MCTA028-15: Programação Estruturada

Aula 12: Listas
(Segunda Parte)

Wagner Tanaka Botelho
wagner.tanaka@ufabc.edu.br / wagtanaka@gmail.com
Universidade Federal do ABC (UFABC)
Centro de Matemática, Computação e Cognição (CMCC)

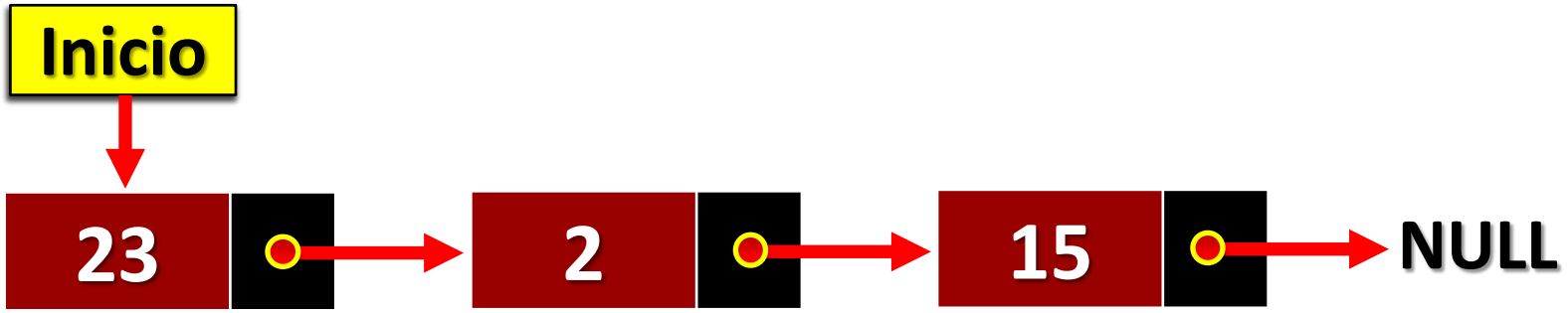
Introdução

Introdução

- Uma lista dinâmica encadeada é uma lista definida utilizando alocação dinâmica e acesso encadeado dos elementos;
- Cada elemento da lista é alocado dinamicamente, á medida que os dados são inseridos dentro da lista, e tem sua memória liberada, à medida que é removido;
- Cada elemento é um ponteiro para uma estrutura contendo dois campos de informação:
  - CAMPO DE DADO: é utilizado para armazenar a informação inserida na lista;
  - CAMPO PROX: é um ponteiro que indica o próximo elemento na lista.

Introdução

- A lista dinâmica encadeada utiliza um PONTEIRO para PONTEIRO para guardar o **PRIMEIRO** elemento da lista:
  - O ponteiro para ponteiro é utilizado para representar o **INÍCIO** da lista.
- **TODOS** os elementos são ponteiros alocados dinamicamente.



Introdução

- **Vantagens**:
  - Melhor utilização dos recursos de memória;
  - Não é preciso definir previamente o tamanho da lista;
  - Não é necessário movimentar os elementos nas operações de inserção e remoção.
- **Desvantagens**:
  - Acesso indireto aos elementos;
  - Necessidade de percorrer a lista para acessar determinado elemento;

Definindo o Tipo

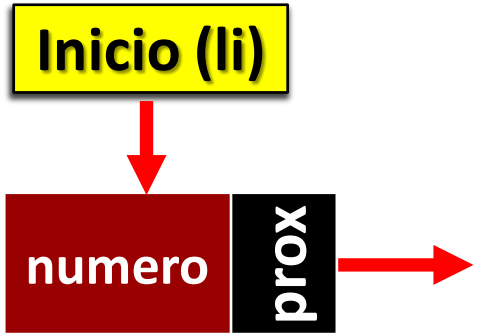
*Ex_03.c

```c
#include<stdio.h>
#include<stdlib.h>

struct elemento{
    int numero;
    struct elemento *prox;
};

typedef struct elemento Elem;

typedef struct elemento* Lista;
```

Definindo o tipo que descreve cada elemento da lista.
+ *prox: ponteiro para o próximo elemento da lista;
+ numero: tipo de dado (int) a ser armazenado na lista.

Redefinindo a *struct* para encurtar o comando.

Lista *li; → **É um ponteiro para Lista que já é um ponteiro para a struct elemento. Portanto, li é um ponteiro para ponteiro. Por ser ponteiro para ponteiro, li armazena o endereço de um ponteiro.**



Criando a Lista

```
13  Lista* cria_Lista(){
14      Lista *li;
15
16      li = (Lista*) malloc(sizeof(Lista));
17
18      if(li != NULL){
19          *li = NULL;
20      }
21      return li;
22  }
```

Alocando uma área de memória para armazenar o endereço do início da lista.
li é um ponteiro para ponteiro

O conteúdo de li é NULL, indicando que não existe nenhum elemento alocado após o atual, ou seja, a lista esta vazia.

Watches

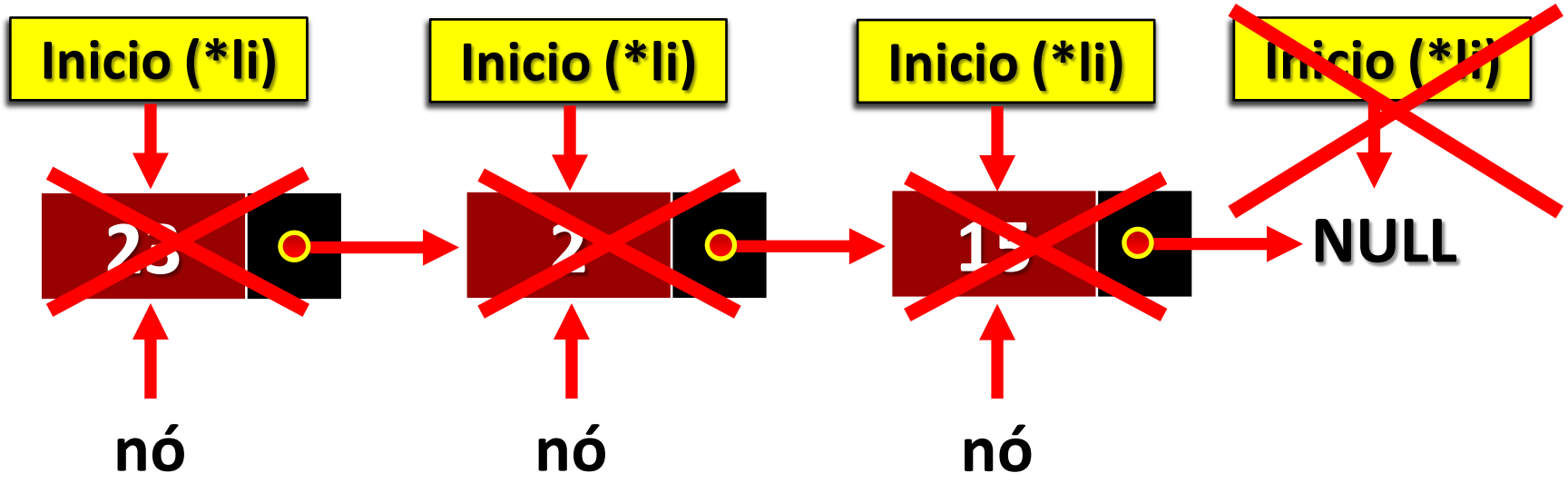
Function argument		
Locals		
li	0x941720	
*li	(Lista) 0x0	Lista

“Destruindo” a Lista

```
24  void libera_lista(Lista *li){
25      if(li != NULL){
26          Elem *no;
27          while((*li) != NULL){
28              no = *li;
29              *li = (*li)->prox;
30              free(no);
31          }
32          free(li);
33      }
34  }
```

Se a lista foi criada com sucesso.

Cada elemento da lista é percorrido, enquanto conteúdo do INÍCIO da lista for diferente de NULL.



Tamanho da Lista

Tamanho da Lista

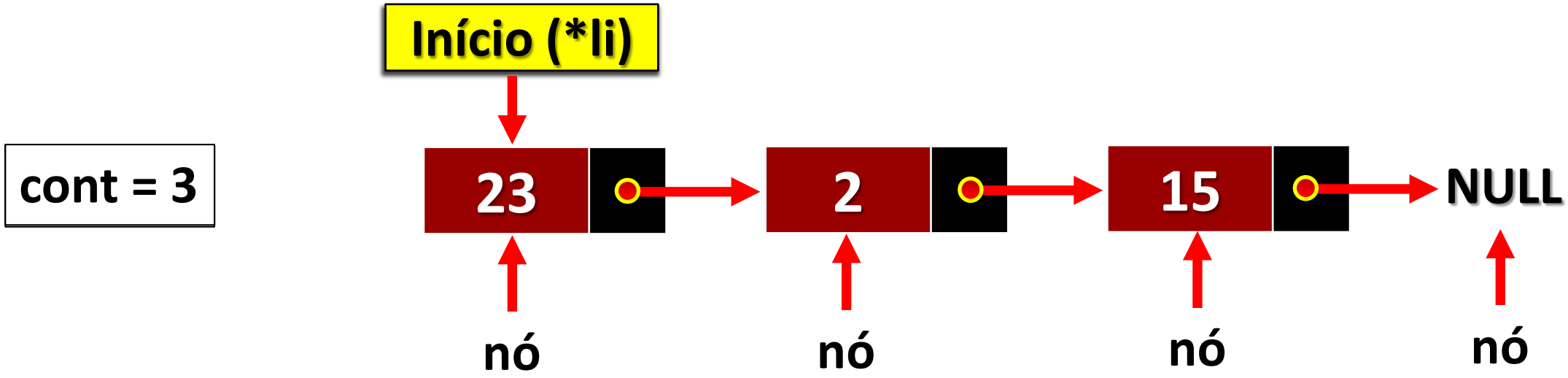
- Diferente da lista sequencial estática, para saber o tamanho da lista dinâmica, é preciso PERCORRER toda a lista, CONTANDO os elementos inseridos nela, até encontrar o seu final.

```
36  int tamanho_lista(Lista *li){
37      int cont = 0;
38      if(li == NULL){
39          return 0;
40      }
41      else{
42          Elem *no = *li;
43          while(no != NULL){
44              cont++;
45              no = no->prox;
46          }
47          return cont;
48      }
49  }
```

Verifica se a lista foi criada com sucesso.

Nó aponta para o início da lista.

Percorre a lista até o valor do nó for diferente de NULL (fim da lista).



Lista Cheia

Lista Cheia

- Na lista dinâmica encadeada somente será considerada **CHEIA** quando **NÃO** tiver mais memória disponível para alocar novos elementos:
  - Apenas ocorrerá quando a chamada da função **`malloc()`** retornar **NULL**.

Lista Vazia

Lista Vazia

- Uma lista dinâmica encadeada é considerada **VAZIA** sempre que o conteúdo do seu "**INÍCIO**" apontar para a constante **NULL**.

```
87  int lista_vazia(Lista *li){
88      if(li == NULL){
89          return 1;
90      }
91      else{
92          if(*li == NULL){
93              return 1;
94          }
95          else{
96              return 0;
97          }
98      }
99  }
```

Verifica se a lista foi criada com sucesso.

Acessa o conteúdo do início (*li) para comparar com NULL.

Se a lista estiver vazia.

Lista não vazia.

Inserindo no Início da Lista

Inserindo no Início da Lista

- Diferente da lista sequencial estática, a inserção no inicio de uma lista dinâmica encadeada **NÃO** necessita que se mude o lugar dos demais elementos da lista;
- Basicamente, deve-se alocar espaço para o novo elemento da lista e mudar os valores de alguns ponteiros.

```
69  int insere_lista_inicio(Lista *li, int n){
70      if(li == NULL){
71          return 0;
72      }
73      else{
74          Elem *no;
75          no  = (Elem*) malloc(sizeof(Elem));
76
77          if(no == NULL){
78              return 0;
79          }
80          else{
81              no->numero = n;
82              no->prox = (*li);
83              *li = no;
84              return 1;
85          }
86      }
87  }
```

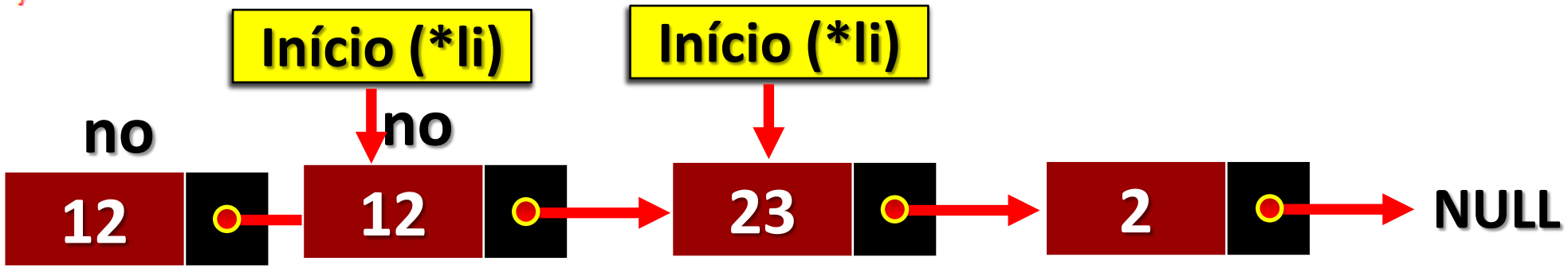
Verifica se a lista foi criada com sucesso.

Se não foi possível alocar memória.

Copiando (para o nó) o número recebido como parâmetro.

Nó está apontando para o início (*li).

Conteúdo do início da lista (*li) recebe o nó.



Inserindo no Final da Lista

Inserindo no Final da Lista

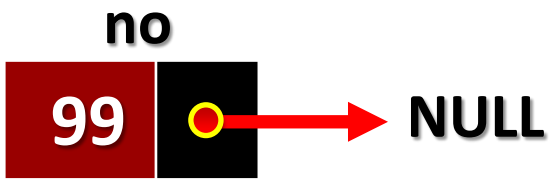
- Como na inserção no início, a INSERÇÃO no FINAL de uma lista dinâmica encadeada não necessita que se mude o lugar dos demais elementos da lista:
  - Entretanto, é preciso percorrer a lista toda para descobrir o **ÚLTIMO** elemento e assim fazer a inserção após ele.

```
100         else{
101             no->numero = n;
102             no->prox = NULL;
103             if((*li) == NULL){
104                 *li = no;
105             }
106             else{
107                 Elem *aux;
108                 aux = *li;
109                 while(aux->prox != NULL){
110                     aux = aux->prox;
111                 }
112                 aux->prox = no;
113             }
114             return 1;
115         }
```

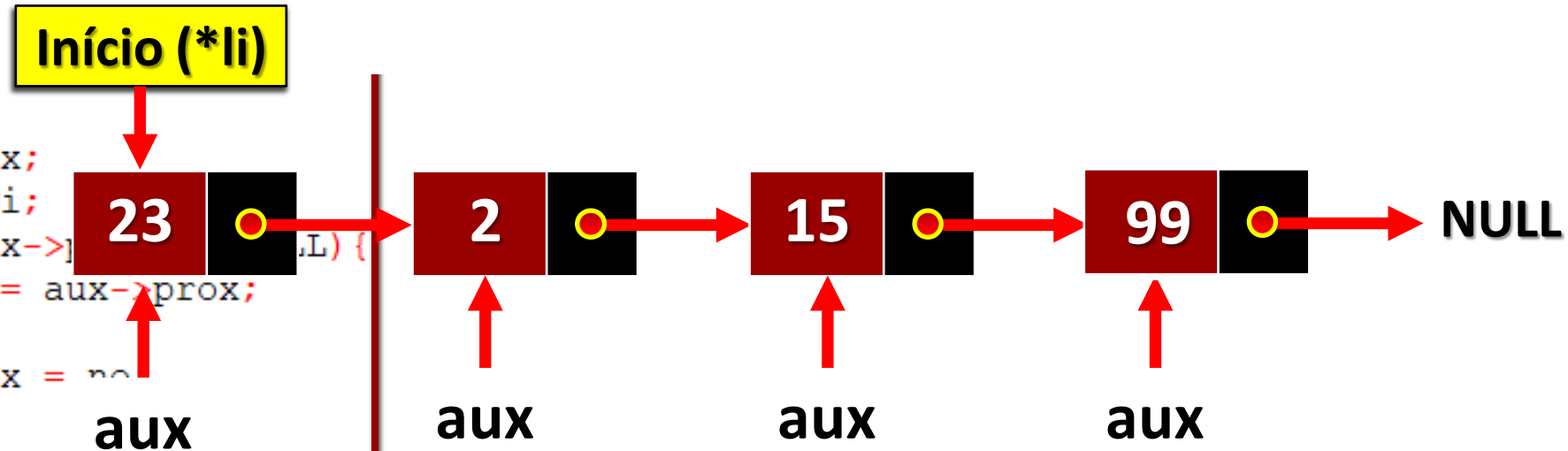
Conteúdo de li está apontando para NULL (lista vazia)?

Verificar se a inserção é possível.

Ponteiro aux recebe o endereço da primeira posição da lista (li).




```
106             else{
107                 Elem *aux;
108                 aux = *li;
109                 while(aux->prox != NULL){
110                     aux = aux->prox;
111                 }
112                 aux->prox = no;
113             }
114             return 1;
115         }
116     }
117 }
```



Removendo do Início da Lista

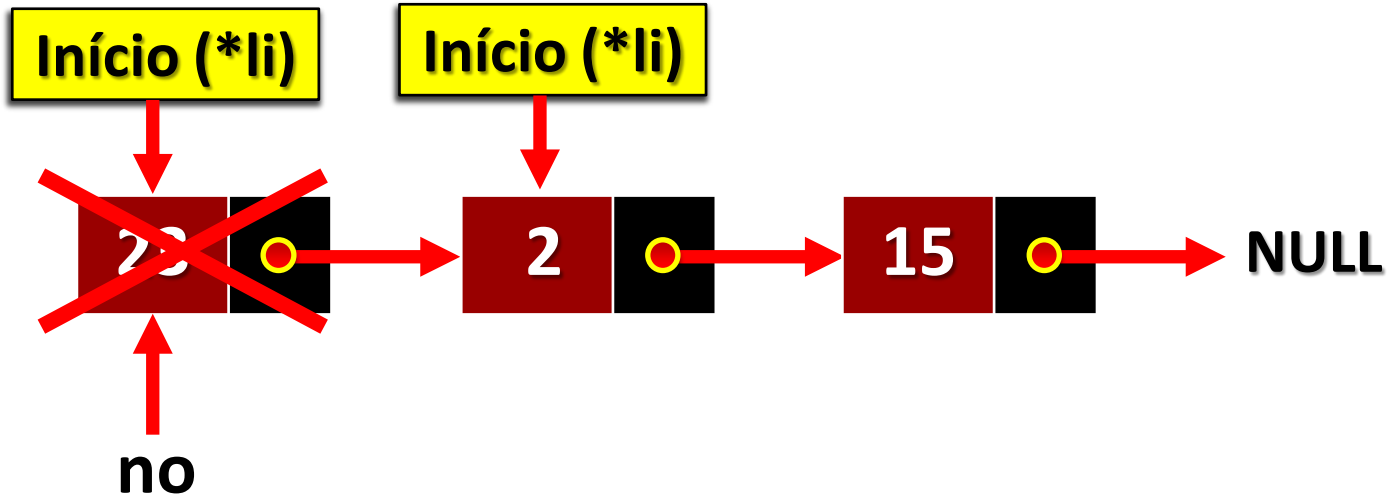
Removendo do Início da Lista

- Diferente da lista sequencial estática, a REMOÇÃO do início de uma lista dinâmica encadeada **NÃO** necessita que se mude o lugar dos demais elementos da lista.

```
119  int remove_lista_inicio(Lista *li){
120      if(li == NULL){
121          return 0;
122      }
123      else if((*li) == NULL){
124          return 0;
125      }
126      else{
127          Elem *no;
128          no = *li;
129          *li = no->prox;
130          free(no);
131          return 1;
132      }
133  }
```

Verificar se a remoção é possível.

Ponteiro no recebe o endereço da primeira posição da lista (li).



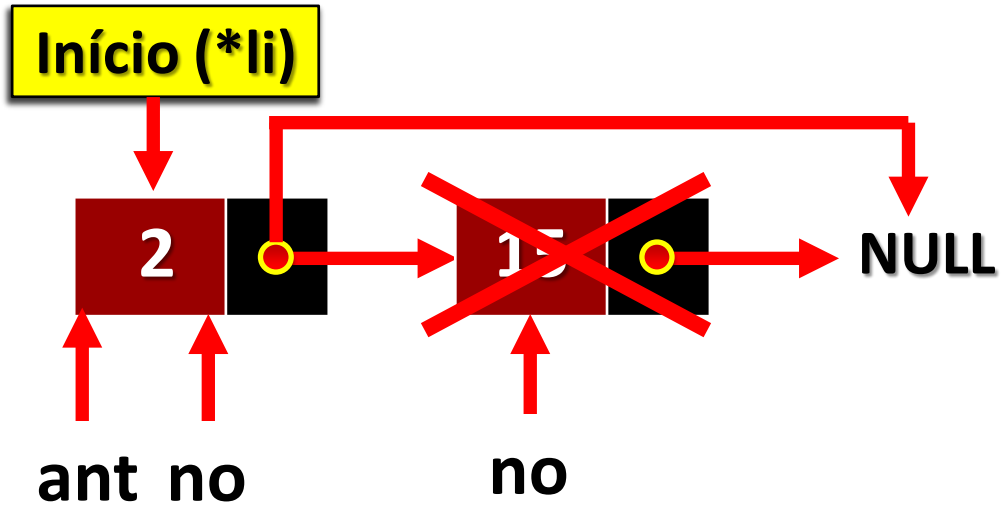
Removendo do Fim da Lista

```
142     else{
143         Elem *ant, *no;
144         no = *li;
145         while(no->prox != NULL){
146             ant = no;
147             no = no->prox;
148         }
149         if(no == (*li)){
150             *li = no->prox;
151         }
152         else{
153             ant->prox = no->prox;
154         }
155         free(no);
156         return 1;
157     }
```

```
151         }
152         else{
153             ant->prox = no->prox;
154         }
155         free(no);
156         return 1;
157     }
158 }
```

- Verificar se a remoção é possível.

➡ Último elemento da lista pode ser o único.



Removendo um Elemento Específico da Lista

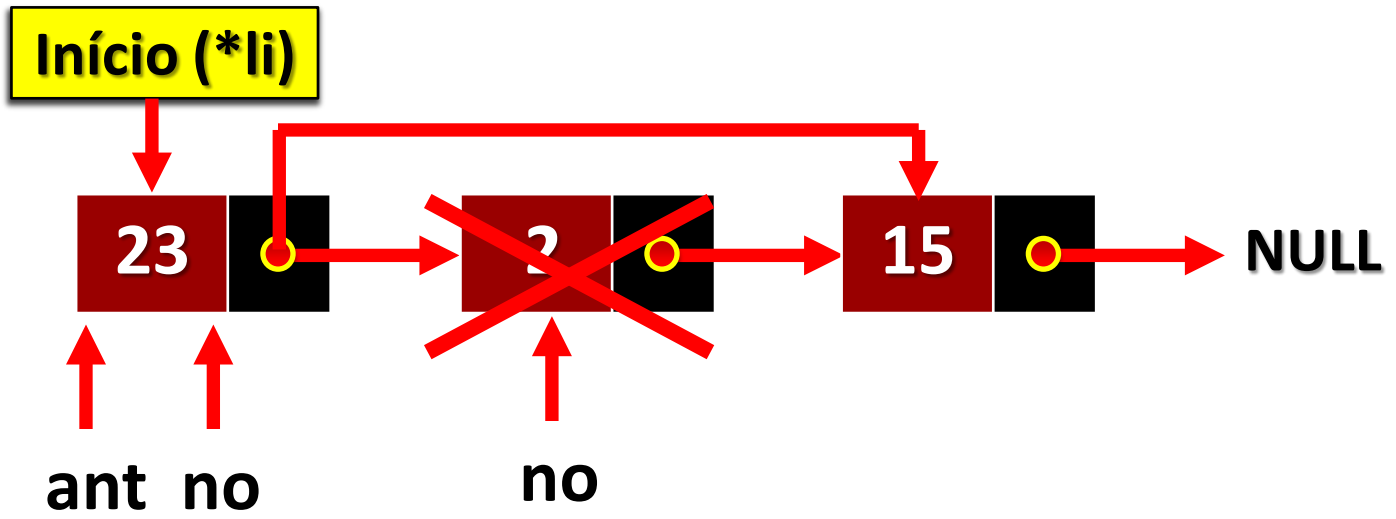
```
167     else{
168         Elem *ant, *no;
169         no = *li;
170         while(no != NULL && no->numero != elem){
171             ant = no;
172             no = no->prox;
173         }
174         if(no == NULL){
175             return 0;
176         }
177         else{
178             if(no == *li){
179                 *li = no->prox;
180             }
181             else{
182                 ant->prox = no->prox;
183             }
184             free(no);
185             return 1;
186         }
187     }
183             }
184             free(no);
185             return 1;
186         }
187     }
188 }
```

emoção é possível.

Não encontrou o elemento (elem) na lista.

Remover o primeiro elemento da lista.

remove_lista(ptList, 2);



Busca por Posição na Lista

```
190 int busca_lista_posicao(Lista *li, int pos){
191     if(li == NULL || pos <= 0){
192         exit(0);
193     }
194     Elem *no;
195     no = *li;
196     int i = 1, resp = 0;
197
198     while(no != NULL && i< pos){
199         no = no->prox;
200         i++;
201     }
202     if(no == NULL){
203         exit(0);
204     }
205     else{
206         resp = no->numero;
207         return resp;
208     }
209 }
```

A lista está vazia ou a posição é negativa?

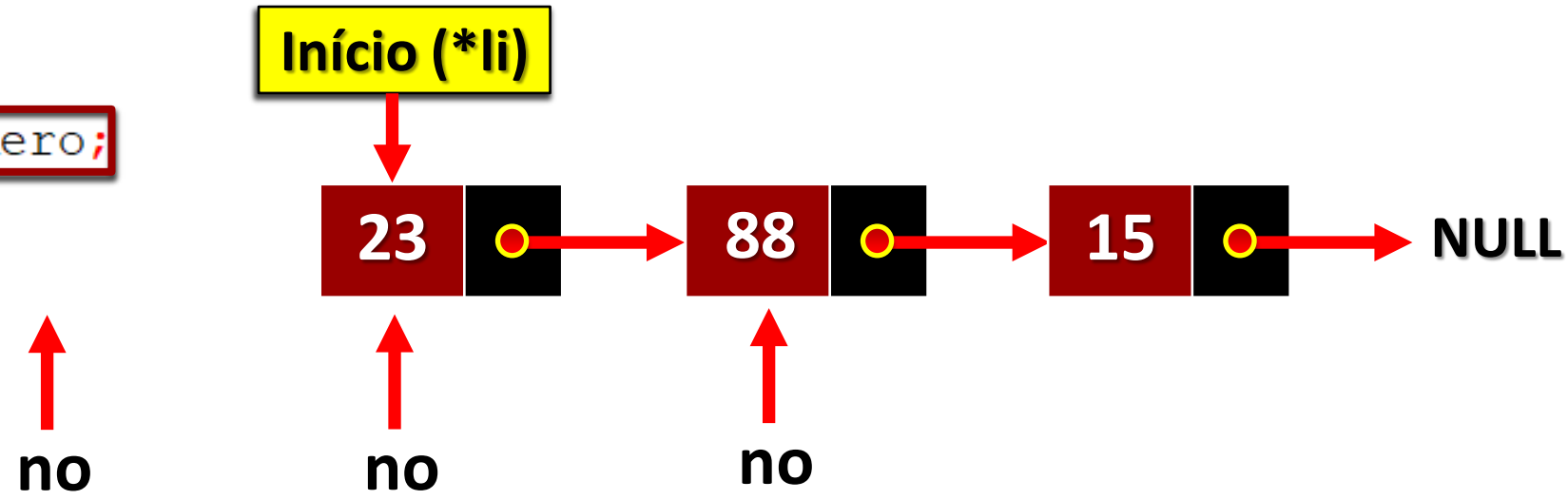
Busca_lista_posicao(ptList, 1);

pos = 2;

i = 2

Não encontrou o elemento na lista.

resp = 88;



Busca por Conteúdo na Lista

```
211 int busca_lista_conteudo(Lista *li, int conte){
212     if(li == NULL){
213         exit(0);
214     }
215     Elem *no;
216     no = *li;
217     int i = 1;
218
219     while(no != NULL && no->numero != conte){
220         no = no->prox;
221         i++;
222     }
223     if(no == NULL){
224         exit(0);
225     }
226     else{
227         return i;
228     }
229 }
```

Se aconteceu algum problema na criação da lista.

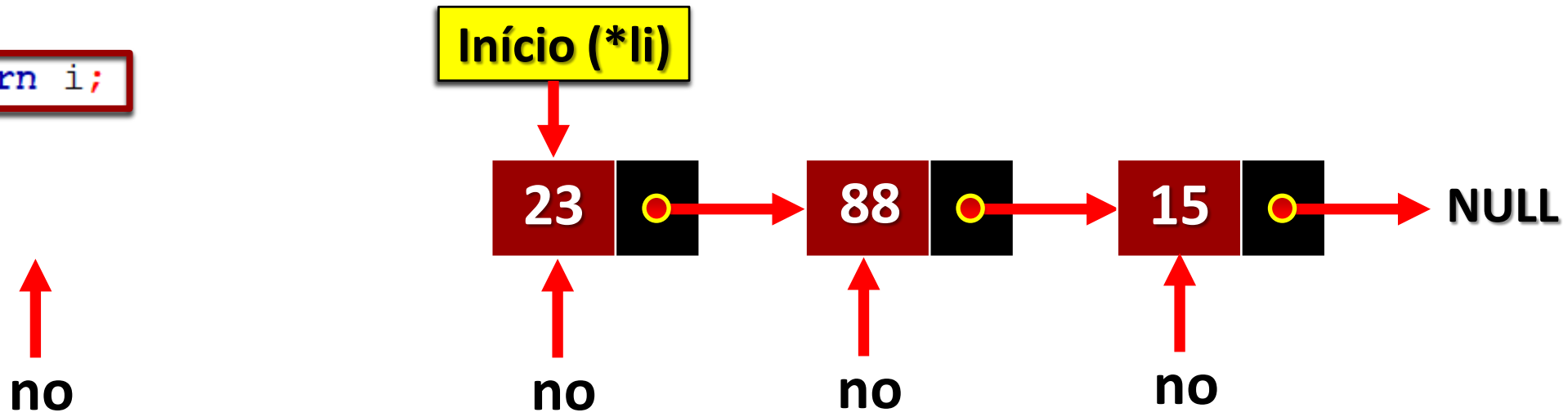
Busca_lista_conteudo(ptList, 15);

conte = 15;

i = 3

Retorna a posição (i=3) que se encontra o elemento buscado.

Não encontrou o elemento na lista.



Referências

- Slides do Prof. Luiz Rozante;
- SALES, André Barros de; AMVAME-NZE, Georges. Linguagem C: roteiro de experimentos para aulas práticas. 2016;
- BACKES, André. Linguagem C Completa e Descomplicada. Editora Campus. 2013;
- SCHILDT, Herbert. C Completo e Total. Makron Books. 1996;
- DAMAS, Luís. Linguagem C. LTC Editora. 1999;
- DEITEL, Paul e DEITEL, Harvey. C Como Programar. Pearson. 2011;
- BACKES, André. Estrutura de Dados Descomplicada em Linguagem C. GEN LTC. 2016.