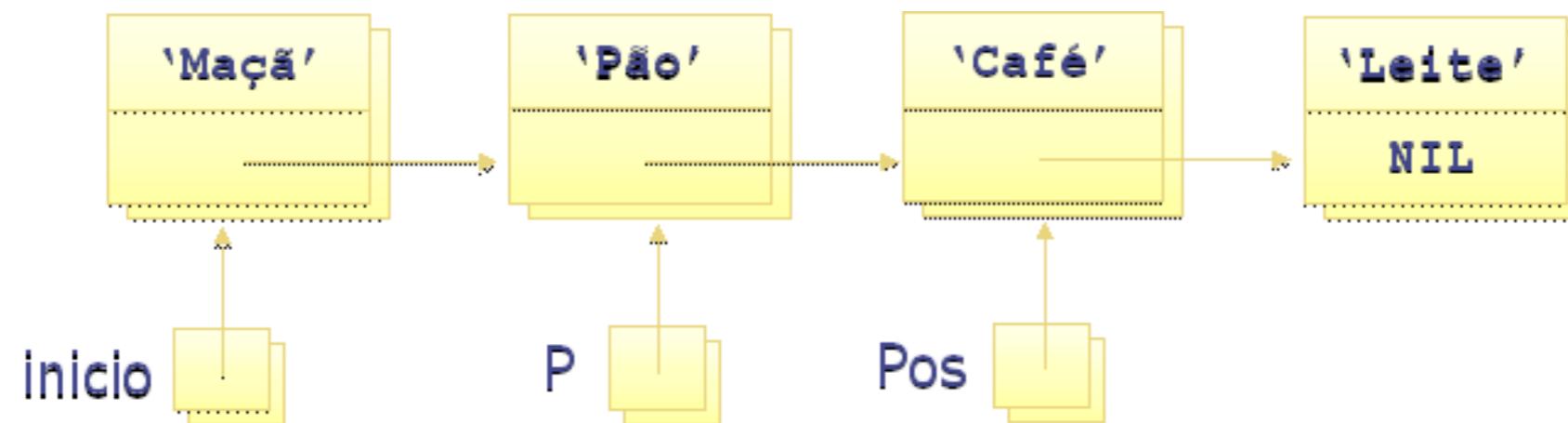
Remover item (por posição)

- **Pré-condição:** uma posição válida da lista é informada
- **Pós-condição:** o item na posição fornecida é removido da lista, retorna **true** se a operação foi executada com sucesso, **false** caso contrário



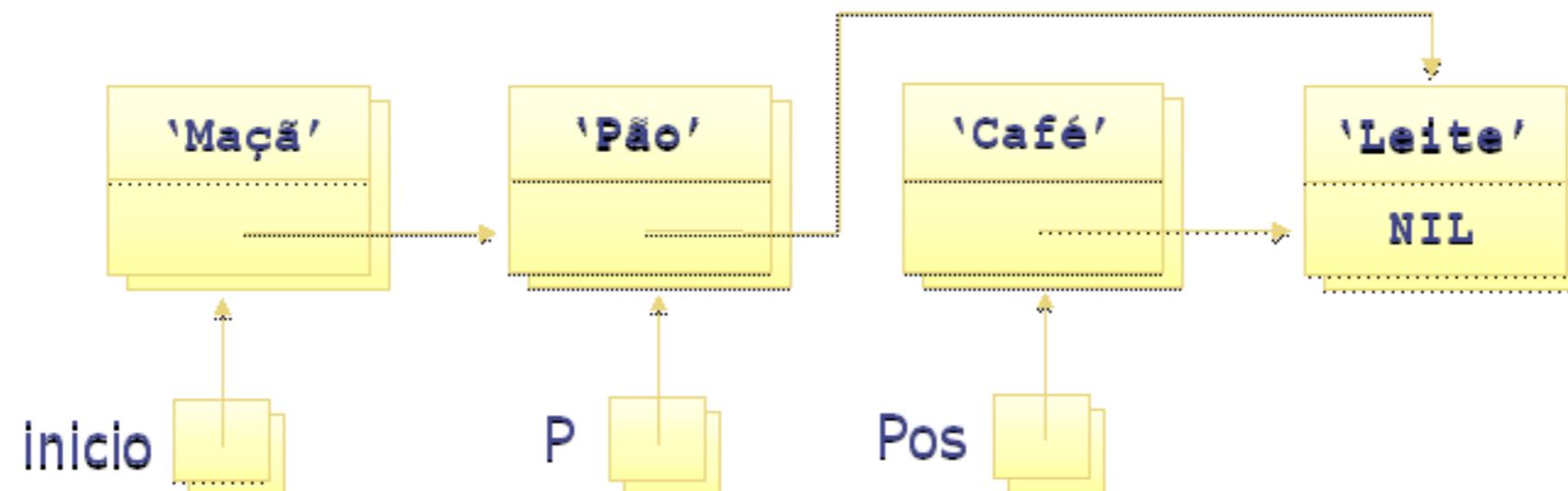
Remover item (por posição)

- **Pré-condição:** uma posição válida da lista é informada
- **Pós-condição:** o item na posição fornecida é removido da lista, retorna **true** se a operação foi executada com sucesso, **false** caso contrário



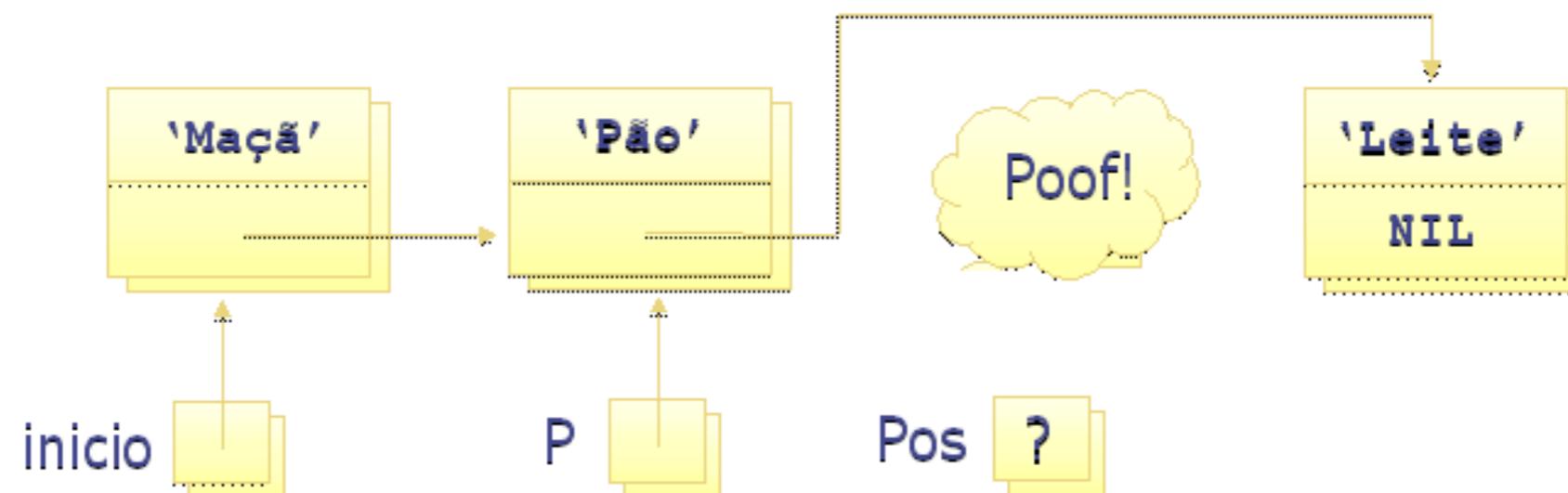
Remover item (por posição)

- **Pré-condição:** uma posição válida da lista é informada
- **Pós-condição:** o item na posição fornecida é removido da lista, retorna **true** se a operação foi executada com sucesso, **false** caso contrário



Remover item (por posição)

- **Pré-condição:** uma posição válida da lista é informada
- **Pós-condição:** o item na posição fornecida é removido da lista, retorna **true** se a operação foi executada com sucesso, **false** caso contrário



Exercício

Implementar as demais operações do TAD Listas

- Limpar lista
- Inserir item (por posição)
- Remover item (por posição)
- Recuperar item (por posição)
- Contar número de itens
- Imprimir lista